UiT The Arctic University of Norway

Faculty of Science and Technology

Department of Mathematics and Statistics

Trans-dimensional inference over Bayesian neural networks

Jonathan Berezowski

Master’s thesis in Statistics – STA-3900 – June 2021

"Happiness is a cookie that the brain bakes for itself."

Joscha Bach, AI researcher

For my mom and dad, who respectively have instilled in me a penchant for creativity and a great sense of curiosity; your inspiration is implicit in the pages that follow.

UIT NORGES ARKTISKE UNIVERSITET

Abstract

Faculty of Science and Technology

Department of Mathematics and Statistics

Master of Statistics

Trans-dimensional Inference over Bayesian Neural Networks

by Jonathan Berezowski

Trans-dimensional Bayesian inference for multi-layer perceptron architectures of varying size by reversible jump Markov chain Monte Carlo is developed and examined for its theoretical and practical merits and considerations. The algorithm features the No-U-Turn Sampler and Hamiltonian Monte Carlo for within-dimension moves, and makes use of a delayed-rejection sampler while exploring a variety of across-dimension moves that propose neural network models with varying numbers of hidden layers and hidden nodes. The advantages and considerations of sampling from a joint posterior distribution over model architecture and parameters are examined, and posterior predictive distributions are developed for classification and regression tasks.

Acknowledgements

I would like to extend my sincerest gratitude to the members of my supervisory team, who have been with me for this entire sojourn through Bayesian deep learning. Thomas, my lead supervisor, has been a terrific mentor with regards to the preparation of this thesis, and has guided me through the acquisition of programming and development skills that were paramount in successfully implementing the inference algorithms used to investigate the Reversible Jump Bayesian Neural Network. Jonas provided a wealth of supportive feedback and guidance in the space of machine learning research; his sense of humour and welcoming demeanour were greatly appreciated. And to Fred, who seeded the investigation of Bayesian deep learning that led to this work; his consistent encouragement through the ups and downs, and his endorsement of me to expand my involvement with the group through internships and presentations, have made for a very positive experience through my master's program.

I offer a very special thanks to Dr. Peter Green for taking the time to discuss my work with regards to his Reversible Jump Markov Chain Monte Carlo algorithm. His suggestions and reassurances were considerably helpful and motivating.

The members of the Machine Learning Group, including my fellow master's students and the academic staff, have been a great source of meaningful inspiration, invaluable advice, and splendid camaraderie through my time as a master's student at UiT.

To Harald, who has been a friend and peer through classes and our involvement with the group; our shared enthusiasm for deep learning and the Bayesian approach has made for many enjoyable coffee breaks. He and his family made me feel very welcome in Tromsø at a time when border restrictions made travel home impossible.

To Alessandro, the mathematics to my statistics, for the deep and meaningful conversations in the realm of the quantitative disciplines and well beyond.

To Katya, for all of the positive memories and support through what has in many ways been a difficult final semester.

And lastly, to Brian, who provided the rubber duck which bore the burden of entertaining a majority of the frustrating programming challenges that I encountered along the way - of which there were many.

Contents

List of Figures

List of Tables

List of Abbreviations

ANN	Artificial Neural Network
BLR	Bayesian Logistic Regression
BR	Bayesian (Linear) Regression
BNN	Bayesian Neural Network
DL	Deep Learning
ELBO	Evidence Lower Bound
HMC	Hamiltonian Monte Carlo
INLA	Integrated Nested Laplace Approximations
MAP	Maximum a-Posteriori
MCI	Monte Carlo Integration
MCMC	Markov Chain Monte Carlo
MH	Metropolis Hastings
MHG	Metropolis Hastings Green
ML	Machine Learning
MLE	Maximum Likelihood Estimate
MLP	Multi-Layer Perceptron
NUTS	No U-Turn Sampler
PC	Principal Component
PCA	Principal Component Analysis
PDF	Probability Density Function
PMF	Probability Mass Function
ReLU	Rectified Linear Unit
RJBNN	Reversible Jump Bayesian Neural Network
RJMCMC	Reversible Jump Markov Chain Monte Carlo
RJNUTS	Reversible Jump No U-Turn Sampler
RMSE	Root Mean Square Error
SVI	Stochastic Variational Inference

List of Symbols

X, Y, Z	A randomly distributed (stochastic) variable
x, y, z	The realization of a random variable
f, g, h	Deterministic functions
$\mathcal{M}$	Statistical model
$\boldsymbol{\theta}$	Parameter vector for a statistical model [1]
$\boldsymbol{\Theta}$	The support of all possible parameterizations for $\boldsymbol{\theta}$
k, ℓ	Model architecture indicators
$\boldsymbol{K}, \boldsymbol{L}$	The support of all possible realizations for k, ℓ
$\mathcal{K}, \mathcal{L}$	Model architecture indicator random variables
$p(\boldsymbol{x})$	Probability Density/Mass Function
$\boldsymbol{D}$	Representation of the data for an inference task
$\mathcal{L}(\boldsymbol{\theta}\|\boldsymbol{D})$	The likelihood of $\boldsymbol{\theta}$ given $\boldsymbol{D}$
$E[X]$	The expectation of X
$N(), Ber(), Cat(), MVN()$	Known probability distributions
$\boldsymbol{\mu}, \boldsymbol{\sigma}^2, \alpha, \beta$	Parameters for known probability distributions
σ	Sigmoid activation function
$\pi(\boldsymbol{x})$	The posterior distribution of $\boldsymbol{x}$

[1]Bolded letters and symbols denote vectors

Chapter 1

Introduction

Trans-dimensional inference may sound like a fantastical term. Consideration of concepts that transcend a given dimension evokes notions of physics-bending phenomena that would be more appropriate to a work of science-fiction than one of academic writing. With this suggestive potential towards the extravagant in mind, the opening chapter of this thesis is devoted to an overview of what is meant by trans-dimensional inference.

The dimensionality that we are concerned with is that of the parameter vector for a statistical model, and inference is the technique that will be used to gain insight into a better understanding of how that dimensionality, as well as other features of the model, should be specified. Inference analyses for which the parameter vector is not of a fixed dimension are therefore trans-dimensional [1] - a relatively straightforward designation.

This is not to say that statistical inference need not be considered extravagant. The appeal of inference may be subtle, but in the information age, data-driven understandings of systems and processes through automated inference and learning techniques are becoming increasingly popular [2]. Directly coinciding with the use of popular machine learning models, such as neural networks, is a push to better understand the inner-workings and limitations of such models - specifically, there is a demand to properly characterize the uncertainty associated with the use of machine learning models for their predictive capabilities [3]. This need becomes increasingly apparent as more applications of machine learning become integrated into everyday technology, with potentially dire implications for poorly-specified models [4].

Neural networks are becoming increasingly complex [5]. Uncertainty regarding the model specification is not limited to its predictive output, but also the model itself. The question of how to propose an optimal size of neural network for a problem is an open problem [6], and insight into how to make choices about an appropriate architecture should therefore be valuable.

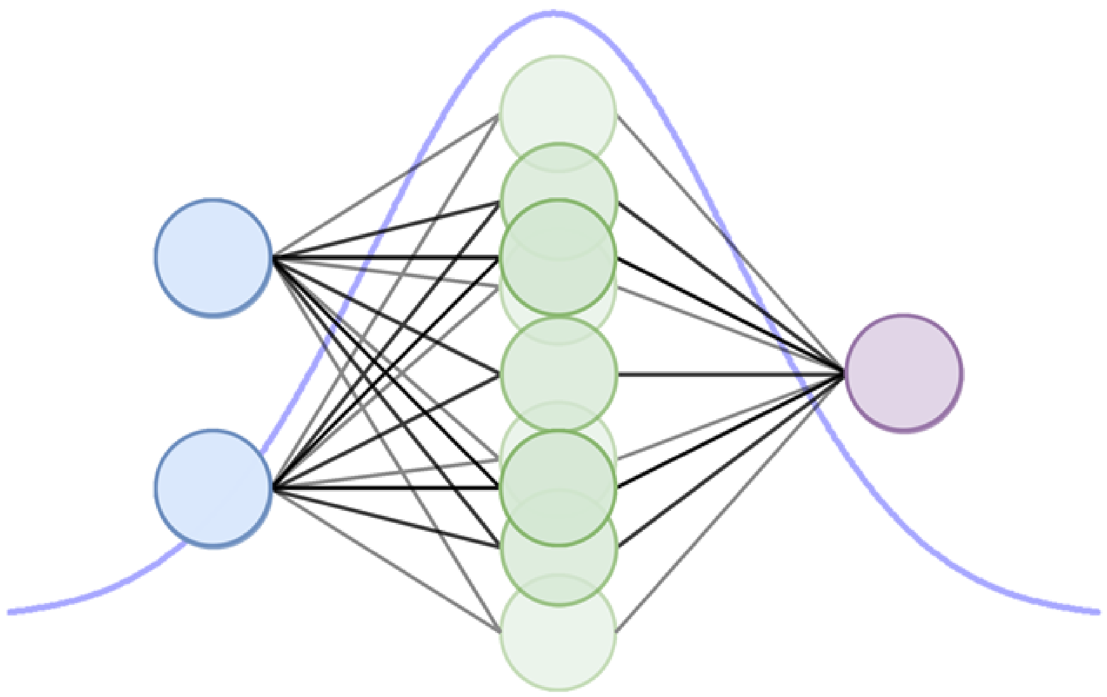


FIGURE 1.1: Trans-dimensional inference over a Neural Network of variable network width

1.1 Targeted Contributions

In this thesis, we aim to develop a method for using Bayesian inference via reversible jump Markov chain Monte Carlo simulation on neural network models of varying size to produce marginal posterior distributions corresponding to two aspects of neural network architecture specification: **network depth**; the number of hidden layers in the network, and **network width**; the number of hidden nodes in a single layer network (figure 1.1). Such an approach is an example of *trans-dimensional inference*, referring to the uncertainty regarding how many parameters our non-parametric neural network model should have, and therefore the dimension of the corresponding parameter spaces. We demonstrate these two contributions based on classification and regression experiments for which reasonable metrics (root mean square error, classification accuracy) are achieved on a held-out test set of samples, while also achieving a reasonable acceptance rate to across-dimension proposals via reversible jump Markov chain Monte Carlo.

We also present a novel approach for delayed-rejection sampling via the combination of reversible jump Markov chain Monte Carlo and automated Hamiltonian Monte Carlo through the No-U-Turn Sampler to improve the acceptance rate of across-dimension proposals.

1.2 Thesis Overview

To fully understand the application of trans-dimensional inference to Bayesian neural networks, we begin with a preliminary discussion of statistical model selection, machine learning, and Bayesian statistics in chapter 2.

Key details of statistical inference in the Bayesian paradigm are presented in chapter 3.

In chapter 4, Markov chain Monte Carlo (MCMC) as the inference approach of interest is reviewed in detail, including an overview of all sampling algorithms applied in this thesis as components of a custom trans-dimensional inference engine.

Chapter 5 presents the novel trans-dimensional Bayesian neural network model, and the implementation of the composite MCMC sampling algorithm.

Chapter 6 outlines experiments conducted to test the inference procedure for the trans-dimensional Bayesian neural network.

A discussion of the results, future research opportunities, and concluding remarks are presented in chapter 7.

Chapter 2

Statistical Modelling and Machine Learning

We begin with a preliminary discussion of statistical modelling at a relatively basic level. The constituents of a statistical model and the concepts surrounding model selection will be important in the discussion as we build towards trans-dimensional inference.

Machine learning is also introduced using this language of model selection, so that the featured neural network models may be presented with the model selection background in mind. The chapter closes with the basics of the Bayesian framework for statistical modelling, serving as the final preliminary ingredient to subsequent chapters, which explore Bayesian neural networks and trans-dimensional inference in detail.

2.1 Statistical Models

Statistical modelling is the design of experiments to explain a set of observed data according to a probability distribution [7]. The modelling practitioner must specify a reasonable distribution based on the characterization of the data, and appropriately tune the associated parameters to accurately reflect the observed data.

A model is a simplified representation of a real-world system or process. This is referred to as the data-generating process [8], which is abstracted down to a mathematically-defined statement about the inputs and outputs of the model, which itself is specified by its structure (architecture) and its parameters.

The model structure refers to its functional composition. In the declaration of a model's structure, the practitioner defines either implicitly or explicitly how many parameters $\boldsymbol{\theta} = [\theta_1, \theta_2, \ldots, \theta_d]$ the model will have, the function f over these parameters, and the independent variables $\boldsymbol{x}$ which the dependant variables $\boldsymbol{y}$ are conditioned on. A model definition will therefore be of the form:

$$\mathcal{M} = \{\boldsymbol{x}, \boldsymbol{y}, f, \boldsymbol{\theta}\} \tag{2.1}$$

$$\boldsymbol{y} = f(\boldsymbol{x}, \boldsymbol{\theta}) \tag{2.2}$$

where $\boldsymbol{y}$ represents the vector of dependent (response) variables, $\boldsymbol{x}$ represents the vector of independent (feature) variables, $\boldsymbol{\theta}$ refers to the set of model parameters, and f refers to the function uniting these model features. Model parameters may by realized from a varying range of values depending on the nature of the model structure. They may be continuous or discrete, and may have such bounds as being strictly positive, negative, finite, or correspond to a more specific interval on the real line.

2.1.1 Model Selection

In statistical model selection, a set of k candidate models $\{\mathcal{M}_1, \mathcal{M}_2, ..., \mathcal{M}_k\}$ are declared and compared on the specific modelling task. As few as two models may be declared as candidates for the selection; in more advanced cases the set may contain an infinite number of models. The latter is not at all uncommon; one may consider for example a continuous support $\boldsymbol{\Theta}$ for the parameter vector, for which a d-dimensional realization $\boldsymbol{\theta} \in \boldsymbol{\Theta}$ will correspond to an element of an uncountable set.

In simple cases, candidate models may be distinguished from one another solely by the parameter vector $\boldsymbol{\theta}$ with a fixed dimension. The models may also differ in terms of functional form f or the number of parameters $d = |\boldsymbol{\theta}|$, in which case models are usually distinguished by a model indicator k. The indicator may be defined explicitly for each candidate model, or correspond to a functional relationship ϕ such that $d = \phi(k)$. This caveat is especially relevant to *non-parametric*[1] models. Table 2.1 displays examples of model definitions for select parametric and non-parametric models.

Model	$\boldsymbol{\theta}$	f	**Type**
Simple Linear Regression	m, b	$y = mx + b$	Parametric
Normal Distribution	μ, σ^2	$y = \frac{1}{\sqrt{2\pi\sigma}} \exp\left\{\frac{1}{2}(\frac{(x-\mu)}{\sigma})^2\right\}$	Parametric
Gaussian Mixture Model	$k, \boldsymbol{\phi}, \boldsymbol{\mu}, \boldsymbol{\sigma}^2$	$y = \sum_{i=1}^{k} \phi_i \mathcal{N}(\boldsymbol{x}\|\mu_i, \sigma_i^2)$	Non-parametric

TABLE 2.1: Examples of select statistical models

Functional structure f dictates the relationship between parameters $\boldsymbol{\theta} = \{\dots\}$, feature variables $\boldsymbol{x}$, and response variables $\boldsymbol{y}$.

[1]A model classified as non-parametric is somewhat a misnomer. Non-parametric model are distinguished from parametric models for having parameters that do not necessarily fulfill unique roles in modelling of the data.

Specification of the set of candidate models must be paired with a suitable criteria for selecting which model should be employed for the desired analysis. Restricted to a handful of discrete models, it may be possible to directly compare each model's score based on some suitable metric for its performance. This is the fundamental concept behind the classical model selection approach known as *hypothesis testing* [9], for which a test statistic $s = f(\boldsymbol{x})$ is assumed to be drawn from some distribution $s \sim p_s(\boldsymbol{\theta})$. The probability associated with the test statistic is known as the *p-value*, and provides information about the relative likelihood of observing the data given competing hypotheses (i.e. parameterizations of the model). Rejection of a model corresponding to the null hypothesis is based on whether or not the p-value is above or below a threshold known as the *significance level*, which must be specified by the researcher prior to performing any analysis on the data.

In the case of a continuous support for model parameters, analytical approaches such as *maximum-likelihood estimation* (MLE) [10] may be appropriate. MLE casts an optimization problem over $\boldsymbol{\theta}$ given the data $\boldsymbol{D} = \{\boldsymbol{x}, \boldsymbol{y}\}$ corresponding to a *likelihood* function $\mathcal{L}(\boldsymbol{\theta}|\boldsymbol{D})$. A likelihood function is a representation of the observed data based on a candidate model, and therefore provides a relative measure of the goodness-of-fit of a given parameter vector. The MLE estimate is therefore the optimal parameterization from the support of possible parameterizations Θ as in equation 2.3. Likelihood functions are discussed further in section 3.2.3.

$$\hat{\boldsymbol{\theta}}_{\text{MLE}} = \underset{\boldsymbol{\theta} \in \boldsymbol{\Theta}}{\arg\max}\, \mathcal{L}(\boldsymbol{\theta}|\boldsymbol{D}) \tag{2.3}$$

When such methods are not tractable, we may require more sophisticated computational approaches that rely on algorithms to iteratively assess realizations of the distribution of models for their competency with regards to the analysis task at hand. The notion of model competency here may correspond to, for example, optimization of the likelihood as above, or a similar metric. Such approaches to modelling are known as *statistical learning* [11].

The target analysis of model selection via statistical learning will generally correspond to one of two motivations - *prediction*, or *inference* [12]. Model selection for prediction seeks to explain the target variable $\boldsymbol{y}$ as a function of the observed variables $\boldsymbol{x}$ so that the observation of a new data point $\boldsymbol{x}_i$ can be treated to a reasonable predicted outcome $\boldsymbol{y}_i$. With regards to statistical inference, model selection aims to provide insight into the data-generating process through analysis of likely values of $\boldsymbol{\theta}$, and corresponding measures of uncertainty around such parameters.

Statistical learning, or rather *machine learning*[2], is now introduced as one approach to statistical modelling.

2.2 Machine Learning

Machine learning (ML) [14] is the application of algorithms to automatically improve statistical models using data. Considered to be a subfield of artificial intelligence [15], the learning aspect of ML arises analogously from a semi-autonomous agent (the model) iteratively improving its representation of a data generating process based on continuous assessment of how well the model explains the observed data, and a feedback signal which dictates how the model must be improved next.

ML methods are computationally intensive, and are often an effective approach for handling massive datasets. Typical ML models are designed to be flexible on the support of their parameters so that reasonable realizations can be learned from the available data, often with minimal restriction regarding the nature of the parameters.

Model selection via ML most commonly concerns statistical prediction. When a dataset is comprised of a series of explanatory variables $\boldsymbol{x}$ and associated response variables $\boldsymbol{y}$, the dataset is said to be labelled with y_i being the label corresponding to observation vector $\boldsymbol{x}_i$. Updating parameters of a model to best map the functional relation between $\boldsymbol{x}$ and $\boldsymbol{y}$ is known as *supervised learning* [16].

In contrast to modelling tasks motivated by prediction, statistical inference doesn't immediately lend itself to the highly flexible models generally dealt with in ML applications. Many ML models are non-parametric with arbitrary parameters that don't necessarily correspond to real world factors, phenomena, or implications. Such models are often colloquially referred to as *black boxes* [17], since we have knowledge of the inputs and outputs of the system, but little insight regarding the internal workings.

This is, of course, not entirely the case. All relevant ML models are derived based on some combination of their ability to fit a given data analysis task, their practical convenience with regards to computational constraints, and the availability of known relevant mathematical results. Careful examination of these models based on their composition and the training programs they are treated with can shed light on features and relationships within the data, behaviour of the stochastic and deterministic aspects of the applied training algorithms, and opportunities for the development of new theoretical results in ML, statistics and information theory.

[2]The literature is somewhat unclear on the distinctions between the terms statistical learning and machine learning [13]. For the sake of this thesis, both approaches are broadly taken to be equivalent, and ML is designated as the term to represent the relevant learning concepts employed.

In this thesis, we will aim to gain some insight into model specification for one particular class of ML models: neural networks.

2.2.1 Artificial Neural Networks

An *artificial neural network* (ANN) or *multi-layer perceptron* (MLP) is a flexible model structure popular in modern machine learning applications for image classification [18], speech translation [19], image segmentation [20], and numerous other industry applications. A known result of ANNs is the ability to universally approximate any arbitrary function with continuous inputs and outputs given a sufficiently large network [21]. This compelling opportunity simultaneously motivates the popularity of neural networks in practice, and our investigation into trans-dimensional inference.

Perceptron

The basic building block of an ANN is a *perceptron* [22], which pairs an activation function $g(z)$ with a linear transformation $z = h(\boldsymbol{X})$. The function h is a linear transformation comprised of a weight parameter $\boldsymbol{w}$ together with a bias b, analogous to the slope and intercept parameters in a linear regression. Taken all together, equation 2.4 corresponds to an estimate for the response vector $\hat{\boldsymbol{y}} \approx \boldsymbol{y}$.

$$\hat{\boldsymbol{y}} = g(\boldsymbol{w}\boldsymbol{x} + b) \tag{2.4}$$

For a *regression* or *interpolation* problem, the objective is to directly estimate $\boldsymbol{y}$, and thus g is often taken to be the identity function. The model may instead correspond to a *classification* task, for which each data point is to be labelled according to some set $j \in \{1, 2, \ldots, c\}$. In this case, g is commonly taken to be a *sigmoid* function (equation 2.5) in the case of binary labels ($c = 2$), or the *softmax* function (equation 2.6) when dealing with multiple labels ($c > 2$).

$$g(\boldsymbol{z}) = \frac{1}{1 + e^{-\boldsymbol{z}}} \tag{2.5}$$

$$g(z_j) = \frac{e^{z_j}}{\sum_{i=1}^{c} e^{z_i}} \tag{2.6}$$

Multilayer Perceptrons

An ANN extends the perceptron model to a hierarchical function. We now consider multiple parallel perceptrons (henceforth referred to as *neurons*) which simultaneously compute activations on linear transformations of the input data, each contributing a realization to a vector of outputs. The weight parameters are now represented by a vector $\boldsymbol{w}$. We consider this to be a layer of an ANN, which we may stack arbitrarily many of such that the output of each layer is treated as the input to the next layer.

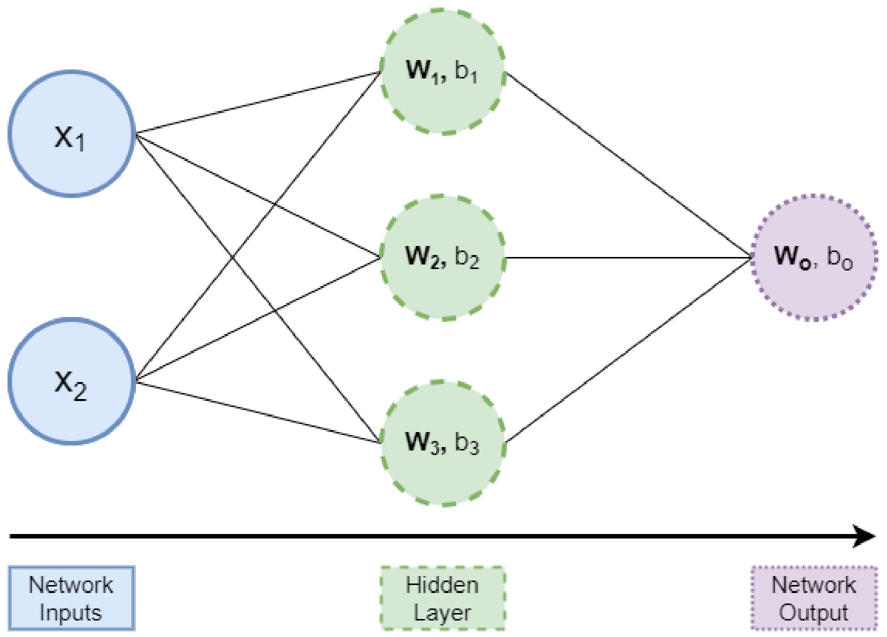


FIGURE 2.1: A simple ANN with one hidden layer.

Denoting the i-th layer with its corresponding linear transformation z_i and activation function g_i:

$$z_i = \boldsymbol{w}_i^T \boldsymbol{y}_{i-1} + b_i \tag{2.7}$$

$$\boldsymbol{y}_i = g_i(\boldsymbol{z}_i) \tag{2.8}$$

The input layer may be thought of as the observational data, such that $\boldsymbol{y}_0 = \boldsymbol{x}$, for which the number of nodes will correspond to the dimension d of the feature data. The final output of the model will be the final activation function corresponding to the modelling task. For a network with ℓ layers, the final output is therefore:

$$\hat{\boldsymbol{y}} = g_\ell(\boldsymbol{z}_\ell) \tag{2.9}$$

The layers in between the inputs and outputs are referred to as hidden layers. An example of a simple ANN architecture is presented in figure 2.1. A network with more than one hidden layer is a deep neural network, and may be considered the flagship model of *deep learning* [23].

The activation functions used for the hidden layers may differ from those of the output layer. The sigmoid activation (equation 2.5) may be used, or the similarly featured *tanh* activation (equation 2.10), which scales output values onto a range of (-1,1). Increasingly popular in modern neural networks is use of the *rectified linear unit* (ReLU) [24] (equation 2.11), which projects negative values to 0 and the identity function to positive values. ReLU's properties allow for quick gradient calculations and has been shown to perform optimally in feedforward networks.

$$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{2.10}$$

$$g(z_j) = \begin{cases} 0, & z_j \leq 0 \\ z_j & \text{otherwise} \end{cases} \tag{2.11}$$

As an important note of clarification, a neural network corresponds to the functional structure f in the definition of a model $\mathcal{M}$ in equation 2.1. This will be an important baseline as we introduce models that extend the concept of a neural network beyond fixed realizations for the weights and biases (Bayesian Neural Networks, section 3.5.2), and later consider the architecture to be variable as well (section 5.1). For clarity's sake, in this work the term ANN will be used to strictly refer to neural network architectures employed in classical machine learning models as introduced in this section, for which the result of a training program is a neural network model with both the architecture and the paramaterization considered to be fixed.

2.2.2 Principal Components Analysis

Having introduced ANNs as an example of supervised learning, we briefly present a method of unsupervised learning for contrast. This method is also used in the experiments section to augment a dataset to be more tenable for the computationally intensive inference algorithm that will be the focus of this thesis.

Principal Component Analysis (PCA) [25] is a *dimensionality-reduction* technique that produces a representation of a d-dimensional dataset $\boldsymbol{x}$ through a change-of-basis to the d original feature vectors to maximize the variance of the feature space. These optimized features are known as the principal components (PCs) of the dataset, and are established such that projection of the data samples onto the PCs maximizes the "spread" of the data to make the difference between samples more apparent.

Formally, given the covariance matrix $\boldsymbol{\Sigma}$ for $\boldsymbol{x}$, we define the projection $\boldsymbol{z}_1$ onto the first PC $\boldsymbol{c}_1$:

$$\boldsymbol{z}_1 = \boldsymbol{c}_1^T \boldsymbol{x} \tag{2.12}$$

and seek to maximize the variance $\boldsymbol{c}_1^T \boldsymbol{\Sigma} \boldsymbol{c}_1$ subject to the constraint that $||\boldsymbol{c}_1|| = 1$ for a unique solution. The Lagrange optimization problem is then:

$$\max_{\boldsymbol{c}_1} \boldsymbol{c}_1^T \boldsymbol{\Sigma} \boldsymbol{c}_1 - \alpha(\boldsymbol{c}_1^T \boldsymbol{c}_1 - 1) \tag{2.13}$$

We take the derivative of 2.13 with respect to $\boldsymbol{c}_1$ and set it equal to 0 to arrive at:

$$\boldsymbol{\Sigma c}_1 = \alpha \boldsymbol{c}_1 \tag{2.14}$$

which holds if $\boldsymbol{c}_1$ is an eigenvector of $\boldsymbol{\Sigma}$ with corresponding eigenvalue α, and for maximization we therefore select the eigenvector corresponding to the largest eigenvalue λ_1. The argument follows for selection of subsequent principal components $\boldsymbol{c}_2, \ldots, \boldsymbol{c}_d$ as the eigenvectors $\boldsymbol{e}_2, \ldots, \boldsymbol{e}_d$ corresponding to the eigenvalues ordered in decreasing size $\lambda_2, \ldots, \lambda_d$.

The target representation will concern some $d' <= d$ PCs and will be optimized based solely on the variance of the data samples across these features. This is what makes the method unsupervised, as no manner of output data is considered in determination of the optimization criterion. Selection of a number of d' of the PCs may correspond to a targeted proportion of explained variance as defined by the user. The total proportion of variance explained by the first d' PCs can be calculated based on the sum of the d' eigenvalues over the sum of all d eigenvalues as in equation 2.15.

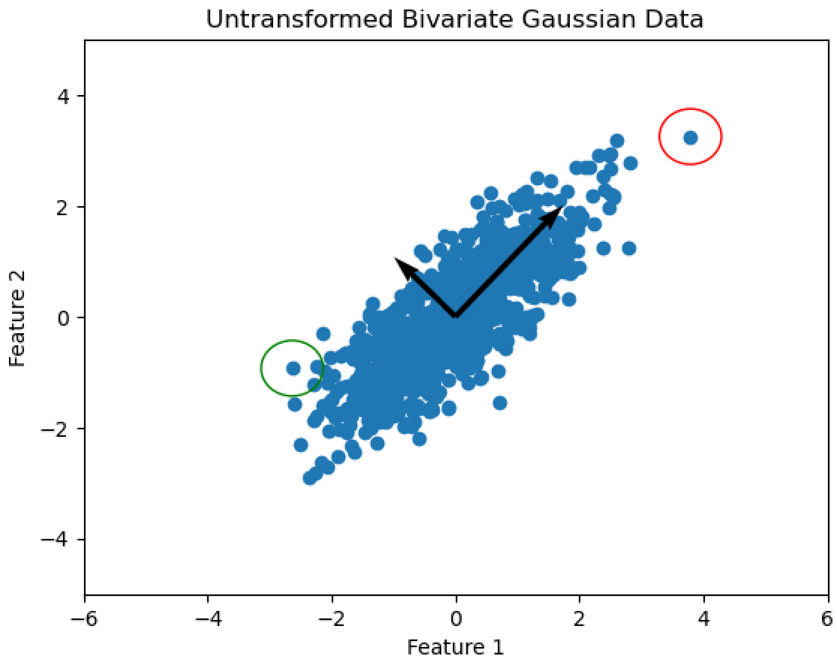
$$\text{proportion of variance} = \frac{\sum_{i=1}^{d'} \lambda_i}{\sum_{j=1}^{d} \lambda_j} \tag{2.15}$$

2.3 Bayesian Statistics

Statistical modelling relies heavily on key concepts from probability theory. Probability arises from measure theory as a rigorous examination of sets of possible outcomes for processes. A realization of one of these outcomes or a specific outcome from a given subset of the possibilities is known as an event. The measure of how likely[3] any given event is to occur is referred to as the probability of the event, but a further examination of precisely what is meant by "likely" now diverges according to which of two paradigms of statistics one wishes to consider.

1. In the *frequentist* (often labelled the "classical" approach to statistics) paradigm, probability is a measure of the occurence of outcomes relative to all possible outcomes that could occur over repeating incidents of the given process.
2. In the *Bayesian* paradigm, probability is instead considered to be a "degree of belief" in a definite statement about an outcome corresponding to a given event.

[3]Respect is given to the fact that "likely" corresponds to a more precise definition in statistics, but is used here somewhat colloquially.



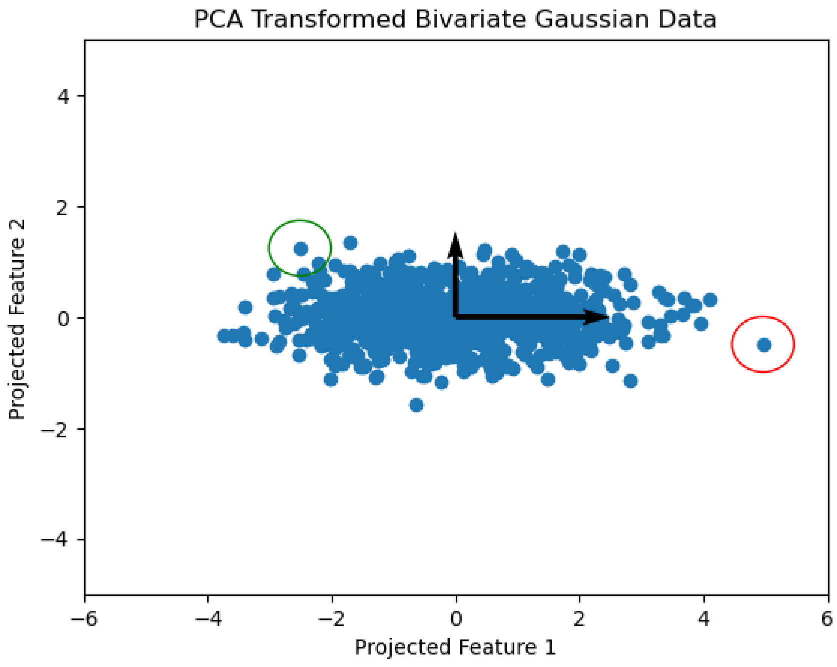


FIGURE 2.2: PCA transform applied to bivariate Gaussian data

Reverand Thomas Bayes is credited with the titular result regarding conditional probabilities known as "Bayes' Rule" [26] (equation 2.16).

$$p(a|b) = \frac{p(b|a)p(a)}{p(b)} \tag{2.16}$$

In the above statement, a and b refer to fixed probabilities of some events happening. Bayesian inference takes the heart of this approach and generalizes it to distributions of model parameters as random variables given the observation of data.

$$p(\boldsymbol{\theta}|\boldsymbol{D}) = \frac{p(\boldsymbol{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\boldsymbol{D})} \tag{2.17}$$

In the left side of equation 2.17, the distribution of the parameters $\boldsymbol{\theta}$ based on the data $\boldsymbol{D}$ is known as the *posterior distribution*. Perhaps more succinctly, this represents a distribution of candidate models, for which the probability of a given model is weighted according to the prior belief regarding $\boldsymbol{\theta}$ and the observed data.

The posterior distribution is the main product of Bayesian inference, which is discussed in chapter 3.

2.3.1 Bayesian Model Selection

When comparing two Bayesian models, the *Bayes factor* [27] can be computed to determine the relative efficacy of one model over the other. Given the conditional distributions for two models $\mathcal{M}_1$ and $\mathcal{M}_2$ based on the observed data as well as the prior probabilities over the two alternative models, the Bayes factor Ψ is computed as the ratio of the two (equation 2.18).

$$\Psi = \frac{p(\mathcal{M}_1|\boldsymbol{D})}{p(\mathcal{M}_2|\boldsymbol{D})}\frac{p(\mathcal{M}_2)}{p(\mathcal{M}_1)} \tag{2.18}$$

The Bayes factor considers the likelihood of observing $\boldsymbol{D}$ given all possible parameterizations of $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$. The second ratio in equation 2.18 refers to the prior probabilities for the models, which may as a default be taken to be equal such that $p(\mathcal{M}_1) = p(\mathcal{M}_2) = 0.5$. In this case, equation 2.18 reduces to a ratio of the two posterior distributions. Ψ is therefore an indicator of which of the two models better explains the data, and to what degree it outperforms the other. A value of $\Psi > 1$ suggests that $\mathcal{M}_1$ is preferable.

The Bayes factor may be used to compare two models, but model selection by Bayesian inference is not limited to cases featuring binary candidates [28]. Given that no prior preference is assigned to any particular model (i.e. $p(\mathcal{M}_i) = \frac{1}{m}\,\forall i \in \{1 : m\}$), the posterior score as a result of some inference procedure may be used to "rank" each candidate model. This approach is presented in chapter 3.

Chapter 3

Bayesian Inference

We have chosen to pursue a Bayesian analysis of neural network models, and will therefore be dealing with degrees of belief in our discussion of model selection. The implication of this injection of belief must be formalized to connect philosophical conjecture to a practical implication - namely, how our prior understanding of the problem can be balanced with the observed data, and how to interpret our updated beliefs after inference has been performed.

This chapter presents information on the motivation and procedure of Bayesian inference, an introduction to the relevant methods, and culminates in a detailed description of Bayesian Neural Networks. We begin with a discussion of uncertainty, specifically referencing how it will pertain to the Bayesian modelling approach and our goal to gain a better understanding of optimal BNN architecture selection.

3.1 The Role of Uncertainty in Statistical Modelling

In defining a statistical model, we abstract a complex natural process down to a select few key components. A well-defined model includes a sufficient subset of the true components of the data-generating process such that a reasonable degree of inferential or predictive insight is gained to make meaningful statements about the examined processes. Except for trivial analyses or perfectly isolated systems, some information is inherently lost in the abstraction. With regards to Bayesian machine learning, the goal is often to achieve a distribution of parameters of a model for the sake of predictive capabilities, but we do not typically strive to exactly recreate the complexity of the examined natural phenomena at hand. [29].

Given that statements about model parameters and resultant predictions regarding estimation or classification of the response variables correspond to degrees of belief, uncertainty is therefore inherent within a statistical model, categorized as arising from two distinct sources [3]. First, one acknowledges that the amount of available information for a data set is limited by the existing prior knowledge and the size of the data set. This consequent source of uncertainty is defined to be *epistemic* (also known as *model uncertainty*), and may be reduced either by defining additional prior knowledge, or obtaining additional data. In contrast, *aleatoric* uncertainty is that which arises from the inherent randomness of the data generating process. Any finite data set can only ever represent a snapshot of the ground truth, placing a constraint on the limit of model efficacy. Aleatoric uncertainty may always be present as a limitation of that which cannot be known about the data-generating process.

Both sources of uncertainty are present in non-parametric model selection. The aleatoric uncertainty arises intrinsically due to the stochastic nature of the data-generating process responsible for the observed data. Some epistemic uncertainty can be attributed to a lack of knowledge about this data-generating process, but also due to the design of the non-parametric model, including its functional representation, architecture, and the factors affecting its parameterization (training procedure, learning metrics i.e. cost function). It is therefore desirable when dealing with non-parametric models, such as neural networks, to be able to characterize this source of uncertainty for a better understanding of the limitations of the predictive capabilities of the model.

3.1.1 Model Architecture Uncertainty

Chapter 2 presented a minimum of prerequisite information to define trans-dimensional inference, for which insight into model architecture specification may be sought.

With the language of uncertainty available, we will demonstrate through experiments that Bayesian inference can provide not only an optimal point estimate of associated parameters, but distributions of model architectures. We emphasize the following claim:

Proposing models that do not specify a fixed architecture corresponds to an assumption that we are not certain any one non-parametric model architecture is necessarily appropriate or optimal for analysis of the data.

We therefore strive to represent a source of epistemic uncertainty which is not addressed by default in classical ML approaches to ANN learning - specifically, the size of the neural network architecture.

Many of the details presented in the following sections are expressed in terms of model parameters for standard (fixed-dimension) inference, but extend naturally to the trans-dimensional case for inferring architecture. This extension is addressed through the use of model indicators to represent architecture selection as explored in section 5.1.

3.2 Components of Bayesian Inference

Bayes' approach is mathematically intense, and the methods are computationally expensive [30]. Successful generation of the target approximations requires that our model specification and algorithm design correspond to the available prior information that we have regarding appropriate model selection for representation of the data.

To understand how the Bayesian approach allows for this principled inclusion of prior information and to properly generate uncertainty measures around the model parameters, a sound understanding of the components of Bayesian inference is required.

3.2.1 The Posterior Distribution

The Bayesian paradigm for statistical inference proposes that the parameters of a statistical model are random elements, and the observed data are fixed. Given a model $\mathcal{M} : \boldsymbol{y} = f(\boldsymbol{\theta}, \boldsymbol{x})$, the vector of model parameters $\boldsymbol{\theta}$ is assumed to be a random variable arising from a distribution dependent on the observed data $\boldsymbol{D} = \{\boldsymbol{x}, \boldsymbol{y}\}$:

$$\boldsymbol{\theta} \sim p(\boldsymbol{\theta}|\boldsymbol{D}), \boldsymbol{\theta} \in \boldsymbol{\Theta} \tag{3.1}$$

where $\boldsymbol{\Theta}$ is the set of possible realizations of $\boldsymbol{\theta}$. Such a distribution is known as the posterior distribution, and is the mathematical entity of interest in Bayesian inference. It presents all of the features and information associated with a probability distribution.

The posterior distribution is obtained through Bayes' rule as it is applied to distributions of random variables. Given marginal distributions over the model parameters $p(\boldsymbol{\theta})$ and the data $p(\boldsymbol{D})$, respectively referred to as the *prior* distribution over the parameters and the *marginal evidence* of the data, as well as the conditional distribution of the data given the model parameters $p(\boldsymbol{D}|\boldsymbol{\theta})$, the posterior distribution is computed as

$$p(\boldsymbol{\theta}|\boldsymbol{D}) = \frac{p(\boldsymbol{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\boldsymbol{D})} \tag{3.2}$$

A corresponding *maximum a-posteriori* (MAP) estimate is a point estimate $\hat{\boldsymbol{y}}$ for $\boldsymbol{y}$ that maximizes the posterior score of the full Bayesian inference as in equation 3.3. The MAP estimate is considered analogous to the MLE technique for classical learning procedures, extending the metric to include how well a model explains the data while constrained by prior information.

$$\hat{\boldsymbol{\theta}}_{\text{MAP}} = \arg\max_{\boldsymbol{\theta} \in \boldsymbol{\Theta}} p(\boldsymbol{\theta}|\boldsymbol{D}) \tag{3.3}$$

Except in simple cases with convenient analytical properties, the posterior distribution is difficult to obtain exactly. The suite of inference techniques used to sample from the posterior or an approximation thereof are introduced in section 3.4.

3.2.2 The Prior Distribution

Whatever is known or suspected about the nature of $\boldsymbol{\theta}$ - before data has been observed - is reflected in the prior distribution, $p(\boldsymbol{\theta})$. It is simply a joint probability distribution over the model parameters. The simplest such case might be a multivariate distribution with a known functional form, such as a multivariate Gaussian, but it can also represent the product of independent probability distributions over different types of model parameters. The parameters need not be independent; the distribution can be expressed analytically or approximately to represent a conditional structure between different parameter types. This is common in the case of hierarchical models [31] (section 3.3).

A prior could also be defined as a posterior distribution from a previous analysis when new data becomes available. The philosophical appeal of this technique is the natural iterative ability of Bayesian inference - postulate the initial prior as a sort of null hypothesis, observe data, obtain a posterior, observe more data, obtain a new posterior, repeat.

The nature and amount of information available in specification of a prior is dependent on the analysis at hand. It might not be known *a priori* how certain model parameters are expected to behave in a complex model. This does not necessarily weaken the proposal of a Bayesian approach, and in fact may be a benefit - a prior with minimal imposition on select model parameters is still a principled way to define an assumption such as "little as assumed about $\boldsymbol{\theta}$" [32]. This leads to a distinction between informed and vague priors, which each present a trade-off between benefits and costs.

Informed Priors

The Bayesian approach offers a principled way for domain experts to inject their established expertise on a problem into the inference task. A distribution may be specified that places a narrow band or bands of relatively high probability density across certain parameter values, usually via modification of a prior scale parameter. The distribution may also be selected based on its established characteristics, including its range of possible values (namely whether values may go to $\pm\infty$ or be restricted on one or both ends of the range interval), the overall shape of the curve, and often deliberate selection of a value for the location parameter. A set of any or all of these insights characterize an informed prior distribution for the model parameters.

Vague Priors

Whenever the interpretability of model parameters is difficult, it is consequently challenging to define prior information about the nature of the distributions from which those parameters arise. In these situations, a vague prior may be appropriate, such that minimal restriction or relative weighting is placed on certain parameter values. This might correspond to a zero (or otherwise) centered Gaussian with particularly wide variances, such that all real-numbered values are candidates for the parameter, with only moderate preference given to those within a neighbourhood of the specified mean. In absence of a more informed choice of mean, centering a Gaussian on zero implies a weak preference for smaller parameter values.

A discussion of vague priors is specifically relevant to non-parametric models. Technically speaking, non-parametric models have parameters, or there would be no random variables to perform inference on using the Bayesian approach. Non-parametric models are those which do not have an a-priori model structure specified. A normal distribution is a parametric model characterized by a mean and variance parameter, whereas a MLP is an example of a non-parametric model which may have any number of hidden layers and varying numbers of nodes within those layers.

Vague priors are still required to be proper probability distributions, such that integration over the full support of the distribution is equal to one. This is in contrast to improper priors, for which the integrals diverge - such as a uniform distribution. Such priors are occasionally used in Bayesian inference, despite potentially introducing pathologies to the analysis [33].

It might seem as though vague priors offer little advantage over strictly likelihood based methods, but this is not found to be the case [34]. Even weakly-informative priors (as are common in BNN specification [35] - see section 3.5.3) assist in the practical implementation of inference methods. Even if the data is insufficiently informative to result in narrow posteriors distributions when using vague or improper priors, this result will be represented by appropriate uncertainty measures around parameters and predictions. This provides the researcher with information regarding whether the predictions are certain enough to proceed with the model as is, or that additional data/analysis is required.

Figures 3.1 and 3.2 illustrate a situation in which a vague prior leads to a more accurate estimation of the posterior distribution than an equivalent narrow prior, and develops some general intuition to the computation of a posterior distribution through Bayesian inference.

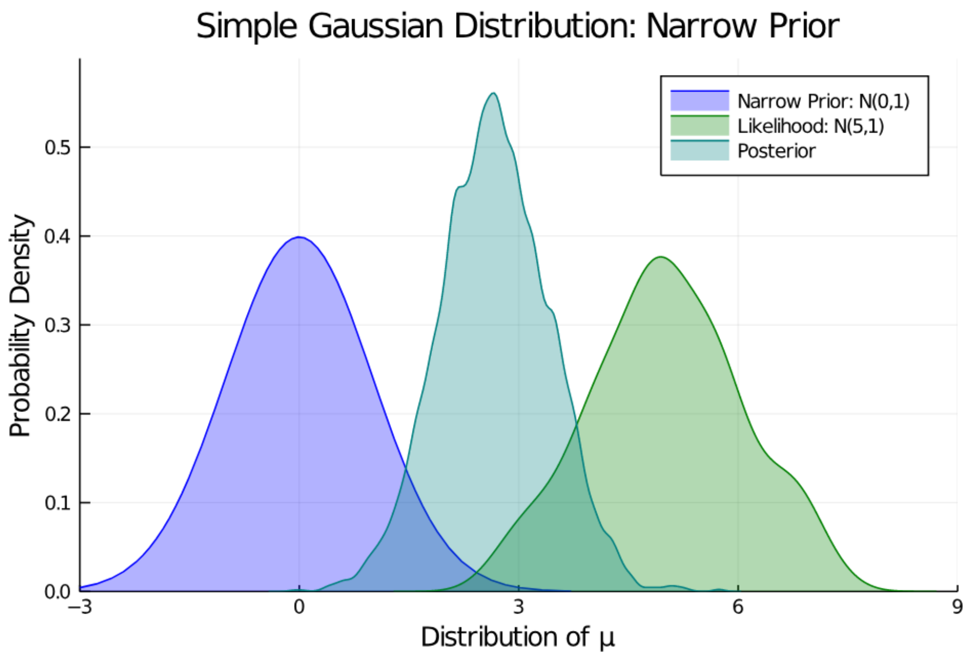


FIGURE 3.1: Inference over a simple Gaussian distribution: narrow prior

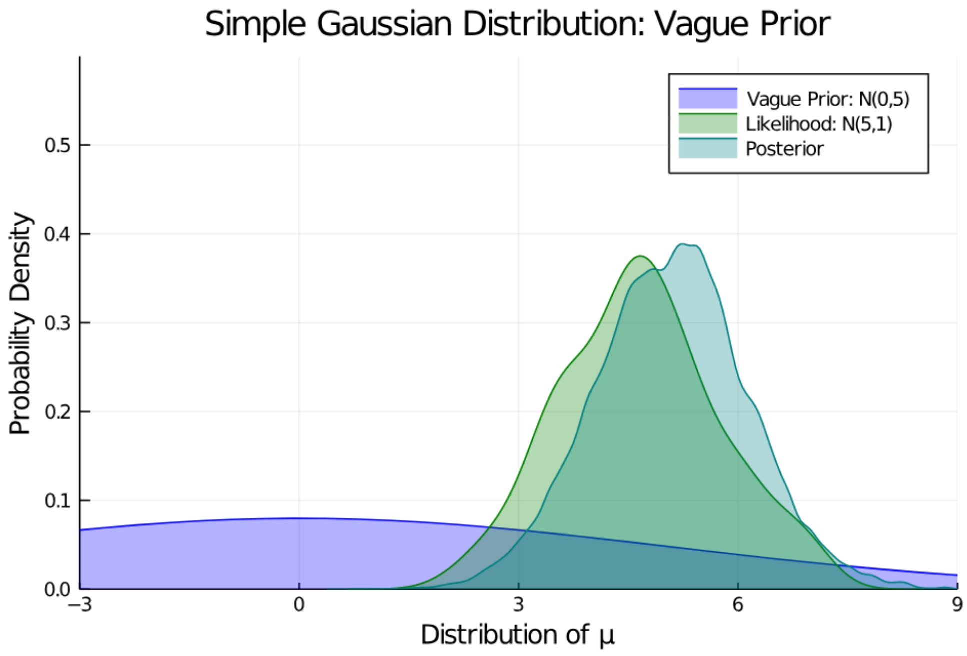


FIGURE 3.2: Inference over a simple Gaussian distribution: vague prior

100 samples are generated from a Gaussian distribution with mean 5 and a standard deviation of 1. Random-walk MCMC (see section 4.3) is used to simulate an approximate posterior distribution. The posterior is biased when a narrow prior is employed (figure 3.1). A vague prior allows the posterior to better mimic the likelihood (figure 3.2).

3.2.3 The Likelihood

The conditional distribution of the data given model parameters is more commonly referred to as the *likelihood* of the model parameters (equation 3.4).

$$p(\boldsymbol{D}|\boldsymbol{\theta}) \propto \mathcal{L}(\boldsymbol{\theta}|D) \tag{3.4}$$

The Bayesian formulation for a posterior distribution (equation 2.17) then becomes:

$$p(\boldsymbol{\theta}|\boldsymbol{D}) = \frac{\mathcal{L}(\boldsymbol{\theta}|\boldsymbol{D})p(\boldsymbol{\theta})}{p(\boldsymbol{D})} \tag{3.5}$$

The likelihood is a function, not a probability distribution. The output of a likelihood function $\mathcal{L}(\boldsymbol{\theta}|\boldsymbol{D})$ for some input parameter $\boldsymbol{\theta}$ given data $\boldsymbol{D}$ will not provide any information about the quality of the parameterization independently, as the relative scores are dependant on the specifics of the model structure. It can be used instead to compare two or more models based on their relative fit of the data. Such an approach is the basis for the method of MLE, wherein an optimal parameterization is determined as that model which maximizes the likelihood function and therefore best represents the data.

It is the pairing of a likelihood function with a properly specified prior distribution and normalizing constant that yields a posterior probability distribution. Whereas specification of the prior distribution is flexible, the likelihood is (partially) implicitly defined by the modelling task [36]. Fitted data as determined by a candidate model is measured for its goodness-of-fit based on the unnormalized likelihood, allowing for relative comparison of model quality as discussed in section 2.1. A tenuous analogy may be drawn between the likelihood of Bayesian inference to the loss function of classical machine learning as two components responsible for assessing the quality of the active model in their respective paradigms.

Typical likelihoods for classification and regression tasks are defined for the models of interest in this thesis in section 3.5.4.

3.2.4 The Model Evidence

The denominator of the Bayesian inference equation, $p(\boldsymbol{D})$[1], is the marginal distribution of the data, independent of model parameters. The use of $p(\boldsymbol{D})$ is shorthand for the expression representing the marginalization over all possible parameterizations of the model:

$$p(\boldsymbol{\theta}|\boldsymbol{D}) = \frac{\mathcal{L}(\boldsymbol{\theta}|\boldsymbol{D})p(\boldsymbol{\theta})}{\int_{\Theta} p(\boldsymbol{\theta},\boldsymbol{D})d\boldsymbol{\theta}} \tag{3.6}$$

[1]In the supervised setting, it is more formally written as the conditional distribution of the response based on the feature data, such that $P(\boldsymbol{D}) \equiv P(\boldsymbol{Y}|\boldsymbol{X})$.

Such a distribution is difficult to define and not of particular interest in terms of inferring the nature of the model, or extending a model to predictions on newly observed data [9]. It is also constant across model parameterizations, architectures, and functional specifications. It is therefore not necessary to explicitly compute the evidence when performing Bayesian inference for model selection. The statement can be made that the posterior distribution is proportional to the product of the likelihood and prior distributions (equation 3.7).

$$p(\boldsymbol{\theta}|\boldsymbol{D}) \propto \mathcal{L}(\boldsymbol{\theta}|\boldsymbol{D})p(\boldsymbol{\theta}) \tag{3.7}$$

A valid probability distribution by definition must integrate to 1 across its support, which holds for the exact posterior as in equation 3.8. Evaluation of the full posterior distribution following a Bayesian update should meet this criteria if $p(\boldsymbol{D})$ can be determined, but this is not a necessary validation where model selection is concerned. It is instead sufficient to evaluate a given posterior based on its unnormalized log-posterior score in comparison to other possible parameterizations in terms of returning point or interval estimates, or expectations.

$$\int_{\boldsymbol{\theta}\in\boldsymbol{\Theta}} p(\boldsymbol{\theta}|\boldsymbol{D})d\boldsymbol{\theta} = 1 \tag{3.8}$$

3.2.5 The Posterior Predictive Distribution

The posterior distribution itself serves as the end goal of Bayesian inference for model selection, but we need not stop there. A natural motivation for model selection in either a frequentist or Bayesian machine learning setting may then be to perform statistical prediction, for which the focus will then be on generating predictions for newly observed data. In the Bayesian case, these predictions will be the aggregated output of models drawn from the distribution of model parameters represented by the posterior, weighted by their posterior score. The posterior predictive distribution obtained through Bayesian inference treats each observed data point $\boldsymbol{x}$ with a distribution of possible response targets[2] $\boldsymbol{Y} = \boldsymbol{y}$:

$$p(\boldsymbol{y}|\boldsymbol{x}) = \int_{\boldsymbol{\Theta}} f(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta})p(\boldsymbol{\theta}|\boldsymbol{D})d\boldsymbol{\theta} \tag{3.9}$$

Evaluation of this *posterior predictive distribution* (or simply the *predictive distribution*) rewards the Bayesian practitioner with a wealth of information about each observed $\boldsymbol{x}$. An *expectation* can be approximated through a numerical integration technique over samples from the distribution. Uncertainty estimates can easily be achieved by evaluating credible intervals for each predicted response based on the *variance* as determined from the predictive distribution.

[2]In the case of supervised learning

All of this comes "for free" in contrast to the output of a classical machine learning model training run. Standard gradient descent methods produce MLE estimates for which MAP estimates may be seen as the Bayesian equivalent. Beyond that, methods exist to augment standard ML algorithms so that uncertainty estimates can be obtained, and ensemble methods can be employed to roughly approximate the approach of sampling from a posterior distribution [37]. These, however, must be employed separately in the classical setting.

3.2.6 Why Bayesian Inference: Prior Beliefs vs Evidence

A high-level interpretation of Bayesian inference might be as follows: a prior belief about the nature of a model is specified, and then data is observed to automatically update the model to better explain the data. The posterior distribution is thus characterized by a weighted mix of the original prior distribution over the model parameters as well as the likelihood of the observed data. The degree to which this mixture is weighted by these two components depends on the number of observations and the nature of the prior distribution. The addition of newly observed data will increase the relative impact of the likelihood, while the prior's impact can be increased by narrowing the specified joint distribution over parameters, or decreased by selecting a more vague distribution.

To both proponents and critics of the Bayesian paradigm, the prior distribution is often the foreward feature that distinguishes a Bayesian approach from a more orthodox method [38]. It is often associated with the introduction of subjectivity into a modelling task, which critics may regard as a non-rigorous feature of a Bayesian experiment design [39].

The comparison is made to frequentist approaches for which no such explicit subjectivity exists, suggesting that such an approach is more disciplined. This does not present as an honest comparison, however, because subjectivity is inherent to any frequentist design. Specification of a prior distribution over model parameters in the Bayesian paradigm needs to be compared to its direct equivalent in the frequentist approach for a fair assessment of this proposed subjectivity issue.

Experimental design is inherently subject to a series of subjective choices. A practitioner begins the specification of an experiment by choosing a set of values of interest, which immediately places a bias of attention on the space of the problem. Model structure and hyperparameters are then selected and tuned, and not always necessarily in a principled manner. Selection of metrics such as p-value threshold and confidence interval sizes are all subjective choices that the practitioner must make.

Consider a simple example involving the rolling of a six-sided die. To investigate whether the die in question is fair, a Bayesian practitioner may place a categorical prior over the distribution of single die-rolls with probability vector $p = \left[\frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}\right]$. The die can then be rolled to simulate the process sufficiently many times to achieve an estimate of the deviation between the expected probability vector and the realized sample averages. The frequentist approach might instead be to establish a null hypothesis with the same probability vector p and observe whether the returned p-value motivates rejection of the null hypothesis. The subjectivity is here abstracted outside the model specification: the researcher implicitly decides what sort of evidence might alter the model by specifying the null hypothesis and the p-value for which it is rejected.

This is the sort of implicit assumption that this thesis is trying to address. Training an ANN model with a given architecture might be akin to a prior belief claim that *this particular architecture is the correct one for the analysis*. Taking the Bayesian approach allows us to better express the uncertainty in our model selection. This also provides a benefit in terms of automating the model selection task, so that specification of the model can benefit from a principled data-driven approach, with fewer assumptions needed to be made at the outset of the model design.

3.3 Hierarchical Models

The components of Bayesian inference as presented above apply specifically to the case of standard statistical models. While not explicitly specified, above it is assumed that each model parameter θ_i as an element of the parameter vector is independent of any other parameter. When this is not the case, as in trans-dimensional inference, we can extend the Bayesian formulation to consider hierarchical models [40].

We make the claim that the Bayesian approach is a favourable one when there is uncertainty over model parameters, but the modelling uncertainty doesn't necessarily stop there. Especially with non-parametric models, it is difficult to claim that the specified joint prior distribution is uniquely appropriate for the modelling task. Each component of the model parameter vector is drawn from a known distribution with its own *hyperparameters*, which have thus far implicitly been assumed to be fixed values. If we are not adequately certain about these values, then we by definition have another "higher-level" source of uncertainty.

A principled solution would therefore be to specify distributions for these hyperparameters and treat them as random variables in the inference procedure. To do so is implement a two-level *hierarchical model*, wherein low-level model parameters are conditionally dependent on higher-level hyperparameters.

It is mathematically straightforward to incorporate such hyperparameters into the formulation of a posterior distribution over model parameters. It might be assumed that corresponding sets of model parameters may be drawn from the same distribution:

$$\boldsymbol{\theta} \sim p(\boldsymbol{\theta}|\alpha, \beta) \tag{3.10}$$

where α, β are the hyperparameters, and $p(\boldsymbol{\theta}|\alpha, \beta)$ is a joint prior distribution for $\boldsymbol{\theta}$. Given fixed values for α and β, equation 3.5 is naturally extended:

$$p(\boldsymbol{\theta}|\boldsymbol{D}) = \frac{\mathcal{L}(\boldsymbol{\theta}|\boldsymbol{D})p(\boldsymbol{\theta}|\alpha, \beta)}{p(\boldsymbol{D})} \tag{3.11}$$

When there is further uncertainty over specification of α and β, *hyperprior* distributions $p(\alpha)$ and $p(\beta)$ can be specified based on the high-level hyperparameters, which themselves can be specified to be drawn from respective higher-level hyperprior distributions.

$$p(\boldsymbol{\theta}|\boldsymbol{D}) = \frac{\mathcal{L}(\boldsymbol{\theta}|\boldsymbol{D})p(\boldsymbol{\theta} \mid \alpha, \beta)p(\alpha)p(\beta)}{p(\boldsymbol{D})} \tag{3.12}$$

This procedure can be repeated *ad infinitum* such that a model features hyperparameters "all the way down". Practically speaking, each additional level in a hierarchical model may increase a model's robustness with respect to uncertainty, but will consequently demand greater computational resources where training or inference computation is concerned. A weakly principled approach may be to define as many levels as is computationally feasible based on the available compute resources.

Non-parametric models in particular may benefit from a treatment of one or more levels of hyperparameter specification. In a hierarchical model, an appropriate degree of regularization may be automatically specified by the data [41]. The previously discussed difficulties associated with interpretability of complex ML models may motivate a need for more robustness than is provided by a model with no hyperparameters, and neural networks in particular are shown to greatly benefit from a two-level model [29].

3.4 Inference Methods

The posterior and posterior predictive distributions are technically defined at the moment of specification of the prior and likelihood, but are only available for analysis after some method of Bayesian inference has been performed. In exceptional cases, a convenient mathematical representation of the posterior may be obtainable; our exact posterior is otherwise analytically intractable, and may only be approximated or sampled from. Three main classes of approach exist for sampling from an analytically intractable posterior distribution: variational inference, Laplace approximations, and Monte Carlo methods.

3.4.1 Approximate Posterior Distributions

Variational Inference

Though we have departed from the classical ML paradigm, there is still a space for optimization. The main idea behind variational approaches [42] is to approximate the posterior distribution with a known tractable distribution $q(\boldsymbol{\theta}) \approx p(\boldsymbol{\theta}|\boldsymbol{D})$, and solve an optimization problem over some divergence metric between the true posterior and the estimate. Such a divergence would still require knowledge of the posterior, but a lower bound on the divergence can instead be minimized. This is referred to as the *evidence lower-bound* (ELBO).

Considering the log of the marginal likelihood, an expression for the lower bound is found using Jensen's inequality (equation 3.13), where $\mathrm{KL}(q||p)$ is the Kullback-Leibler divergence. We denote $q^*(\boldsymbol{\theta})$ as the optimal member of the family of distributions q (equation 3.14).

$$\begin{aligned}
\log p(\boldsymbol{D}) &= \log \int p(\boldsymbol{\theta}, \boldsymbol{D}) d\boldsymbol{\theta} \\
&= \log \int q(\boldsymbol{\theta}) \frac{p(\boldsymbol{\theta}, \boldsymbol{D})}{q(\boldsymbol{\theta})} \\
&\geq \int q(\boldsymbol{\theta}) \frac{p(\boldsymbol{\theta}, \boldsymbol{D})}{q(\boldsymbol{\theta})} \\
&= \int q(\boldsymbol{\theta}) \log p(\boldsymbol{\theta}, \boldsymbol{D}) d\boldsymbol{\theta} + \int q(\boldsymbol{\theta}) \log \frac{1}{q(\boldsymbol{\theta})} d(\boldsymbol{\theta}) \\
&= \log p(\boldsymbol{D}) - \mathrm{KL}(q(\boldsymbol{\theta})||p(\boldsymbol{\theta}|\boldsymbol{D}))
\end{aligned} \tag{3.13}$$

$$q^*(\boldsymbol{\theta}) = \underset{\boldsymbol{\theta} \in \boldsymbol{\Theta}}{\arg\min}\, \mathrm{KL}(q||p) \tag{3.14}$$

Casting the inference procedure as an optimization problem presents a number of advantages, which has led to reasonable popularity of variational approaches in ML. Variational inference integrates seemlessly with stochastic optimization [43], making it appropriate for the data-intensive analyses common to DL. It is still, however, only an approximate method, as the produced density is only close to the exact target posterior.

Laplace Approximations

Oftentimes, the most important aspect of a complex posterior distribution may be the mode - an area of particular high probability density in a given neighbourhood of the distribution. This is especially the case if the distribution is being assessed simply for an optimal point estimate, such as the MAP. In this situation, a Gaussian centered on the mode may be an appropriate starting point for an approximation of the posterior; this is the basis for the method of Laplace approximations, which extend Laplace's method for the approximation of complex integrals [44] to the application of Bayesian inference.

The posterior distribution may be more generally expressed as a density over a normalization constant as in equation 3.15.

$$p(\boldsymbol{\theta}|\boldsymbol{D}) = \frac{\tilde{p}(\boldsymbol{\theta}|\boldsymbol{D})}{\mathcal{D}} \tag{3.15}$$

Now, the posterior distribution is assumed to be approximately Gaussian around the MAP estimate $\boldsymbol{\theta}_{\text{MAP}}$:

$$p(\boldsymbol{\theta}|\boldsymbol{D}) \approx \frac{\Big|A\Big|}{(2\pi)^{d/2}} \exp\Bigg(-\frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_{\text{MAP}})^T A(\boldsymbol{\theta} - \boldsymbol{\theta}_{\text{MAP}})\Bigg) \tag{3.16}$$

where A is the Hessian Matrix of $\tilde{p}(\boldsymbol{\theta}|\boldsymbol{D})$ evaluated at $\boldsymbol{\theta}_{\text{MAP}}$.

As far as approximations go, the Laplace method is relatively analytically appealing, as the model evidence can be approximated. This approach is applied to Bayesian Neural Networks in early works by Mackay ([45], [41]), but are not as popular in current ML research as variational inference or MCMC. One interesting exception is the Integrated Nested Laplace Approximation (INLA) [46] technique for latent Gaussian models, for which ML applications offer an interesting future research direction.

3.4.2 Exact Posterior Distributions

Conjugate Priors

In specific and often trivial cases, analytical techniques allow for the computation of posterior distributions based on a given likelihood paired with a specific prior distribution known as a *conjugate prior*. The requirement for a conjugate prior is that the product of the functional forms of the likelihood and the prior distribution may be algebraically expressed in the form of a known distribution. The consequent posterior is as convenient to work with as the known distribution, and is specified exactly across the entire support of the variable.

The analytically computed posterior will be of the same form as the prior distribution with updated hyperparameters. Table 3.1 displays examples of likelihood functions with appropriate conjugate priors. We emphasize entry one corresponding to a normal likelihood with precision τ as a popular result used in related works [29], [47] and explored in our experiments.

Likelihood	**Conjugate Prior**	**Prior Hyperparameters**	**Posterior Hyperparameters**
Normal (μ,τ)	Gamma	Shape α, Scale β	Shape $\alpha + \frac{n}{2}$, Scale $\beta + \frac{\sum_{i=1}^{n}(x_i - \mu)}{2}$
Exponential (λ)	Gamma	Shape α, Scale β	Shape $\alpha + n$, Scale $\beta + \sum_{i=1}^{n} x_i$
Poisson (λ)	Gamma	Shape α, Scale β	Shape $\alpha + \sum_{i=1}^{n}$, Scale $\beta + n$
Categorical ($\boldsymbol{p}$)	Dirichlet	Conc. $\boldsymbol{\alpha} \in \mathcal{R}^k$	Conc. $\boldsymbol{\alpha} + (c_1, ..., c_k)$, $c_i =$ # of observations for category i

TABLE 3.1: Examples of conjugate priors

Conjugate priors are not often applicable in machine learning settings for the low level model parameters, for which the distributions of interest are characterized by complex structures and high-dimensional parameter spaces. They are, however, a convenient approach to higher level hyperparameters for fast computational updating via Gibbs sampling (see section 4.4).

Markov chain Monte Carlo

Markov chain Monte Carlo is the only inference method that allows for sampling from the exact posterior distribution when it is analytically intractable. A stochastic process (a Markov chain) is defined to iteratively draw samples from different elements of the support of the random variable, visiting each value a number of times proportional to its probability in the asymptotic limit of iterations. Practically speaking, only a finite number of these iterations can be obtained, but a sufficiently large number can provide a tenable approximation to the exact posterior distribution and consequent point or interval estimates for the model parameters.

A full treatment of MCMC methods is presented in the following chapter, including those which make use of gradients for efficient sampling of the posterior distribution, and trans-dimensional methods that can explore a join posterior of model specifications with differing numbers of randomly distributed model parameters.

3.5 Learning as Inference

Machine learning techniques can be interpreted through the framework of statistical inference. To view learning in this manner, the models to be trained must be interpreted probabilistically. This can be achieved through the use of so-called *ensemble methods* [48], which extend standard learning algorithms to principled schedules of sample values for hyperparameters of interest, or through the Bayesian framework. The focus of this work is on the latter, though notes are first presented on ensemble methods to build intuition.

3.5.1 Ensemble Methods

In the interest of robustness, we may be interested in a class of ANN models with distinct parameterizations, as opposed to one trained network. A set of realizations (i.e., trained networks) from this class is dubbed an ensemble, and by treating a learning task to each model within this set, we can perform inference over model parameters and generate measures of uncertainty over our $\hat{\boldsymbol{y}}$ vector of model outputs.

Formally speaking, we begin by declaring $\boldsymbol{\mathcal{M}} = \{\mathcal{M}_1, \mathcal{M}_2, ..., \mathcal{M}_m\}$ where each $\mathcal{M}_i$ is characterized by an ANN function $f(\boldsymbol{\theta}_i, \boldsymbol{x})$. The parameter vectors $\boldsymbol{\theta}_i = \{\boldsymbol{W}, \boldsymbol{b}\}$ are considered to be drawn from some distribution $\boldsymbol{\Theta}$. Network outputs $\hat{\boldsymbol{y}}_i = f(\boldsymbol{\theta}_i, \boldsymbol{x})$ can then be drawn from each network, and an average response $\hat{\boldsymbol{y}} = \frac{1}{m}\sum_{i=1}^{m}\hat{\boldsymbol{y}}_i$ is calculated as an approximate *expectation* - a robust estimate of the true response vector (equation 3.17).

$$E[\boldsymbol{y}] = \int \boldsymbol{y} f(\boldsymbol{y}) d\boldsymbol{y} \tag{3.17}$$

How many networks should there be in a well-defined ensemble? Two unsatisfying answers might be as many as we deem appropriate, or as many as are computationally feasible. The former answer corresponds to situations where we are uncertain about a particular set of architecture specifications. We may know that a network would benefit from either 2 layers or 3, but can not say which of the two might be more optimal. We might instead wish to define 32, 64 or 128 neurons in each layer, or we may seek to consider networks with different activation functions. Alternatively, we may train an entire table of network specification permutations, one such approach to Neural Architecture Search [49].

This discussion of how an optimal ensemble is defined hints towards a Bayesian approach. It might be argued that the optimal ensemble size is the asymptotic limit - an infinite number of networks, over which an expected distribution can be integrated.

3.5.2 Bayesian Neural Networks

Bayesian Neural Networks (BNNs) [29],[45], [50] are non-parametric models structured in the same manner as ANNs, for which probability distributions are placed over the weight and bias matrices corresponding to each layer of the network. A joint posterior distribution of network parameters can then be computed by declaring prior distributions, establishing the appropriate likelihood for the paired observational and response data given the model task, and executing an appropriate approximate inference technique.

A formal definition of a BNN model follows from the definition of an ANN in section 2.2.1 and the components of Bayesian inference in section 3.2. An ℓ-layered neural network framework $\boldsymbol{y} = f(\boldsymbol{x}, \boldsymbol{\theta})$ is declared for fitting responses $\boldsymbol{y}$ to feature vectors $\boldsymbol{x}$ based on parameterizations $\boldsymbol{\theta} = \{\boldsymbol{W}, \boldsymbol{b}\}$, respectively the weights and biases of the linear transformations $\boldsymbol{z}_j = h_j(\boldsymbol{z}_{j-1}, \boldsymbol{W}_j, \boldsymbol{b}_j)$ for each layer $j = 1, ..., \ell$, where $\boldsymbol{z}_1 = \boldsymbol{x}$. Activation functions $g_j(\boldsymbol{z})$ are applied after each linear transformation, and will typically be taken to be equivalent across all layers excluding perhaps the final layer, depending on the nature of the analysis.

The parameter vector $\boldsymbol{\theta}$ is assumed to contain components θ_k that are iid random variables, the joint prior distribution for which is declared such that $\boldsymbol{\theta} \sim p(\boldsymbol{\theta})$. A likelihood function for the parameters based on the observed data is selected based on the network's role as a classification or regression model and will be of the form $\mathcal{L}(\boldsymbol{\theta}|\boldsymbol{x}, \boldsymbol{y})$. The marginal distribution of the data is discarded, and a posterior distribution of the model parameters is achieved through equation 3.5.

The end result of inference over a BNN is a posterior distribution of network parameterizations. To draw a single sample from this distribution is to generate a single ANN that is considered to have effectively completed its training regimen. The fitted response data as modeled should provide a reasonable estimate of the training labels[3]. To draw n samples from the posterior would be to generate an ensemble of these ANNs, each of which may contribute estimated labels for the training data as one sample estimate of the true response. More generally, samples contributing to an expectation of the responses $\boldsymbol{y}$ are drawn from the posterior predictive distribution as a means of obtaining point or interval estimates.

The BNN approach offers several advantages over the standard machine learning approach. Inference procedures offer the same benefits as ensemble methods, but these benefits may be thought of as being "built-in" to the inference procedure, and need not be approximated through the addition of auxiliary methods and corresponding hyperparameters. Select advantages are outline below:

1. Robust, built-in generalization

   A known result of BNNs is that the use of zero-centered Gaussian priors over weight parameters (*see section 3.5.3*) induces an equivalence to L2-regularization

[3]The distribution of trained networks may not be defined such that every sample will correspond to a network that has been trained "effectively". Low-probability draws from the *tails* of the distribution (those with a relatively low posterior score) may not achieve optimal results based on appropriate metrics, such as classification accuracy or regression MSE.

via the inclusion of a weight decay penalty [51]. Such a penalty term is included in the loss function of an ANN as $\lambda \|\boldsymbol{W}^2\|$, where $\lambda > 0$ is some coefficient affecting the strength of the penalty on the squared norm of the weight parameters. Networks with smaller weights are therefore favoured in producing a trained ANN model.

For a BNN, the posterior distribution of network parameterizations corresponds to an infinite ensemble of networks that treat the data to all possible functional representations $\boldsymbol{y} = f(\boldsymbol{x})$ weighted by corresponding posterior scores, for which the contribution of a distribution centered at zero will similarly favour parameter absolute values closer to zero [41].

2. Uncertainty measures built-in

   The expectation represents the first-order moment of the posterior distribution; the second-order moment is referred to as the variance. The variance of each component of the parameter vector provides as a metric the spread of likely values for the corresponding model parameter, indicating how widely distributed random realizations of the marginal posterior may be.

   Variance for the posterior distribution allows for statements about the certainty of estimates for network parameters, but does not directly provide predictive insight. Variance for the predictive distribution, however, can be extremely beneficial in represented the limitations of a neural network's predictive capabilities. A large variance associated with a fitted data point can indicate the degree of caution that should be regarded when relying on a network's predictions for real-world applications.

3. Interpretability of network parameters

   Referring to the discussion of black box models in section 2.2, it is not always clear what precisely motivates the parameterization of a well-specified non-parametric model. The posterior distribution of parameters facilitates inference about the nature of the model, in that draws from the posterior effectively produce a dataset of parameterizations. Analysis of this dataset may provide insight into the model at hand; we are essentially turning statistical analysis inward on itself.

   Treating neural networks to Bayesian inference therefore proposes an inherent opportunity to understand such models, in parallel to modelling the data according to predictive motivations as in a classical machine learning approach. The posterior predictive distribution provides a fitted estimate of the data while the posterior distribution lends clarity to the model itself, such that we don't merely return *how* to represent the data, but also information about *why* those representations may be reasonable.

Essentially, a BNN is a distribution of ANNs which may be learned through one of the three inference techniques previously introduced.

3.5.3 Priors for BNNs

Prior distributions in Bayesian inference allow for the injection of previously acquired domain knowledge into the modelling task, but this is not an immediately intuitive proposition when dealing with non-parametric models [52]. To impose a distribution over the weights and biases of a neural network model is to implicitly define expectations about the nature of these parameters, such as their restricted domain, and the regions of the real number line corresponding to high probability mass for parameter values.

Discussion of BNN priors dates back to Mackay [45],[53],[41] and Neal [29]. The consensus in the literature to date largely agree with their forward approach of employing zero-centered Gaussian distributions for both weights and biases, with variance parameters σ_i^2 declared for sets of parameters i corresponding to similar roles. Neal for example declares separate variances respectively for sets of hidden node weights, hidden node biases, output node weights, and output node biases, treating each corresponding Gaussian prior to an appropriate width based on the previous layer size.

Further discussion of BNN priors is included in section 5.1.1.

3.5.4 BNN Likelihoods

The likelihood function must be appropriately specified based on the nature of the analysis for a given BNN model. In this work, BNNs are used for both classification and interpolation tasks. Suitable likelihoods extend from equivalent analyses via Bayesian regression (BR) and Bayesian logistic regression (BLR).

Interpolation Networks

Each fitted value y_i is assumed to be drawn from a normal distribution centered on the network output $\hat{y}_i = f(\boldsymbol{x})$ with noise term σ_y. This corresponds to the noise terms $\epsilon_i = y_i - \hat{y}_i$ in standard regression, which are assumed to be drawn $\epsilon_i \sim N(0, \sigma_y)$. σ_y is often considered to be a hyperparameter; it is marginalized over in computing the predictive distribution.

Classification Networks

In the binary case of two labels, a sigmoid non-linearity (eq. 2.5) σ is applied to the network output $\hat{\boldsymbol{y}}_i = \sigma(f(\boldsymbol{x}_i))$. This corresponding "score" vector serves as the probability parameter for a Bernoulli distribution concerning whether y_i belongs to class $j = \{0, 1\}$, $y \sim Ber(\hat{y}_i)$.

For multiple classes, a softmax non-linearity (eq. 2.6) generates a probability vector based on the number of output nodes $j = 1, 2, ..., m$ where m corresponds to the number of candidate classes. The datapoint y_i is assumed to be drawn from a categorical distribution with probability vector $\hat{\boldsymbol{y}}_i$, $y_i \sim Cat(\hat{\boldsymbol{y}}_i)$.

3.6 Model Summary

The Bayesian neural network has been introduced as a distribution of parameterizations over a given ANN architecture. The key feature of the Bayesian approach to ANN models that will be exploited in this work is the ability to specify a neural network for which the parameterizations will be learned alongside architectural specifications such as the number of layers in the network, and the number of hidden nodes within each layer. In this sense, our model will not concern a single network architecture, but a *joint distribution over possible architectures and parameterizations* - an opportunity that is known to exist, but has received minimal research attention to date [54]. In the next chapter, relevant MCMC algorithms are reviewed in detail as components of the trans-dimensional inference engine for such models.

Chapter 4

Markov Chain Monte Carlo

The claim can be made that Bayesian inference is difficult because integration is difficult [55]. In the Bayesian paradigm, an expectation as integration over an analytically intractable posterior distribution - for example, a distribution over the weights and biases of a BNN - must be approximated by sophisticated, computationally expensive numerical methods. Integration is also necessary in determination of the posterior predictive distribution, and moments thereof.

More generally speaking, analytically solvable integrals may be seen as rare exceptions in the space of all possible integrable functions. Consider the definite integral:

$$I = \int_{\boldsymbol{\mathcal{A}}} f(\boldsymbol{x}) d\boldsymbol{x} \tag{4.1}$$

for some function f over a d-dimensional vector $\boldsymbol{x}$, where $\boldsymbol{\mathcal{A}}$ is a subset $\boldsymbol{\mathcal{A}} \subset \mathbb{R}^d$ with volume $V(\mathcal{A}) = \int_{\boldsymbol{\mathcal{A}}} d\boldsymbol{x}$. Unless I happens to correspond to a convenient analytical result by the fundamental theorem of calculus over f, the integral is considered analytically intractable and must be computed through numerical means.

4.1 Monte Carlo Integration

Monte Carlo integration (MCI) [56] is an approach to integral estimation that makes use of sufficiently sized stochastic samples of $f(\boldsymbol{x})$. Assuming that $X_1, X_2, \ldots$ can be simulated from the same distribution as $\boldsymbol{x}$, an estimation of I by MCI can then be defined as in equation 4.2.

$$\hat{I} = V \frac{1}{n} \sum_{i=1}^{n} f(X_i) \tag{4.2}$$

By the law of large numbers, $\hat{I}$ almost surely converges to I given a sufficient number of samples (equation 4.3).

$$\lim_{n \to \infty} \hat{I} = I \tag{4.3}$$

In the case of a probability distribution for a random variable, the volume is implicit in the functional definition of the distribution so long as it is properly defined to integrate to one. With a Bayesian posterior predictive distribution, the necessary normalizing constant is the marginal evidence $p(\boldsymbol{D})$, which is often disregarded for analytical convenience. An MCI estimate of the unnormalized predictive distribution is therefore obtainable as in equation 4.4 with posterior samples $\boldsymbol{\theta}^{(i)} \sim p(\boldsymbol{\theta}|\boldsymbol{D})$.

$$\begin{aligned} p(\boldsymbol{y}|\boldsymbol{x}) &= \int_{\boldsymbol{\Theta}} f(\boldsymbol{y}|\boldsymbol{x},\boldsymbol{\theta})p(\boldsymbol{\theta}|\boldsymbol{D})d\boldsymbol{\theta} \\ &\approx \frac{1}{n}\sum_{i=1}^{n} p(\boldsymbol{y}|\boldsymbol{x},\boldsymbol{\theta}^{(i)}) \end{aligned} \tag{4.4}$$

MCI is a convenient tool for approximating functions of random variables, so long as it possible to obtain a sufficient number of samples. For complex, high-dimensional posterior distributions, this will likely not be immediately possible.

In this chapter, the method of Markov chain Monte Carlo is reviewed as an approach to sample exactly from an analytically intractible posterior distribution. A summary of MCMC is first presented as the basis for the case of trans-dimensional models. We also present gradient-based methods so that we may scale the trans-dimensional algorithm implementation up to the high-dimensional models that will be explored in the experiments conducted in chapter 6.

4.2 Inference by Markov Chain Monte Carlo

A Markov chain is a series of random variables known as states, $X^{(0)}, X^{(1)}, \ldots, X^{(n)}$ with the property that the next state, $X^{(n+1)}$ is only dependent on $X^{(n)}$, that is to say; $X^{(n+1)}$ is conditionally independent on $X^{(n)}$ for any other state of the chain:

$$p(X^{(n+1)}|X^{(n)}, X^{(i)}) = P(X^{(n+1)}|X^{(n)}) \quad \forall i : i \in \{i : n\} \tag{4.5}$$

The definition of a Markov chain requires only an initialization state $X^{(0)}$ and a transition program, which in the discrete case can be defined as a matrix T such that t_{ij} represents the transition probability $P(X^{(n+1)} = x_j|X^{(n)} = x_i)$, where $i = 1, \ldots, n$, and $j = 1, \ldots, n$, i.e. the support of all possible states. More generally and for the continuous case, $T(x, x')$ is a transition kernel representing the probability of moving to state x' given that the chain is in state x, $P(X^{(n+1)} = x'|X^{(n)} = x)$.

MCMC uses a Markov chain with certain properties to sample from a specified target distribution $\pi(x)$. Somewhat trivially, it must be guaranteed that the initial state $X^{(0)}$ is within the support of the desired distribution, i.e. if $X^{(0)} = x_0$, then $\pi(X = x_0) > 0$. With regards to the transition program, MCMC requires that the Markov chain defined by T is *stationary*, such that for every positive integer m, the distribution of the m-tuple $(X_{n+1}, \ldots, X_{n+m})$ does not depend on n. Stationarity is obtained in the design of MCMC algorithms most commonly by ensuring that the chain is *reversible*, meaning for any i and j the joint distributions of the states $\pi(X_{n+1}, \ldots, X_{n+m}) = \pi(X_{n+m}, \ldots, X_{n+1})$.

Algorithm 1 Markov Chain Monte Carlo [57]

```
1: function MCMC($\boldsymbol{\theta}$)
2:     for n=1 to N do
3:         $\boldsymbol{\theta}^* \leftarrow$ update($\boldsymbol{\theta}$)
4:     end for
5:     return $\boldsymbol{\theta}$
6: end function
```

MCMC is a class of algorithms, not an individual algorithm. The provided pseudocode in algorithm 1 is intended to demonstrate just how general an MCMC program is. The update function perturbs the current state $X^{(n-1)}$ to produce the next state, $X^{(n)}$. Any combination of stochastic and deterministic procedures to produce the new state can be employed, provided that such a function does not rely on any other information about the state of the program outside the scope of $X^{(n-1)}$. The output of an MCMC program is a series of states $\{X^{(i)}\}_{i=1}^{n}$ that represent samples from the stationary distribution of the Markov chain, $\pi(X)$.

Essentially, a Markov chain is used to explore a complex distribution, with states of the chain representing samples from the distribution. An MCI estimate is performed on the states of the chain to return an estimate of the expectation of the distribution, and realizations of the chain (i.e. states sampled from the chain) will have an approximate marginal distribution π [58].

Detailed Balance

The goal is to define $T(x', x)$ such that it makes the Markov chain *ergodic*. Ergodicity is a property of a Markov chain such that any state in the stationary distribution of the chain can be reached in some finite n amount of steps. A convenient way to achieve this that is most generally applied to MCMC in practice is by satisfying *detailed balance* (equation 4.6). If $\pi(x)$ is the target distribution of the chain, then we want equation 4.6 to hold.

$$\pi(x)T(x, x') = \pi(x')T(x', x) \tag{4.6}$$

Satisfying detailed balance and therefore reversibility implies stationarity of the Markov chain, but the reverse is not true. Recent research investigates MCMC algorithms that meet the stationarity requirement without satisfying detailed balance [59], but the overwhelming majority of MCMC applications, including those of interest in this thesis, make use of this result.

4.2.1 MCMC for Bayesian Inference

In the case of Bayesian inference, the target stationary distribution is the posterior distribution over the model parameters $\boldsymbol{\theta}$ conditioned on the observed data $\boldsymbol{D} = \{\boldsymbol{x}, \boldsymbol{y}\}$ (equation 3.5). The appeal of MCMC inference is that if $T(\boldsymbol{\theta}', \boldsymbol{\theta})$ satisfies certain properties, the chain is theoretically guaranteed to sample from the exact unnormalized posterior distribution obtained through Bayesian inference over a given likelihood and specified prior distribution. As such, Monte Carlo estimates from the distribution are asymptotically unbiased. As explored in chapter 3, the fact that the distribution is unnormalized does not matter in terms of estimating relative probability densities of the parameter vector of interest.

It is worth reflecting on the power of this guarantee. Outside of the set of analytically computable posterior distributions (such as those obtained through the use of conjugate priors in section 3.4.2), MCMC is the only inference approach that allows for *unbiased* sampling from the (unnormalized) posterior distribution [60]. Laplacian and variational approaches return approximations, which can be perfectly serviceable for the intent of the given inference, but only MCMC will preserve the relative shape of the posterior distribution in its entirety as sampling proceeds. This theoretical advantage comes with a major caveat, however, in terms of computational feasibility.

Usage of MCMC in this thesis is restricted to the application of Bayesian inference, and we therefore continue with treatment of $\boldsymbol{\theta}$ as the parameter vector for a Bayesian model and consequently the state of the Markov chain.

4.3 The Metropolis-Hastings Algorithm

The seminal paper on MCMC developed the precursor to the most popular and widely-used MCMC algorithm in 1953 [61]. The Metropolis algorithm was introduced as a simulation for physical phenomena, and later generalized to statistics problems for sampling a complex distribution [62].

In the Metropolis-Hastings algorithm (algorithm 2), a candidate state is accepted or rejected in favour of the current state dependent on an acceptance probability which is calculated to satisfy the detailed balance requirement.

This means that an acceptance probability for a proposed state $\boldsymbol{\theta}'$ is determined by the transition to the proposed state from the current state and its reverse simultaneously. If $\alpha(\boldsymbol{\theta}', \boldsymbol{\theta})$ represents the acceptance probability of such a move, the acceptance probability for the reverse move will be $\alpha(\boldsymbol{\theta}, \boldsymbol{\theta}')$. The transition kernel in equation 4.6 is then defined by the proposal distribution q weighted by its corresponding acceptance probability, and detailed balance is updated accordingly (equation 4.7). The acceptance probability

Algorithm 2 Metropolis-Hastings

```
1: function MetropolisHastings($\boldsymbol{\theta}$)
2:     $\boldsymbol{\theta}' \sim q(\boldsymbol{\theta}', \boldsymbol{\theta})$
3:     $\alpha = \min\left\{1, \frac{\pi(\boldsymbol{\theta}')q(\boldsymbol{\theta},\boldsymbol{\theta}')}{\pi(\boldsymbol{\theta})q(\boldsymbol{\theta}',\boldsymbol{\theta})}\right\}$
4:     $u \sim$ unif(0,1)
5:     if $u \leq \alpha$ then
6:         return $\boldsymbol{\theta}'$                    ▷ Accept the newly proposed state
7:     else
8:         return $\boldsymbol{\theta}$                     ▷ Reject the proposed state, register the current state
9:     end if
10: end function
```

follows from this description of detailed balance (equation 4.8).

$$\int_{(\boldsymbol{\theta},\boldsymbol{\theta}')\in\Theta} \pi(\boldsymbol{\theta})q(\boldsymbol{\theta},\boldsymbol{\theta}')\alpha(\boldsymbol{\theta},\boldsymbol{\theta}')d\boldsymbol{\theta} = \int_{(\boldsymbol{\theta}',\boldsymbol{\theta})\in\Theta} \pi(\boldsymbol{\theta}')q(\boldsymbol{\theta}',\boldsymbol{\theta})\alpha(\boldsymbol{\theta}',\boldsymbol{\theta})d\boldsymbol{\theta}' \tag{4.7}$$

$$\alpha(\boldsymbol{\theta},\boldsymbol{\theta}') = \min\left\{1, \frac{\pi(\boldsymbol{\theta}')q(\boldsymbol{\theta},\boldsymbol{\theta}')}{\pi(\boldsymbol{\theta})q(\boldsymbol{\theta}',\boldsymbol{\theta})}\right\} \tag{4.8}$$

If the distribution q is equivalent for both the forward and reverse moves, the second term in equation 4.8 reduces to the quotient of posterior scores for the proposed state over the current state, as in equation 4.9.

$$\alpha(\boldsymbol{\theta},\boldsymbol{\theta}') = \min\left\{1, \frac{\pi(\boldsymbol{\theta}')}{\pi(\boldsymbol{\theta})}\right\} \tag{4.9}$$

Notice that the acceptance probability is equal to 1 whenever the posterior score is superior for the proposed state. This means that any proposal representing an improved model parameterization in accordance with the prior distribution and likelihood will always be accepted. The inverse does not necessarily hold: a proposed state with a lower posterior score will be accepted at a rate based on the quotient of its score compared to the current state. This is reflected in the uniform threshold variable u in algorithm 2.

The practitioner is afforded a good deal of flexibility in defining the proposal distribution $q(\boldsymbol{\theta},\boldsymbol{\theta}')$. The default approach is considered to be a *random walk*, wherein the perturbation to the d-dimensional parameter vector $\boldsymbol{\theta}$ is some small addition of noise - perhaps Gaussian - so that $\boldsymbol{\theta}' := \boldsymbol{\theta} \oplus \boldsymbol{\epsilon}$[1], with $\boldsymbol{\epsilon} \sim N(\mathbf{0}, \boldsymbol{I})$, where $\boldsymbol{I}$ is the dXd identity matrix. A random-walk is one such case where the proposal distribution q will be equivalent for both the forward and reverse moves, leading to the acceptance probability as in equation 4.9.

[1] $\oplus$ denotes element-wise vector addition.

4.4 Gibbs Sampling

In consideration of a vector of model parameters, it may be useful to propose an update to the state of the Markov chain that perturbs only one component while the others are held to be fixed. Sampling from the conditional distribution of one particular component θ_i, or perhaps some subset $\boldsymbol{\theta_i} = \boldsymbol{\theta}_{\{a,b,c,\ldots\}}$ with respect to the remaining components, is known as a Gibbs update [63]. When combined with a Metropolis-Hastings acceptance step, this technique is known as variable-at-a-time Metropolis-Hastings[2].

Gibbs sampling is particularly applicable to hierarchical models. Updates can be performed iteratively for each level of the model, beginning with the highest level - those hyperparameters which do not rely on other random variables in the model definition. This is a sensible approach, albeit with a trade-off in terms of increased computational demand, as the exploration of the full parameter space now takes place over several steps through lower-dimensional subspaces. It also allows for different proposal mechanisms to be defined for each of the v blocks of variables, of the form $\boldsymbol{\theta}'_{\boldsymbol{i}} \sim q_i(\boldsymbol{\theta}'_i|\boldsymbol{\theta}_{-i}, \boldsymbol{\theta}_i)$, for all $i \in \{1, \ldots, v\}$.

Algorithm 3 demonstrates the general Gibbs sampler as a special case of Metropolis-Hastings, which forms the "backbone" of the trans-dimensional inference procedure for the hierarchical BNN model presented in chapter 5.

Algorithm 3 Gibbs Sampling

```
1:  function GIBBS($\boldsymbol{\theta}$)
2:      for i=1:$v$ do                                   ▷ $v$; the number of partitions of $\boldsymbol{\theta}$
3:          $\boldsymbol{\theta}'_i \sim q_i(\boldsymbol{\theta}'_i|\boldsymbol{\theta}_{-i}, \boldsymbol{\theta}_i)$
4:          $\boldsymbol{\theta}' = \{\boldsymbol{\theta}'_i, \boldsymbol{\theta}_{-i}\}$
5:          $\alpha = \min\left\{1, \frac{\pi(\boldsymbol{\theta}')q_i(\boldsymbol{\theta}_i|\boldsymbol{\theta}_{-i},\boldsymbol{\theta}'_i)}{\pi(\boldsymbol{\theta})q_i(\boldsymbol{\theta}'_i|\boldsymbol{\theta}_{-i},\boldsymbol{\theta}_i)}\right\}$
6:          $u \sim$ unif(0,1)
7:          if $u \leq \alpha$ then
8:              $\boldsymbol{\theta} = \boldsymbol{\theta}'$                  ▷ Accept the proposed components
9:          else
10:             $\boldsymbol{\theta} = \boldsymbol{\theta}$                   ▷ Reject the proposed components
11:         end if
12:     end for
13:     return $\boldsymbol{\theta}$
14: end function
```

[2]"Gibbs Sampling" occasionally implicitly refers to variable-at-a-time Metropolis-Hastings [57]

4.5 Reversible Jump MCMC

The Metropolis-Hastings algorithm is generalized to account for a parameter vector of unfixed size with the introduction of the Metropolis-Hastings-Green (MHG) algorithm, commonly referred to as Reversible Jump Markov chain Monte Carlo (RJMCMC) [64]. The original author behind this approach, Peter Green, offers the oft-quoted motivation for RJMCMC as its applicability to "statistical problems where the number of things you don't know is one of the things you don't know" [65].

This topic is presented as follows.

1. The strategy for introducing jumps in a fixed dimension is introduced
2. A Jacobian correction to the acceptance probability is discussed
3. The trans-dimensional case is established via the concept of dimension-matching
4. Difficulties associated with the practical implementation of RJMCMC are briefly reviewed

4.5.1 Fixed-Dimension Reversible Jumps

Consider a Markov chain in state i with corresponding parameter vector $\boldsymbol{\theta}$. For state $i+1$, a proposed parameter vector $\boldsymbol{\theta}'$ is sought such that the acceptance probability α will be "reasonable". In order to achieve this, a proposal distribution $q(\boldsymbol{\theta}'|\boldsymbol{\theta})$ is declared and a proposal is formulated through some combination of random sampling and deterministic transformations. To satisfy detailed balance, only reversible chains are considered; the probability of moving from a state $\boldsymbol{\theta} \in \boldsymbol{\Theta}$ to $\boldsymbol{\theta}' \in \boldsymbol{\Theta}'$ must be equal to the reverse move. The burden of satisfying this condition is placed on the derivation of an appropriate acceptance probability for the proposal.

At the current state $\boldsymbol{\theta}$, we begin by drawing n random numbers $\boldsymbol{u}$ from a known joint density, $g(\boldsymbol{u})$. Our proposal distribution is then a deterministic function h acting on the current state and $\boldsymbol{u}$, proposing a new state for each (equation 4.10).

$$(\boldsymbol{\theta}', \boldsymbol{u}') = h(\boldsymbol{\theta}, \boldsymbol{u}) \tag{4.10}$$

The proposed adjustment to the random numbers, $\boldsymbol{u}'$, are the n-dimensional random numbers from density $g'(\boldsymbol{u}')$ that correspond to the reverse move from $\boldsymbol{\theta}'$ to $\boldsymbol{\theta}$, and the associated inverse function h'. The detailed balance requirement from equation 4.6 can be updated with a change-of-variable formula, so long as h and h' are differentiable.

$$\int_{(\boldsymbol{\theta},\boldsymbol{\theta}')\in\Theta} \pi(\boldsymbol{\theta})g(\boldsymbol{u}), \boldsymbol{\theta}')\alpha(\boldsymbol{\theta}, \boldsymbol{\theta}')d\boldsymbol{\theta}d\boldsymbol{u} = \int_{(\boldsymbol{\theta}',\boldsymbol{\theta})\in\Theta} \pi(\boldsymbol{\theta}')g(\boldsymbol{u}'), \boldsymbol{\theta})\alpha(\boldsymbol{\theta}', \boldsymbol{\theta})d\boldsymbol{\theta}'d\boldsymbol{u}' \tag{4.11}$$

$$\pi(\boldsymbol{\theta})g(\boldsymbol{u}),\boldsymbol{\theta}')\alpha(\boldsymbol{\theta},\boldsymbol{\theta}') = \pi(\boldsymbol{\theta}')g(\boldsymbol{u}'),\boldsymbol{\theta})\alpha(\boldsymbol{\theta}',\boldsymbol{\theta}) \left| \frac{\delta(\boldsymbol{\theta}',\boldsymbol{u}')}{\delta(\boldsymbol{\theta},\boldsymbol{u})} \right| \tag{4.12}$$

The last factor in equation 4.12 is the Jacobian of the transformation $(\boldsymbol{\theta}',\boldsymbol{u}') = h(\boldsymbol{\theta},\boldsymbol{u})$. The Metropolis-Hastings style acceptance probability follows in equation 4.13.

$$\alpha(\boldsymbol{\theta},\boldsymbol{\theta}') = \min\left\{1, \frac{\pi(\boldsymbol{\theta}')g'(\boldsymbol{u}')}{\pi(\boldsymbol{\theta})g(\boldsymbol{u})} \left| \frac{\delta(\boldsymbol{\theta}',\boldsymbol{u}')}{\delta(\boldsymbol{\theta},\boldsymbol{u})} \right| \right\} \tag{4.13}$$

The "reversible jump" of RJMCMC doesn't directly refer to the trans-dimensional aspect of the proposal mechanism. The MHG algorithm introduces the general procedure for making a jump to any part of a model's parameter space, and the corresponding method for determining the acceptance ratio in the metropolis step of the inference procedure. This formally extends the design framework of the proposal distribution beyond the random-walk approach. The ability to handle discrete parameters, such as model indicators representing changes in the dimension of the model parameter, happens to be a convenient consequence of this approach to MCMC. RJMCMC proposals require a Jacobian correction, as well as a technique referred to as dimension matching, in order to maintain reversibility and thus detailed balance.

4.5.2 Proposals with Jacobian Corrections

The Jacobian correction for the deterministic transformation aspect of the proposal transition kernel. This provides more flexibility to construction of the proposal program for the Markov chain. The mapping h from $(\boldsymbol{\theta},\boldsymbol{u})$ to $(\boldsymbol{\theta}',\boldsymbol{u})'$ is a *diffeomorphism* from one manifold to another, for which the Jacobian corrects for a difference in volume in order to main detailed balance.

Each component of the joint vector $(\boldsymbol{\theta},\boldsymbol{u})$ is generated based on the mapping h, dependent perhaps on itself and other components from the same vector. The entries δ_{ij} in the Jacobian matrix correspond to a differentiation of component i with respect to component j. Therefore, if each component's mapping is only dependent on itself, the non-diagonal entries will be zero and the determinant will reduce to 1, eliminating the Jacobian from the acceptance probability equation.

Proposals can therefore be employed in special cases of the MHG algorithm that do not necessarily employ the Jacobian correction for custom-tailored "jumps". It is possible and occasionally convenient to move between dimensions for which the invertible proposal kernel uses only the identity function on the existing state of the chain and the randomly-drawn auxiliary variables. It is important to point out this distinction between the two most salient features of RJMCMC as a remark in response to the popular notion that RJMCMC is difficult to implement [66].

4.5.3 Dimension Matching

Now that we have defined a procedure for within-dimension "jump"-based proposals, we need only introduce dimension matching as a way to extend the algorithm to so-called across-dimension proposals.

Consider now the current state of the Markov chain i corresponds to a parameter vector $\boldsymbol{\theta}$ with d_1 components, that is, $\boldsymbol{\theta}$ is a d_1-dimension vector. The proposal for state $i+1$ will be a d_2 dimension parameter vector $\boldsymbol{\theta}'$, such that $d_2 > d_1$. In order to maintain detailed balance, it is required that our proposal distribution $q(\boldsymbol{\theta}'|\boldsymbol{\theta})$ will treat an output vector that is the same dimension as the vector that is being conditioned on. To do so, we introduced auxiliary variables u_1, u_2 with respective dimensions n, m and require that the condition of dimension matching (equation 4.14) is met.

$$d_1 + n = d_2 + m \tag{4.14}$$

It is emphasized that m may be equal to 0, representing a proposal for which auxiliary variables are only included in the dimension matching procedure to "grow" the dimension of the current state. A similar condition exists that n may be equal to 0 when the proposal represents a lower-dimensional model, i.e. $d_2 < d_1$.

4.5.4 RJMCMC in Summary

RJMCMC allows for a trans-dimensional approach to MCMC. The method of defining reversible "jumps" across parameter vectors of different dimensions allows for inference over statistical models with an unfixed number of parameters while preserving the detailed-balance condition required to guarantee the asymptotic convergence behaviour of MCMC algorithms.

A naive RJMCMC sampler is quite straightforward - the trick is to extend the transition kernel such that a *reasonable acceptance rate* is achieved for proposed across-dimension jumps. The computational difficulties associated with MCMC in general become yet more pronounced when moves correspond to a change in parameter dimension, as in most applications the chain is overwhelmingly likely to move to an area of low posterior density, and will experience an arbitrarily small chance of being accepted. This challenge is discussed in detail with regards to the implementation of RJMCMC on the extended BNN model in chapter 5. A general RJMCMC procedure is demonstrated in algorithm 4.

Algorithm 4 Reversible Jump Markov chain Monte Carlo

```
1:  function RJMCMC($\boldsymbol{\theta}, h$)
2:      $k' \sim p(k)$                          ▷ Propose new parameter vector dimension
3:      $\boldsymbol{u} \sim g(\boldsymbol{u}|k')$         ▷ Dimension matching and deterministic function $h$
4:      $(\boldsymbol{\theta}', \boldsymbol{u}') = h(\boldsymbol{\theta}, \boldsymbol{u})$
5:      $\alpha = \min\left\{1, \frac{\pi(\boldsymbol{\theta}')g(\boldsymbol{u}')}{\pi(\boldsymbol{\theta})g(\boldsymbol{u})}\left|\frac{\delta(\boldsymbol{\theta}',\boldsymbol{u}')}{\delta(\boldsymbol{\theta},\boldsymbol{u})}\right|\right\}$
6:      $\phi \sim$ unif(0,1)
7:      if $\phi \leq \alpha$ then
8:          return $\boldsymbol{\theta}', k'$                   ▷ Accept the newly proposed state
9:      else
10:         return $\boldsymbol{\theta}, k$      ▷ Reject the proposed state, register the current state
11:     end if
12: end function
```

4.6 Hamiltonian Monte Carlo

The random-walk Metropolis-Hastings algorithm achieves good results for problems with few parameters, such as a simple linear regression or low-dimensional multivariate Gaussian. Extending MH to Gibbs sampling facilitates inference over hierarchical models and those with many parameters, albeit with a commensurate increase in computational time. Performance and consequently the acceptance probability will rapidly decrease as the dimension of the parameter space increases. This quickly becomes a significant problem for most practical applications of BNNs.

The "gold-standard" solution to performing MCMC inference on neural networks [67] is to use Hamiltonian Monte Carlo [68] (HMC), a gradient-charged algorithm motivated by concepts arising in statistical physics. The original paper on Hamiltonian Monte Carlo introduced the then-named "Hybrid Monte Carlo" [69] as a method for numerical simulation of lattice field theory.

From the physical perspective, HMC treats $\boldsymbol{\theta}$, the parameter vector of interest, as a d-dimensional position vector of an imaginary particle in parameter space. An auxiliary random vector of the same dimension $\boldsymbol{\rho}$ is drawn, most commonly from a multivariate normal distribution with a mean vector of zeroes and a covariance matrix $\boldsymbol{M}$ (most commonly taken to be the identity matrix $\boldsymbol{I}$), and referred to as the momentum vector. The position and momentum vector together define the Hamiltonian (eq 4.15) of an imaginary particle moving through the joint space of the parameter vector and its associated momentum.

$$H(\boldsymbol{\theta}, \boldsymbol{\rho}) = -\log(\boldsymbol{\theta}) + \frac{1}{2}\boldsymbol{\rho}^T \boldsymbol{M}^{-1} \boldsymbol{\rho} \tag{4.15}$$

The first term of the Hamiltonian equation represents the potential energy (log posterior) and the second term corresponds to the kinetic energy (half of the squared momentum). Hamilton's equations (eq 4.16, 4.17) corresponding to the movement of said particle tempt closed-form optimization solutions which would, statistically speaking, preserve the joint distribution of $(\boldsymbol{\theta}, \boldsymbol{\rho})$ and therefore the marginal distribution $\pi(\boldsymbol{\theta})$ is obtained as a by-product.

$$\frac{\boldsymbol{\theta}}{dt} = \frac{\delta H}{\delta \boldsymbol{\rho}} = \boldsymbol{M}^{-1}\boldsymbol{\rho} \tag{4.16}$$

$$\frac{\boldsymbol{\rho}}{dt} = -\frac{\delta H}{\delta \boldsymbol{\theta}} = \frac{\delta \log \pi}{\delta \boldsymbol{\theta}} \tag{4.17}$$

In practice, the equation is solved approximately, and detailed balance must therefore be preserved via a Metropolis-style correction. Updates to the Hamiltonian can be done through use of a *simplectic* integrator, which along with being reversible preserves volume, meaning no Jacobian correction is required. The simplectic integrator of choice for HMC is the *leapfrog* method, where L "leap-frog" steps of size ϵ iteratively update the two components of the Hamiltonian (algorithm 5).

Algorithm 5 Leapfrog [68]

```
 1: function LEAPFROG($\boldsymbol{\theta}$, $\boldsymbol{\rho}$, $\epsilon$, $L$)
 2:     $\boldsymbol{\rho} = \boldsymbol{\rho} - \frac{\epsilon \nabla \boldsymbol{\theta}'}{2}$        ▷ Half step for the momentum vector
 3:     for i=1:L do
 4:         $\boldsymbol{\theta} = \boldsymbol{\theta} + \epsilon \boldsymbol{\rho}$        ▷ Full step for the position vector
 5:         if i!=L then
 6:             $\boldsymbol{\rho} = \boldsymbol{\rho} - \epsilon \nabla \boldsymbol{\theta}$        ▷ Full step for the momentum
 7:         else
 8:             $\boldsymbol{\rho} = \boldsymbol{\rho} - \frac{\epsilon \nabla \boldsymbol{\theta}}{2}$        ▷ Half step for the momentum vector
 9:         end if
10:     end for
11:     return $\boldsymbol{\theta}, \boldsymbol{\rho}$
12: end function
```

This sojourn across the joint distribution of the model parameter vector and the randomly sampled momentum produces only small changes to the Hamiltonian. The end result of HMC (algorithm 6), provided ϵ and L are selected to specify a sufficiently long trajectory, is a new sample from $\pi(\boldsymbol{\theta})$ much further away in parameter space than one which might be proposed via a random-walk. The preservation of the Hamiltonian allows for a relatively consistent acceptance probability throughout the course of an HMC program.

A well-tuned HMC algorithm provides a powerful inference engine for sampling from a posterior distribution, but the tuning is not a trivial task. Hand-tuned HMC requires multiple validating runs as searches over the leapfrog step-size ϵ and the number of

Algorithm 6 Hamiltonian Monte Carlo [68]

```
1:  function HMC($\boldsymbol{\theta}, \epsilon, L$)
2:      $\boldsymbol{\rho} \sim MVN(\mathbf{0}, \boldsymbol{I}_d)$
3:      $\boldsymbol{\theta}', \boldsymbol{\rho}' = \text{Leapfrog}(\boldsymbol{\theta}, \boldsymbol{\rho}, \epsilon, L)$
4:      $\boldsymbol{\rho}' = -\boldsymbol{\rho}'$
5:      $K = \exp\left\{\frac{\boldsymbol{\rho}^T\boldsymbol{\rho}}{2}\right\}$
6:      $K' = \exp\left\{\frac{(\boldsymbol{\rho}')^T(\boldsymbol{\rho}')}{2}\right\}$
7:      $\alpha = \min\left\{1, \frac{\pi(\boldsymbol{\theta}')K}{\pi(\boldsymbol{\theta})K'}\right\}$
8:      $u \sim$ unif(0,1)
9:      if $u \leq \alpha$ then
10:         return $\boldsymbol{\theta}'$                ▷ Accept the newly proposed state
11:     else
12:         return $\boldsymbol{\theta}$                 ▷ Reject the proposed state, register the current state
13:     end if
14: end function
```

steps L. A method for automating the selection of these hyperparameters is introduced next.

4.7 The No-U-Turn Sampler

Sub-optimal HMC hyperparameter selection will typically have grave implications on the efficiency of the Markov Chain. If ϵ is too small, the algorithm will require a larger number of steps in order to avoid random-walk behaviour and consequently waste computation, while a large ϵ values will lead to low acceptance rates. L should be specified sufficiently large to guide the trajectory to a distinct proposal without doubling back to a neighbourhood of the current proposal, as too few or too many steps will not represent an improvement over a random-walk approach.

Optimal selection of these hyperparameters is usually highly dependent on the given model. This leads to particularly problematic implementations of HMC when the nature of a model may vary greatly across parameterizations, as different hyperparameter schedules may perform optimally in select regions of the parameter space and quite poorly in others. In high-dimensional problems, optimal specifications of both hyperparameters will change as the chain moves between modes of high probability density.

Select strategies do exist to augment ϵ automatically, but prior to 2014, no method existed for abstracting away specification of L by the user. Solutions to the automation of both the step-size and the number of steps for the leapfrog integrator in HMC are combined in the form of the No-U-Turn Sampler (NUTS) [70].

Selection of L

The forward idea present in the NUTS algorithm is to prevent the trajectory of the Hamiltonian from revisiting areas of the parameter space it has already visited, all while maintaining the condition of detailed balance for a new proposal in the chain. A criterion is established that identifies when this behaviour occurs and prevents the trajectory from proceeding further, eliminating so-called "U-Turns" and consequent unnecessary computation.

This criterion of interest is the derivative of half the squared distance between the proposal $\boldsymbol{\theta}'$ and the current state $\boldsymbol{\theta}$ (equation 4.18), which is proportional to progress made away from $\boldsymbol{\theta}$.

$$\frac{d}{dt}\frac{(\boldsymbol{\theta}' - \boldsymbol{\theta}) \cdot (\boldsymbol{\theta}' - \boldsymbol{\theta})}{2} \tag{4.18}$$

To satisfy detailed balance, reversibility is preserved through a recursive algorithm which runs the Hamiltonian simulation both forward and backwards in time, building up a set of potential proposals $C = \{\boldsymbol{\theta}'_1, \boldsymbol{\theta}'_2, \dots, \boldsymbol{\theta}'_c\}$ from which the resultant proposal $\boldsymbol{\theta}'$ is randomly drawn. The candidates are produced as leaf nodes of a balanced binary tree, which is iteratively produced by randomly going forward or backward 2^i leapfrog steps. The tree is built until the U-Turn criterion (equation 4.19) is met for the subtrajectory between the leftmost and rightmost nodes of any balanced subtree, which represents the point at which the largest trajectory begins to decrease in size.

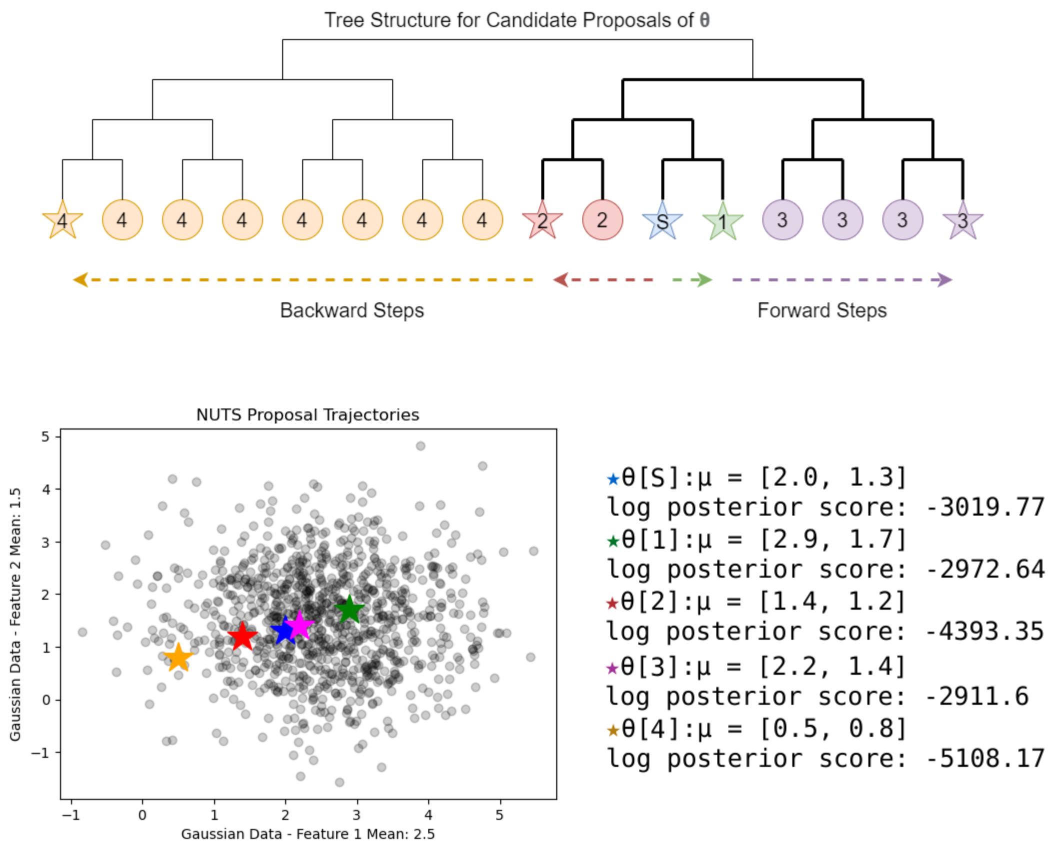
$$\mathcal{L}(\boldsymbol{\theta}) - \frac{1}{2}\boldsymbol{\rho} \cdot \boldsymbol{\rho} - \log u < -\Delta_{\max} \tag{4.19}$$

Here $\mathcal{L}(\boldsymbol{\theta})$ refers to the negative log-likelihood of the parameter vector, $\boldsymbol{\rho}$ is the randomly drawn momentum vector, u is a slice variable, and $\Delta_{\max}$ is some large value that prevents the algorithm from proceeding into regions of extremely low probability density.

Figure 4.1 displays the tree-building operation for taking leapfrog steps forward and backward in time.

Selection of ϵ

Iterative improvement of ϵ represents a stochastic convex optimization problem. The MCMC operation can be adapted to a dual averaging scheme as originally derived by Nesterov [71]. Each time a new NUTS trajectory is specified, the value of ϵ is updated through a number of burn-in steps m_b to effectively direct the trajectory as it moves between the transient burn-in stage to the stationary stage of the Markov chain. After m_b steps have been completed, HMC proposals continue with the most recently updated


Tree Structure for Candidate Proposals of θ
4
4
4
4
4
4
4
4
2
2
S
1
3
3
3
3
Backward Steps
Forward Steps
NUTS Proposal Trajectories
Gaussian Data - Feature 2 Mean: 1.5
Gaussian Data - Feature 1 Mean: 2.5
−1
0
1
2
3
4
5

```
★θ[S]:μ = [2.0, 1.3]
log posterior score: -3019.77
★θ[1]:μ = [2.9, 1.7]
log posterior score: -2972.64
★θ[2]:μ = [1.4, 1.2]
log posterior score: -4393.35
★θ[3]:μ = [2.2, 1.4]
log posterior score: -2911.6
★θ[4]:μ = [0.5, 0.8]
log posterior score: -5108.17
```

FIGURE 4.1: The No-U-Turn Sampler for Bivariate Gaussian Data

One iteration of NUTS is performed on the 2-dimensional mean vector for a bivariate Gaussian model. The parameter vector $\boldsymbol{\theta}$ corresponds to the mean $\boldsymbol{\mu} = [2.5, 1.5]$.

In figure (a), a binary tree is built by going forward and backward in time by 2^i steps at each iteration i. Each leaf node corresponds to a parameter vector reached by each of these leapfrog steps. Figure (b) displays the corresponding means for each terminal (left or rightmost for a given iteration) node of the tree. The stopping criterion identifies when a "u-turn" has been made in the gradient of the parameter vector and prevents further computation.

The proposal for $\boldsymbol{\theta}$ is then drawn from the balanced binary tree indicated by the bold section of the full tree. These represent the states that may be sampled from while preserving detailed balance. In this example, the "u-turn" occurs at step 3, stopping the simulation. The proposal state is then drawn at random from 1 of the 8 nodes of the highlighted section of the tree.

Algorithm 7 No U-Turn Sampler [70]

```
1:  function NUTS($\boldsymbol{\theta}^0$, $\epsilon$, $\mathcal{L}$)
2:      Sample momentum and slice variable
3:      $\boldsymbol{\rho} \sim N(\mathbf{0}, \boldsymbol{I})$
4:      $u \sim \text{Uniform}([0, \exp\{\mathcal{L}(\boldsymbol{\theta}^0) - \frac{1}{2}(\boldsymbol{\rho}^0 \cdot \boldsymbol{\rho}^0)\}])$
5:      Initialize $\boldsymbol{\theta}^-, \boldsymbol{\theta}^+ = \boldsymbol{\theta}^0; \boldsymbol{\rho}^-, \boldsymbol{\rho}^+ = \boldsymbol{\rho}^0; j = 0; \boldsymbol{C} = \{(\boldsymbol{\theta}^0; \boldsymbol{\rho}^0)\}; s = 1$
6:      while s = 1 do
7:          Choose a direction
8:          $v_j \sim \text{Uniform}(\{-1, 1\})$
9:          if $v_j = -1$ then
10:             $\boldsymbol{\theta}^-, \boldsymbol{\rho}^-, -, -, \boldsymbol{C}', s' \leftarrow$ BuildTree($\boldsymbol{\theta}^-, \boldsymbol{\rho}^-, u, v_j, j, \epsilon$)
11:         else
12:             $-, -, \boldsymbol{\theta}^+, \boldsymbol{\rho}^+, \boldsymbol{C}', s' \leftarrow$ BuildTree($\boldsymbol{\theta}^+, \boldsymbol{\rho}^+, u, v_j, j, \epsilon$)
13:         end if
14:         if $s' = 1$ then
15:             $\boldsymbol{C} \leftarrow \boldsymbol{C} \cup \boldsymbol{C'}$
16:         end if
17:         $s \leftarrow s'\mathbb{I}[(\boldsymbol{\theta}^+ - \boldsymbol{\theta}^-) \cdot \boldsymbol{\rho}^- \geq 0]\mathbb{I}[(\boldsymbol{\theta}^+ - \boldsymbol{\theta}^-) \cdot \boldsymbol{\rho}^+ \geq 0]$
18:         $j \leftarrow j + 1$
19:     end while
20:     Sample $\boldsymbol{\theta}$, $\boldsymbol{\rho}$ uniformly from $\boldsymbol{C}$
21:     return $\boldsymbol{\theta}$
22: end function

23: function BuildTree($\boldsymbol{\theta}, \boldsymbol{\rho}, u, v, j, \epsilon$)
24:     if $j = 0$ then
25:         Base case - one leapfrog step with direction v
26:         $\boldsymbol{\theta}', \boldsymbol{\rho}' \leftarrow$ Leapfrog($\boldsymbol{\theta}, \boldsymbol{\rho}, v\epsilon$)
27:         $\boldsymbol{C}' \leftarrow \begin{cases} \{(\boldsymbol{\theta}', \boldsymbol{\rho}')\}, & \text{if } u \leq \exp\{\mathcal{L}(\boldsymbol{\theta}') - \frac{1}{2}\boldsymbol{\rho}' \cdot \boldsymbol{\rho}'\} \\ 0, & \text{else} \end{cases}$
28:         $s' \leftarrow \mathbb{I}[\mathcal{L}(\boldsymbol{\theta}') - \frac{1}{2}\boldsymbol{\rho}' \cdot \boldsymbol{\rho}' > \log u - \Delta_{\max}]$
29:         return $\boldsymbol{\theta}', \boldsymbol{\rho}', \boldsymbol{\theta}', \boldsymbol{\rho}', \boldsymbol{C}', s'$
30:     else
31:         Recursion - build forward and backward subtrees
32:         $\boldsymbol{\theta}^-, \boldsymbol{\rho}^-, \boldsymbol{\theta}^+, \boldsymbol{\rho}^+, \boldsymbol{C}', s' \leftarrow$ BuildTree($\boldsymbol{\theta}, \boldsymbol{\rho}, u, v, j - 1, \epsilon$)
33:         if $v = -1$ then
34:             $\boldsymbol{\theta}^-, \boldsymbol{\rho}^-, -, -, \boldsymbol{C}'', s'' \leftarrow$ BuildTree($\boldsymbol{\theta}^-, \boldsymbol{\rho}^-, u, v, j - 1, \epsilon$)
35:         else
36:             $-, -, \boldsymbol{\theta}^+, \boldsymbol{\rho}^+, \boldsymbol{C}'', s'' \leftarrow$ BuildTree($\boldsymbol{\theta}^+, \boldsymbol{\rho}^+, u, v, j - 1, \epsilon$)
37:         end if
38:         $s \leftarrow s's''\mathbb{I}[(\boldsymbol{\theta}^+ - \boldsymbol{\theta}^-) \cdot \boldsymbol{\rho}^- \geq 0]\mathbb{I}[(\boldsymbol{\theta}^+ - \boldsymbol{\theta}^-) \cdot \boldsymbol{\rho}^+ \geq 0]$
39:         $\boldsymbol{C}' \leftarrow \boldsymbol{C}' \cup \boldsymbol{C}''$
40:         return $\boldsymbol{\theta}^-, \boldsymbol{\rho}^-, \boldsymbol{\theta}^+, \boldsymbol{\rho}^+, \boldsymbol{C}', s'$
41:     end if
42: end function
```

value for ϵ, while in the case of NUTS, L continues to be selected based on the tree-building method for proposals as above.

This strategy for automating ϵ allows for specification of a desired acceptance probability $\hat{\alpha}$ for each HMC step. It has been shown that an optimal choice for $\hat{\alpha}$ in HMC is 0.65 [72], which is used for all experiments in this thesis.

NUTS in a Nutshell

The HMC hyperparameters ϵ and L are automatically adapted based on a target proposal acceptance probability and the current state of the Markov chain dependant on the individual inference problem definition. This automation simplifies the implementation of within-dimension moves as part of the trans-dimensional inference task for moving between BNN architectures, efficiently abstracting specification of two of the "moving parts" of the algorithm away from the user.

Algorithm 7 provides an intuitive solution to the selection of L, but is considered by the authors to be "naive". A more efficient implementation with the dual-averaging approach to the selection of ϵ is proposed as Algorithm 6 in the original paper [70], and is the version used in the composite sampling algorithm presented in section 5.4.2 of this thesis.

Theoretical proofs of the validity of NUTS are presented in the original paper. We also refer to the forward usage of NUTS in the popular STAN probabilistic programming package [73] as justification for favouring NUTS as the solution to fast within-dimension MCMC updates. Applying NUTS to BNN inference is advocated for in a recent review of BNN techniques [50].

4.8 Inference Method Summary

Trans-dimensional inference via RJMCMC will allow for sampling from the exact joint posterior distribution over parameterizations and architectures for a BNN. In order to promote efficient sampling and therefore reduce the computation time required for convergence of the Markov Chain, the No U-Turn Sampler as an extension of HMC will be used to increase the acceptance rate of within-dimension and across-dimension proposals. The following chapter examines the implementation in detail, including the composite sampling program and a treatment of the challenging specification of across-dimension proposals for moving between architectures.

Chapter 5

The Reversible Jump Bayesian Neural Network

All of the preliminary components necessary to our approach for performing trans-dimensional inference on BNNs have been introduced in chapters 2 - 4. We now present the model in full detail, beginning with descriptions of the components of Bayesian inference - the posterior distribution and the prior selections.

Proposal specifications for RJMCMC applied to BNNs are then discussed, followed by details for successful implementation. This chapter closes with a demonstration of trans-dimensional inference applied to a BNN for a classification task of 2-dimensional "XOR" data clouds.

5.1 Model Description

The goal is to generate a joint distribution of neural network architectures and parameterizations which address the learning task at hand. A standard BNN represents a distribution of parameterizations given a fixed architecture specification, which can be produced by one of the various inference techniques introduced in Chapter 3. In order to extend such a model to represent neural networks of various size, trans-dimensional inference by RJMCMC will be used.

The forward product of interest of trans-dimensional inference will be a joint posterior distribution over the parameter vector and the model indicator as in equation 5.1.

$$p(\boldsymbol{\theta}, \mathcal{M}|\boldsymbol{D}) = \frac{\mathcal{L}(\boldsymbol{\theta}, \mathcal{M}|\boldsymbol{D})p(\boldsymbol{\theta}|\mathcal{M})p(\mathcal{M})}{p(\boldsymbol{D})} \tag{5.1}$$

Generally speaking, the model indicator $\mathcal{M}$ will be used to represent the structure of the model. It may correspond directly to the number of parameters in the model via some function $h(i)$ or may map to a series of m pre-specified models $\mathcal{M}_i \in \{\mathcal{M}_1, \mathcal{M}_2, ...\mathcal{M}_m\}$.

The functional representation f of the parameters for $\mathcal{M}$ is an ANN model as defined in chapter 2. In our experiments, $\mathcal{M}$ is sufficiently represented by two architecture indicator variables. The dimension of the parameter vector $\boldsymbol{\theta}$ is dependent on $\mathcal{L}$, the number of hidden layers, and $\mathcal{K}$, the number of nodes in each hidden layer. Such a model will be referred to in this work as a *Reversible Jump Bayesian neural network* (RJBNN).

A RJBNN model is thus represented by $\mathcal{M}\{f, (\boldsymbol{\theta}, \boldsymbol{\tau}, \tau_y, \mathcal{L}, \mathcal{K}), \boldsymbol{x}, \boldsymbol{y}\}$ where $\boldsymbol{\theta}$ is the vector of network weights and biases, $\boldsymbol{\tau}$ represents the precision hyperparameters for prior distributions on the weights and biases, τ_y is the precision term for the Gaussian likelihood[1], and the network architecture is structured by $\mathcal{L}$ and $\mathcal{K}$. Feature data is represented by $\boldsymbol{x}$, and the response labels/targets are indicated by $\boldsymbol{y}$.

5.1.1 Prior Selections

A general approach will be followed for each RJBNN implementation. Broadly speaking, the network will contain up to four distinct categories of parameters:

1. $\{\mathcal{L}, \mathcal{K}\}$ - architecture parameters which respectively represent the number of hidden layers in the network and the number of nodes in each layer[2]
2. $\boldsymbol{\theta} = \{\boldsymbol{W}_i, \boldsymbol{b}_i\}_{i=1}^{\mathcal{L}+1}$ - the network weights and biases[3]
3. $\boldsymbol{\tau} = \{\tau_i, \tau_{bi}\}_{i=1}^{\mathcal{L}+1}$ - the common precision terms for the input weights and input biases, shared between all nodes of one given layer. The precision terms correspond to the inverse of the variance $\sigma^2 = \tau^{-1}$ hyperparameters for Gaussian priors over the network parameters.
4. τ_y - the precision for the noise parameter $\sigma_y^2 = \tau_y^{-1}$ in the likelihood for a regression network (absent in classification networks)

Architecture Parameters

We are in relatively uncharted territory when it comes to prior knowledge about the size of our networks. In such a case, a vague prior which imposes minimal restriction on the available network capacity may be in order. The distribution needs to produce discrete values, and perhaps allow for infinite support. One could select a geometric, negative binomial, or Poisson distribution, each appropriately modified to not include 0 in the support (the network is assumed to have at least one hidden layer with at least one hidden node). Any of these options could be specified to favour some region or mode

[1]For the regression network only

[2]For practicality, in this work, only networks with the same number of nodes across all hidden layers are considered.

[3]Until now, we have used $\boldsymbol{\theta}$ to represent *all* parameters of inferential interest in a model. We now draw a distinction for our hierarchical model (see section 3.3) between so-called low-level parameters ($\boldsymbol{\theta}$) and hyperparameters ($\boldsymbol{\tau}$)

for the number of components, but a typical approach might be to favour the smallest network possible, promoting the production of the most efficient network model per Occam's razor.

However, in order to promote computational feasibility, it may instead be preferable to declare a finite support on structure parameters so that $\mathcal{L}$ and $\mathcal{K}$ are drawn from pre-determined sets of possible values. Such an approach stands to vastly increase the chances of an RJMCMC chain converging while still providing insight into likely optimal network models. It is also noted that we need not impose any specific preference to certain values over other. The Bayesian approach implicitly induces Occam's razor [41], and the trans-dimensional Markov chain is expected to prefer those models which manage to fit the data using the minimal resources (parameters) required.

With this motivation in mind, experiments are run with pre-defined supports $\boldsymbol{L} = \{1, 2, \ldots \mathcal{L}_{\max}\}$ or $\boldsymbol{K} = \{1, 2, \ldots \mathcal{K}_{\max}\}$, and $\mathcal{L}$, $\mathcal{K}$ drawn from categorical distributions with uniform probabilities $\frac{1}{|\boldsymbol{L}|}$ and $\frac{1}{|\boldsymbol{K}|}$ for each candidate support value.

Network Parameters

The default approach to specifying priors for the weights and biases in a BNN is to use zero-centered Gaussians [41]. Such a prior places no preference on positive vs negative values for the parameters associated with the linear transformation of the previous layer's features at each node, and emphasizes a preference for smaller absolute values for each parameter, akin to L2-regularization.

Precision Hyperparameters

The standard deviation for the network parameter Gaussian prior distributions are assumed to be hyperparameters, each expressed as the corresponding precision term, and drawn from a Gamma distribution. The use of a Gamma distribution here is motivated by a convenient conjugate prior result, allowing for quick and efficient Gibbs sampling updates to the hyperparameters. There is some flexibility in terms of which network parameters share a common precision term. We refer to this specification as the granularity of the hyperparameter schedule. The minimal granularity would be one shared precision term for all weights and biases of the network, and a maximum granularity would be one precision term for every individual network parameter. Early work showed that use of the minimal granularity in regression networks results in a lower correlation between the log-posterior score and the error term (root mean square error, RMSE) [45].

Layer granularity is recommended for its demonstrated superiority on a similar BNN regression task [47]. With this scheme, an individual precision term is shared between all weights in one network layer, and likewise one precision term is shared between all biases for that same layer. Given a network with ℓ hidden layers, the total number of network precision hyperparameters will be $2 * (\ell + 1)$, for the hidden layers and the output layer.

We present the theory behind RJBNNs with this approach in mind. This being said, preliminary runs with layer granularity hyperparameters presented computational challenges to the MCMC inference algorithms. For this reason, only fixed hyperparameter solutions are used in this work as a preliminary investigation. Further discussion of the use of a more detailed hyperparameter schedule are discussed as a future research direction in section 7.3.3.

Regression Precision

The precision term for the regression problem corresponds to the variance in the Gaussian "noise" distribution assumed to represent the likelihood of network outputs, $\hat{\boldsymbol{y}}$. It is similarly drawn from a Gamma distribution, $\tau_y \sim \text{Gamma}(\alpha_y, \beta_y)$, with $\alpha_y = \beta_y = 1$.

5.2 Across-Dimension Proposals for Neural Networks

Successfully running RJMCMC inference on a BNN can be implemented correctly quite easily. Proposals can be entirely based upon random, uncorrelated draws from the prior distributions over each parameter. This is achieved by first drawing the structure indicator parameters $\mathcal{L} \sim p(\mathcal{L})$, $\mathcal{K} \sim p(\mathcal{K})$, followed by the appropriately sized parameter vector $\theta \sim p(\theta|\mathcal{L}, \mathcal{K})$. No specific model tuning is necessary to achieve the guaranteed asymptotic convergence behaviour, as the Markov condition and detailed balance requirements are inherently satisfied. We do not, however, arrive at any guarantee or likelihood that the Markov chain will converge to its stationary distribution in any computationally tractable amount of time. The chance for this to occur as our model grows in dimension with the size of the network becomes almost negligible very quickly.

This is ultimately the problem that the reversible jump framework for the MHG algorithm attempts to alleviate. A well-designed proposal that deterministically jumps between model architectures while meeting the criteria for detailed balance should allow for the parameter vector of the new dimension to land in a region of relatively high density space to have a reasonable chance of being accepted. The task is then to declare a proposal mechanism which can take the current neural network structure indicated by $\mathcal{L}, \mathcal{K}$ and propose a network $\mathcal{L}', \mathcal{K}'$ such that the new network model will achieve a reasonable posterior score and consequent desirable acceptance rate.

5.2.1 The Base Case: Random Draws from the Prior

Perhaps trivially, a within-dimension MCMC proposal may be generated such that $\boldsymbol{\theta}' \sim p(\boldsymbol{\theta})$, the model prior as specified near the top of the Bayesian workflow. Extending this concept to across-dimensional proposals via the RJMCMC framework, the hierarchical model would dictate a sampling schedule where the node parameter is first drawn as $\mathcal{K}' \sim p(\mathcal{K})$, followed by generation of the model parameter proposal $\boldsymbol{\theta}' \sim p(\boldsymbol{\theta}|\mathcal{K}')$. A similar argument would follow for the case of the layer parameter $\mathcal{L}$.

This method of drawing from the prior is meant to motivate a minimal definition for an across-dimension proposal, but is most likely insufficient for achieving a reasonable RJMCMC acceptance probability. We now instead define proposals that utilize the existing structure and parameterization of the network based on the current state of the Markov chain as a weakly-informed proposal.

5.2.2 Reversible Jump Proposals for Neural Network Structures

As a motivating question, what can we say about a proposing a well-specified neural network model $\mathcal{M}_2(f_2, \boldsymbol{\theta'}|\mathcal{K}')$ given a well-specified model $\mathcal{M}_1(f_1, \boldsymbol{\theta}|\mathcal{K})$ when $\mathcal{K}' \neq \mathcal{K}$? To say that $\mathcal{M}_1$ is well-specified means that the associated network paramaterization returns a desirable classification accuracy or regression score on a test data set. To frame the question another way, if neurons are to be added or removed from a given layer of the capable network, what values should the new weight and bias parameters take for the additional neurons and their dependencies in the following layers, and how should the existing neuron parameters be updated? The same questions apply to the case when we adjust the number of layers - given $\mathcal{M}_1(f_1, \boldsymbol{\theta}|\mathcal{L})$, how do we parameterize $\mathcal{M}_2(f_2, \boldsymbol{\theta'}|\mathcal{L}')$ when $\mathcal{L}' \neq \mathcal{L}$?

Recent research into *network morphisms* [74] proposes an algorithmic approach to *growing* a network to $\mathcal{M}_2$ in such a way that the functional representation of $\mathcal{M}_1$ is preserved, i.e. $f_1 \equiv f_2$. This approach unfortunately does not lend itself to RJMCMC, as the algorithm is not calibrated with respect to detailed balance, and no method is defined for shrinking the network. We are not currently aware of any similar technique that satisfies detailed balance.

Absent an exact preservation of the NN function between models of different sizes, we still desire a proposal which exists in a tenable neighbourhood of the new parameter space $\boldsymbol{\Theta}_2$. If such a proposal can be defined, we may then be able to use delayed rejection sampling to move closer to a mode of the distribution in the new dimension as discussed in section 5.3.

For the experiments in this thesis, two proposals for each case of adjusting the number of neurons $\mathcal{K}$ and the number of layers $\mathcal{L}$ are in focus. The proposals are designed to minimally impact the parameterization of the network corresponding to the current state of the Markov chain, in the hopes that some useful functional representation is preserved when jumping to a new network architecture. With this motivation for minimal impact in mind, jump proposals are defined such that only one neuron or layer is added or removed at a time. In reversible jump parlance, such proposals are known respectively as *birth* and *death* moves [75].

Neuron Birth Proposal

Assume a neural network with one hidden layer with $\mathcal{K}$ neurons. Given the current state $\boldsymbol{\theta}$, a proposal is generated as $\mathcal{K}' = \mathcal{K} + 1$, with $\boldsymbol{\theta}' \sim p(\boldsymbol{\theta}|\mathcal{K}')$. The new neuron can be added anywhere in the layer, such that the insertion place i is randomly drawn from a discrete uniform, $i \sim u(1, \mathcal{K}')$.

Three sets of new parameters must be drawn. A new weight vector $\boldsymbol{W}_\ell^{(i)}$ and bias $b_\ell^{(i)}$ are proposed for the added neuron. New weights are also drawn for the output layer, $\boldsymbol{W}_{\ell+1}$ - one new weight for each of the neurons in the output layer. Every other network parameter is carried forward from the current state $\boldsymbol{\theta}$.

Each of these new parameters are drawn from a zero-centered Gaussian prior distribution dependent on their respective variance parameters, $\sigma^2 = \frac{1}{\tau}$, with τ the associated precision parameter drawn from the Gamma hyperprior.

These new parameters together correspond to the random number draw for vector $\boldsymbol{u}$ in the RJMCMC framework. The adjustment to the acceptance ratio $q(\boldsymbol{u})$ is therefore the joint density of the new parameters, which due to independence of the parameters is simply the product of the prior densities for each drawn parameter.

The birth of a neuron is illustrated in figure 5.1.

Neuron Death Proposal

Assume a neural network with one hidden layer with $\mathcal{K}$ neurons. Given the current state $\boldsymbol{\theta}$, a proposal is generated as $\mathcal{K}' = \mathcal{K} - 1$, with $\boldsymbol{\theta}' \sim p(\boldsymbol{\theta}|\mathcal{K}')$. The deleted neuron can be selected as any neuron in the layer, such that the deletion place i is randomly drawn from a discrete uniform, $i \sim u(1, \mathcal{K})$.

Three sets of parameters are removed. The weight vector for the neuron $\boldsymbol{W}_\ell^{(i)}$ and bias $b_\ell^{(i)}$ are deleted. Weights are also removed from the output layer, $\boldsymbol{W}_{\ell+1}$ - one weight removed for each of the neurons in the output layer. Every other network parameter is carried forward from the current state $\boldsymbol{\theta}$. Each of these parameters are assessed from the density of a zero-centered Gaussian prior dependent on their respective variance parameters, $\sigma^2 = \frac{1}{\tau}$, with τ the associated precision parameter drawn from the Gamma hyperprior.

These deleted parameters together correspond to the random number draw for vector $\boldsymbol{u}$ in the *reverse step* of the RJMCMC framework. The adjustment to the acceptance ratio $q(\boldsymbol{u})$ is therefore the *inverse* of the joint density of the removed parameters, which due to independence of the parameters is simply the product of the prior densities for each drawn parameter.

The death of a neuron is illustrated in figure 5.2.

Layer Birth Proposal

Assume a neural network with $\mathcal{L}$ hidden layers, and for each hidden layer a similar number of neurons $\mathcal{K}$. Given the current state $\boldsymbol{\theta}$, a proposal is generated as $\mathcal{L}' = \mathcal{L} + 1$, with $\boldsymbol{\theta}' \sim p(\boldsymbol{\theta}|\mathcal{L}')$. The new layer is added as the final hidden layer of the network, such that the insertion place $i = \mathcal{L}'$. Two sets of new parameters must be drawn. A new weight matrix $\boldsymbol{W}_{\mathcal{L}'}$ and bias vector $b_{\mathcal{L}'}$ are proposed for the added layer. Every other network parameter is carried forward from the current state $\boldsymbol{\theta}$.

Each of these new parameters are drawn from a zero-centered Gaussian prior distribution dependent on their respective variance parameters, $\sigma^2 = \frac{1}{\tau}$, with τ the associated precision parameter drawn from the Gamma hyperprior.

These new parameters together correspond to the random number draw for vector $\boldsymbol{u}$ in the RJMCMC framework. The adjustment to the acceptance ratio $q(\boldsymbol{u})$ is therefore the joint density of the new parameters, which due to independence of the parameters is simply the product of the prior densities for each drawn parameter.

The birth of a layer is illustrated in figure 5.3.

Layer Death Proposal

Assume a neural network with $\mathcal{L}$ hidden layers, and for each hidden layer a similar number of neurons $\mathcal{K}$. Given the current state $\boldsymbol{\theta}$, a proposal is generated as $\mathcal{L}' = \mathcal{L} - 1$, with $\boldsymbol{\theta}' \sim p(\boldsymbol{\theta}|\mathcal{L}')$. The final hidden layer of the network is deleted, such that the deletion place $i = \mathcal{L}$.

Two sets of parameters must be removed. The weight matrix $\boldsymbol{W}_\ell$ and bias vector b_ℓ are deleted for the removed layer. Every other network parameter is carried forward from the current state $\boldsymbol{\theta}$. Each of these parameters are assessed from the density of a zero-centered Gaussian prior dependent on their respective variance parameters, $\sigma^2 = \frac{1}{\tau}$, with τ the associated precision parameter drawn from the Gamma hyperprior.

These deleted parameters together correspond to the random number draw for vector $\boldsymbol{u}$ in the *reverse step* of the RJMCMC framework. The adjustment to the acceptance ratio $q(\boldsymbol{u})$ is therefore the *inverse* of the joint density of the removed parameters, which due to independence of the parameters is simply the product of the prior densities for each drawn parameter.

The death of a layer is illustrated in figure 5.4.

5.3 Improving Across-Dimension Proposals

The proposal mechanism described in section 5.2.2 alone may not be sufficient to increase the acceptance probability of a new network architecture. The estimated targets (or class labels) $\hat{\boldsymbol{y}}$ have not been calibrated based on the new architecture, contributing through the likelihood to an overall low posterior score for the new model proposal. Once the chain is in a state that corresponds to a relatively "trained" network parameterization, acceptance is unlikely for a network that does not fit the data well in comparison.

Several tricks to better realize the desired convergence behaviour in a practical amount of time appear throughout the literature regarding RJMCMC, HMC, and BNNs. These techniques are explored in this section, with implementation details highlighted in the final section of this chapter.

5.3.1 Multiple Proposals

When designing a custom MCMC algorithm, the practitioner is not limited to a single proposal mechanism. Multiple proposal definitions, up to and including an infinite variety, can be implemented and selected from to prompt better convergence behaviour of the chain for the given inference task. Proposals may target all parameters and hyperparameters simultaneously, or iterate through subsets of the model's parameter space.

Composite proposals extend the MCMC algorithm with an additional step wherein a proposal mechanism is randomly selected according to some probability distribution that adheres to the Markov property of the chain. Given the current state, one of the defined proposal mechanisms is selected before proceeding to the generation of proposed parameter values and the subsequent steps of Metropolis-Hastings-Green algorithm. So long as the probability $p(q_j)$ of selecting a proposal of type j is independent of the state of the Markov chain i, detailed balance is maintained for a sampling program with multiple proposals which each individually preserve detailed balance.

In RJMCMC, composite proposals can be used to switch between instances of across-dimension and within-dimension proposals. The former here refers to a proposal which

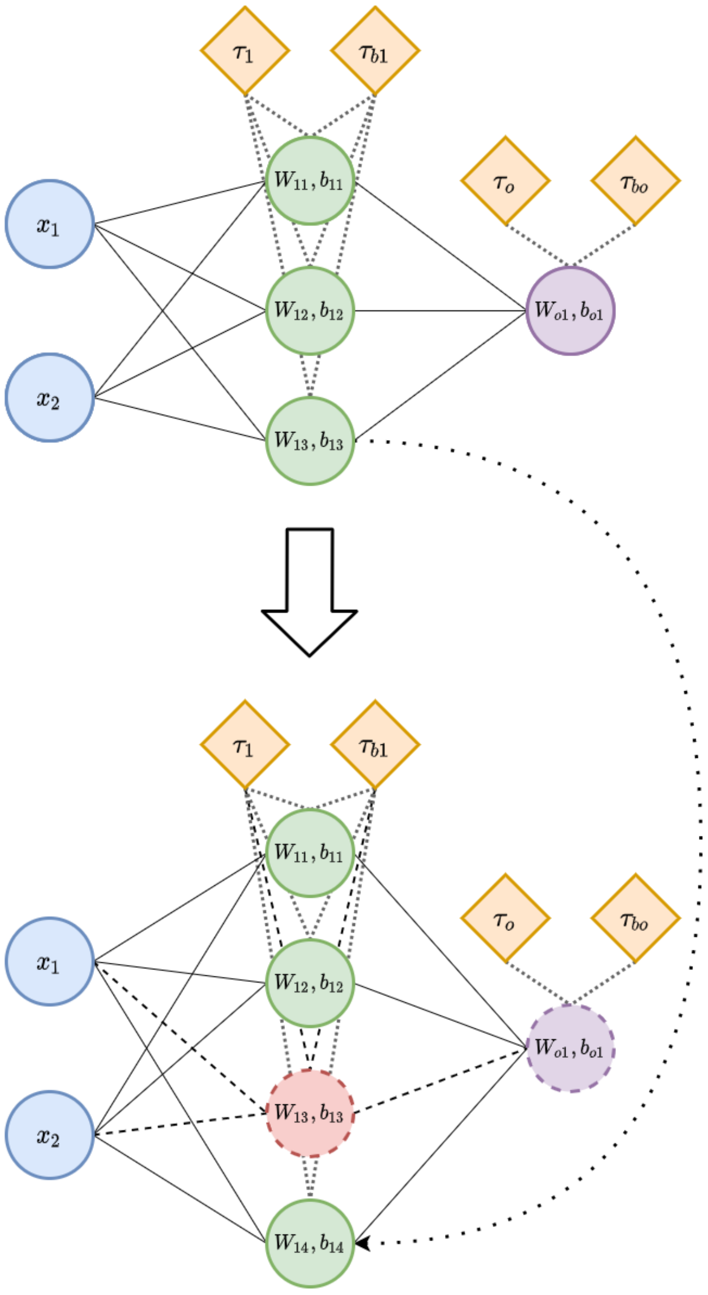


FIGURE 5.1: Neuron birth RJMCMC proposal

The current state of the Markov chain corresponds to a network with $\mathcal{K} = 3$. A proposal is generated such that $\mathcal{K}' = 4$. A neuron is inserted randomly at position $i \sim \text{uniform}[1,4]$; in this case $i = 3$. Notice that network parameters for neuron 4 are copied from neuron 3 in the current network. New network parameters are drawn from the prior distribution for (the newly inserted) neuron at position 3. Also note that an additional component is drawn for the output weight vector representing its connection to the new neuron.

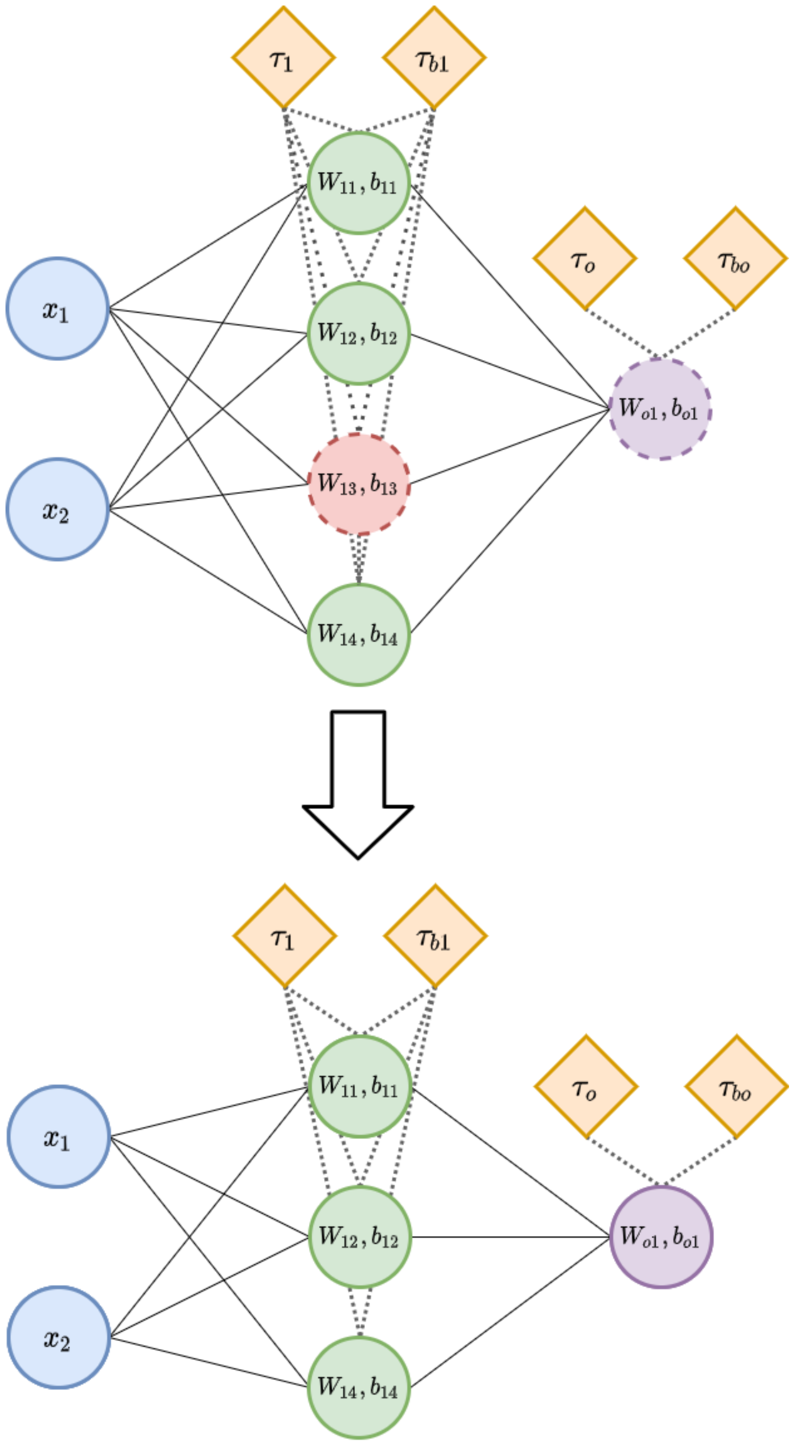


FIGURE 5.2: Neuron death RJMCMC proposal

The current state of the Markov chain corresponds to a network with $\mathcal{K} = 4$. A proposal is generated such that $\mathcal{K}' = 3$. The neuron at position 3 is removed.

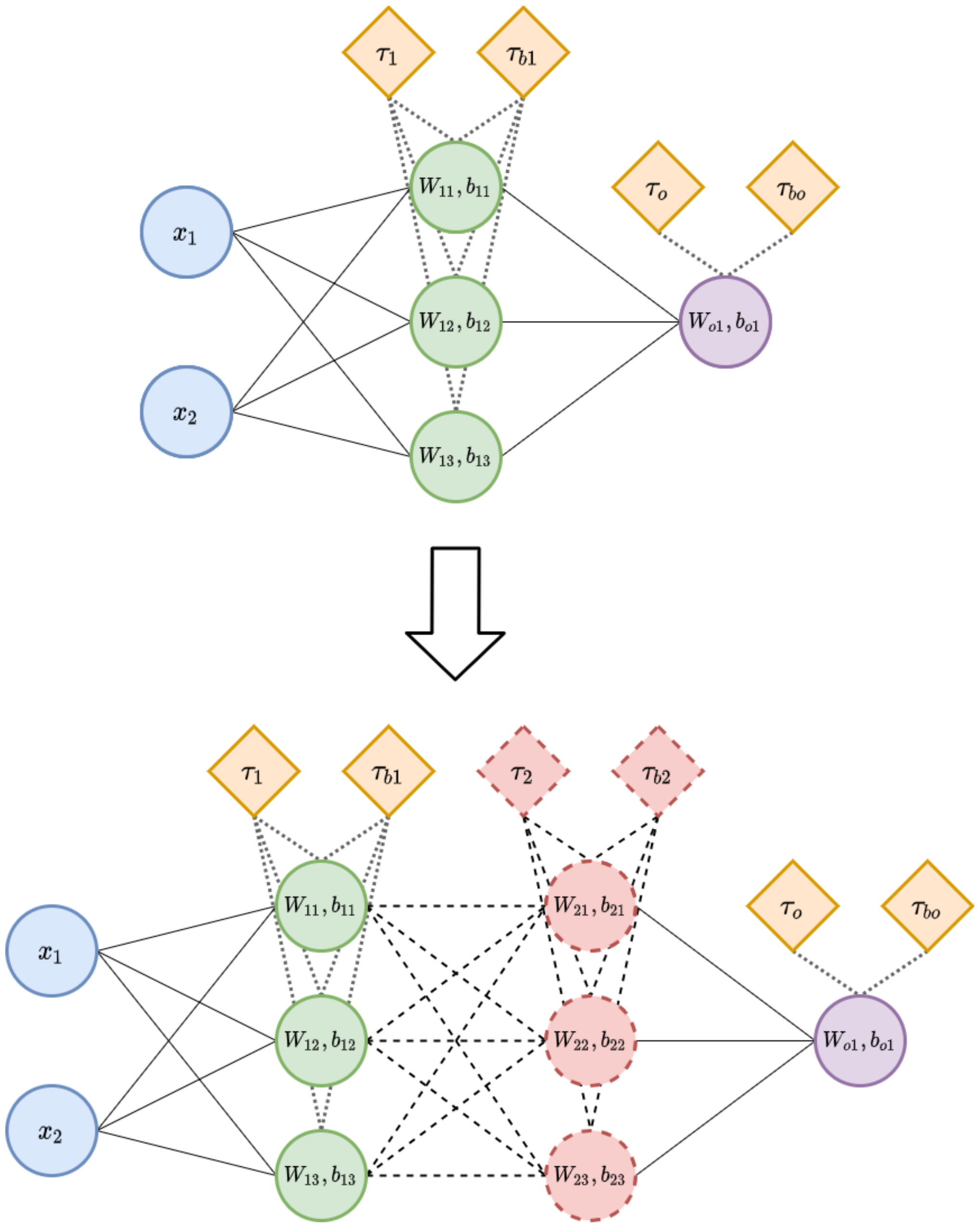


FIGURE 5.3: Layer birth RJMCMC proposal

The current state of the Markov chain corresponds to a network with $\mathcal{L} = 1$. A proposal is generated such that $\mathcal{L}' = 2$. A layer is inserted at position $\mathcal{L}' = 2$, and new network parameters and hyperparameters are drawn accordingly.

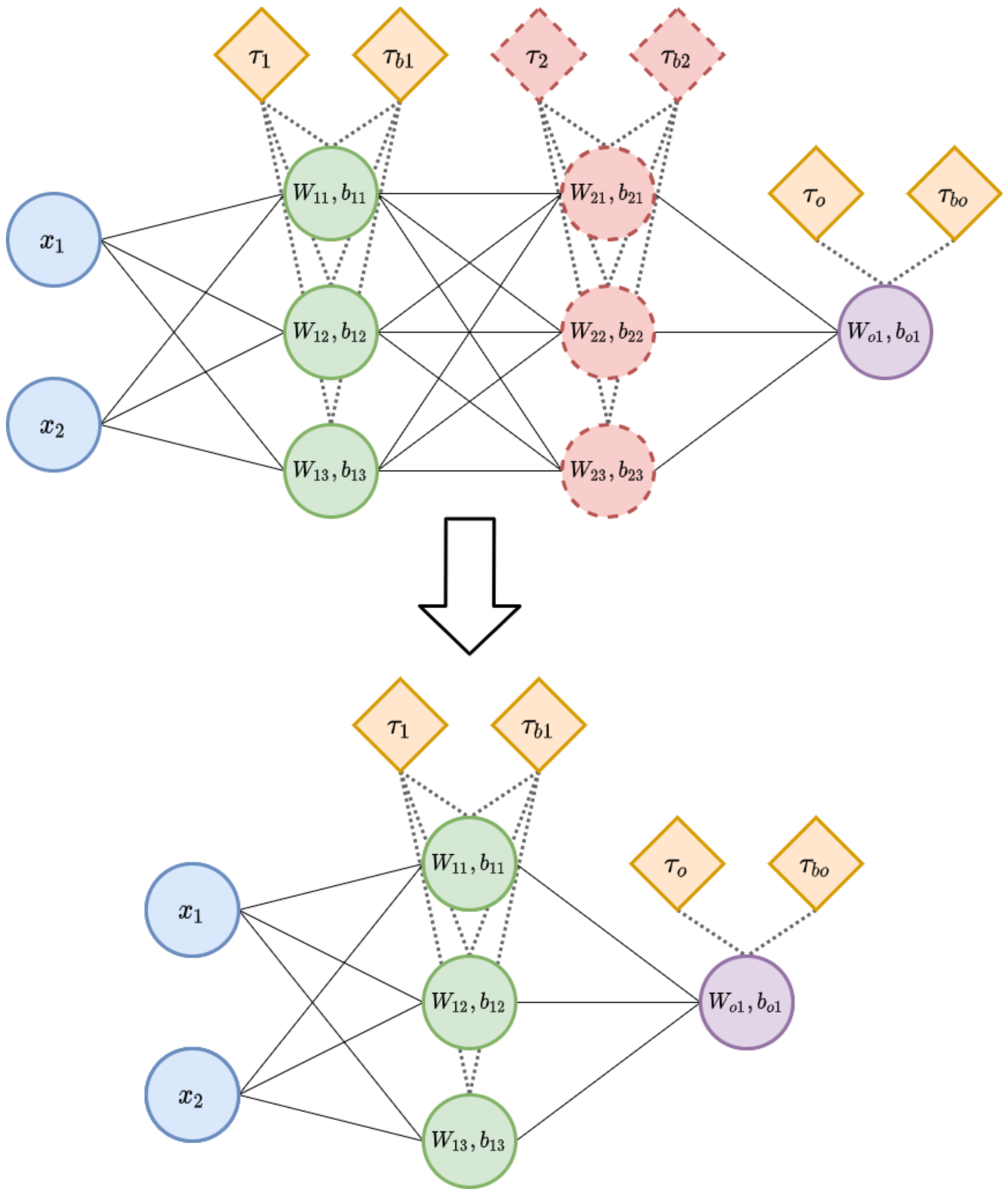


FIGURE 5.4: Layer death RJMCMC proposal

The current state of the Markov chain corresponds to a network with $\mathcal{L} = 2$. A proposal is generated such that $\mathcal{L}' = 1$. A layer is removed at position $\mathcal{L} = 2$.

alters the size of the parameter vector to be sampled (trans-dimensional inference) while the latter takes the dimension as fixed, based on the current state of the Markov chain. Distinguishing within-model moves and thus inducing temporary fixed dimensionality allows for the inclusion of a gradient-based HMC or NUTS proposal.

5.3.2 Delayed Rejection for Across-Dimension Proposals

Even with a well-designed across-dimension move proposal, high-dimensional parameter spaces can be daunting when it comes to achieving a reasonable acceptance rate. The geometry of the probability density as dimensionality increases becomes exponentially narrower, as regions of low probability density expand exponentially faster than those with high probability density. It becomes dramatically more likely that a Markov chain entering a new parameter space will find a parameterization with a low log-posterior score and thus be rejected.

An alternative proposal mechanism for improving the acceptance rate of reversible jump proposals [76] is to augment the across-dimension step with additional within-dimension steps in the parameter space of the new dimension before the proposal is assessed for acceptance. So long as detailed balance is maintained, a valid proposal can be formulated such that the Markov chain has had an opportunity to move closer to a mode of the probability density of the parameter vector in the new space. This can dramatically improve the acceptance probability for across-dimension moves.

The proposed mechanism requries an adjustment to the acceptance probability formulation. Let $\boldsymbol{\theta}$ represent the m-dimensional parameter vector for a network model $\mathcal{M}_1 : \{\boldsymbol{\theta}, \boldsymbol{k}, l\}$, and we wish to propose an across-dimension jump to $\boldsymbol{\theta}'$ for an n-dimensional network model $\mathcal{M}_2\{\boldsymbol{\theta}', \boldsymbol{k}', l'\}$, $n > m$. As discussed, without a strong jump proposal mechanism, the acceptance probability α_{mn} will likely be untenable. We thus introduce a new chain to propose a new n-dimensional state $\boldsymbol{\theta}^*$ using a j fixed-dimensional MCMC updates of our choosing, taking care to maintain detailed balance throughout each individual step from equation 4.6:

$$\pi(\boldsymbol{\theta}^*)p(\boldsymbol{\theta}', \boldsymbol{\theta}^*) = \pi(\boldsymbol{\theta}')p(\boldsymbol{\theta}^*, \boldsymbol{\theta}') \tag{5.2}$$

The detailed balance requirement for the entire proposal from $\boldsymbol{\theta} \in \mathbb{R}^m$ to $\boldsymbol{\theta}^* \in \mathbb{R}^n$ is now:

$$\int_{\boldsymbol{\theta},\boldsymbol{\theta}',\boldsymbol{\theta}^*} \pi_m(\boldsymbol{\theta})q_{mn}(\boldsymbol{\theta}, \boldsymbol{\theta}')p(\boldsymbol{\theta}', \boldsymbol{\theta}^*)\alpha_{mn}(\boldsymbol{\theta}, \boldsymbol{\theta}^*)d\boldsymbol{\theta}d\boldsymbol{\theta}'d\boldsymbol{\theta}^*$$
$$= \int_{\boldsymbol{\theta}^*,\boldsymbol{\theta}',\boldsymbol{\theta}^*} \pi_n(\boldsymbol{\theta}^*)p(\boldsymbol{\theta}^*, \boldsymbol{\theta}')q_{nm}(\boldsymbol{\theta}', \boldsymbol{\theta})\alpha_{nm}(\boldsymbol{\theta}^*, \boldsymbol{\theta})d\boldsymbol{\theta}d\boldsymbol{\theta}'d\boldsymbol{\theta}^* \tag{5.3}$$

The acceptance probability for such a move will be a modification of the Metropolis-Hastings acceptance probability for an across-dimension jump from equation (RJ acceptance):

$$\alpha_{mn}(\boldsymbol{\theta}, \boldsymbol{\theta}^*) = \min\left\{1, \frac{\pi_n(\boldsymbol{\theta}^*)p(\boldsymbol{\theta}^*, \boldsymbol{\theta}')q_{nm}(\boldsymbol{\theta}', \boldsymbol{\theta})}{\pi_m(\boldsymbol{\theta})q_{mn}(\boldsymbol{\theta}, \boldsymbol{\theta}')p(\boldsymbol{\theta}', \boldsymbol{\theta}^*)}\right\} \tag{5.4}$$

and the reverse move an extension of equation (RJ acceptance reverse):

$$\alpha_{nm}(\boldsymbol{\theta}^*, \boldsymbol{\theta}) = \min\left\{1, \frac{\pi_m(\boldsymbol{\theta})q_{mn}(\boldsymbol{\theta}, \boldsymbol{\theta}')p(\boldsymbol{\theta}', \boldsymbol{\theta}^*)}{\pi_n(\boldsymbol{\theta}^*)p(\boldsymbol{\theta}^*, \boldsymbol{\theta}')q_{nm}(\boldsymbol{\theta}', \boldsymbol{\theta})}\right\} \tag{5.5}$$

5.3.3 Palindromic Proposal Compositions

We have defined a delayed-rejection proposal that extends an across-dimension move with a fixed-dimension move in an attempt to move the intermediate proposal $\boldsymbol{\theta}'$ to an area of higher posterior density, but there is a problem. To maintain reversibility, our reverse move for which the dimension of the parameter vector decreases must be asymmetrical with respect to the original move. This means that the fixed-dimension move takes place *before* the across-dimension, and therefore a decrease in dimension will not benefit from the j improvement steps. Fortunately, the composite proposal can be generalized further.

Palindromic proposal compositions are compositions of v ordered proposals $p1, p2, ...p_v$ such that $p_i = p_{v-i+1}$ for all $i = 1, 2, ...v$. The simplest example would be for $v = 3$, where p_1 and p_3 follow the same proposal scheme, while p_2 as the central proposal can be totally unique. In this way, we can "sandwich" an across-dimension move with fixed-dimension moves on either side, therefore allowing both birth and death moves to benefit from the j fixed-dimensional steps. The first fixed-dimensional proposal will generate a new intermediate state $\boldsymbol{\theta}^{\sim} \in \mathbb{R}^m$ drawn from $p(\boldsymbol{\theta}^{\sim}, \boldsymbol{\theta})$.

Proposals of this nature are the focus of experiments in this work, the details of which are now derived. The detailed balance will be:

$$\begin{aligned}\int_{\boldsymbol{\theta}, \boldsymbol{\theta}^{\sim}, \boldsymbol{\theta}', \boldsymbol{\theta}^*} &\pi_m(\boldsymbol{\theta})p(\boldsymbol{\theta}, \boldsymbol{\theta}^{\sim})q_{mn}(\boldsymbol{\theta}^{\sim}, \boldsymbol{\theta}')p(\boldsymbol{\theta}', \boldsymbol{\theta}^*)\alpha_{mn}(\boldsymbol{\theta}, \boldsymbol{\theta}^*)d\boldsymbol{\theta}d\boldsymbol{\theta}^{\sim}d\boldsymbol{\theta}'d\boldsymbol{\theta}^* \\ &= \int_{\boldsymbol{\theta}, \boldsymbol{\theta}^{\sim}, \boldsymbol{\theta}', \boldsymbol{\theta}^*} \pi_n(\boldsymbol{\theta}^*)p(\boldsymbol{\theta}^*, \boldsymbol{\theta}')q_{nm}(\boldsymbol{\theta}', \boldsymbol{\theta}^{\sim})p(\boldsymbol{\theta}^{\sim}, \boldsymbol{\theta})\alpha_{nm}(\boldsymbol{\theta}^*, \boldsymbol{\theta})d\boldsymbol{\theta}d\boldsymbol{\theta}^{\sim}d\boldsymbol{\theta}'d\boldsymbol{\theta}^*\end{aligned} \tag{5.6}$$

The acceptance probability for a forward move:

$$\alpha_{mn}(\boldsymbol{\theta}, \boldsymbol{\theta}^*) = \min\left\{1, \frac{\pi_n(\boldsymbol{\theta}^*)p(\boldsymbol{\theta}^*, \boldsymbol{\theta}')q_{nm}(\boldsymbol{\theta}', \boldsymbol{\theta}^{\sim})p(\boldsymbol{\theta}^{\sim}, \boldsymbol{\theta})}{\pi_m(\boldsymbol{\theta})p(\boldsymbol{\theta}, \boldsymbol{\theta}^{\sim})q_{mn}(\boldsymbol{\theta}^{\sim}, \boldsymbol{\theta}')p(\boldsymbol{\theta}', \boldsymbol{\theta}^*)}\right\} \tag{5.7}$$

and the reverse move:

$$\alpha_{nm}(\boldsymbol{\theta}^*, \boldsymbol{\theta}) = \min\left\{1, \frac{\pi_m(\boldsymbol{\theta})p(\boldsymbol{\theta}, \boldsymbol{\theta}^\sim)q_{mn}(\boldsymbol{\theta}^\sim, \boldsymbol{\theta}')p(\boldsymbol{\theta}', \boldsymbol{\theta}^*)}{\pi_n(\boldsymbol{\theta}^*)p(\boldsymbol{\theta}^*, \boldsymbol{\theta}')q_{nm}(\boldsymbol{\theta}', \boldsymbol{\theta}^\sim)p(\boldsymbol{\theta}^\sim, \boldsymbol{\theta})}\right\} \tag{5.8}$$

It is critically noted that 5.6, 5.7, and 5.8 now apply both to cases for $m > n$ and $n > m$.

5.4 Implementation

5.4.1 Initializing the Chain

A typical starting methodology for MCMC is to generate a random initialization for the state space and use the burn-in technique to find the typical set of the state space [58]. The chain quickly moves to an area of high posterior probability density, and the intermediate states of low posterior probability density are discarded prior to analysis.

For our experiments, we seed the burn-in by randomly sampling a large number (1000) of networks from the joint prior distribution, and proceed with the network with the lowest posterior score. This ensures that the initial network will not cause computational difficulties for the sampling program. We do this for each individual chain of a parallel sampling program, which effectively allows us to run multiple independent chains for each experiment.

We can use these multiple chains to consider a range of different starting architectures across the support of the architecture parameters. This allows us to assess whether there is bias towards the starting architectures in the analysis of the marginal posterior distributions for the number of layers or nodes.

5.4.2 Algorithm

The overall algorithm for performing trans-dimensional inference on the RJBNN is a Gibbs sampler which first updates the relevant architecture parameter (via an across-dimension move) and then the network weights and biases $\boldsymbol{\theta}$ (within-dimension move) followed by the hyperparameters (if applicable) as described in section 5.1.1.

Algorithm 8 describes the procedure for an across-dimension step. The NUTS algorithm as introduced in chapter 4 handles within-dimension moves, and is used to increase the acceptance probability of across-dimension moves via a palindromic delayed-rejection sampler as described in section 5.3. We refer to this algorithm as RJNUTS.

Algorithm 8 Trans-dimensional RJBNN Inference

```
 1: function RJBNNINFERENCE($\boldsymbol{\theta}$)
 2:     $k = \dim(\boldsymbol{\theta})$
 3:     $\boldsymbol{\theta}^{\sim} = \text{NUTS}(\boldsymbol{\theta})$        ▷ Palindromic delayed-rejection proposal 1
 4:     $h = $ random(birth, death)
 5:     if $h = $ birth then
 6:         $k' = k + 1$
 7:     else $h = $ death
 8:         $k' = k - 1$
 9:     end if
10:     $\boldsymbol{u} \sim g(\boldsymbol{u})$        ▷ Dimension matching
11:     $(\boldsymbol{\theta}', \boldsymbol{u}') = h(\boldsymbol{\theta}^{\sim}, \boldsymbol{u})$
12:     $\boldsymbol{\theta}^{*} = \text{NUTS}(\boldsymbol{\theta}')$        ▷ Palindromic delayed-rejection proposal 1
13:     $\alpha(\boldsymbol{\theta}, \boldsymbol{\theta}^{*}) = \min\left\{1, \frac{\pi(\boldsymbol{\theta}^{*}) p(\boldsymbol{\theta}^{*}, \boldsymbol{\theta}') g(\boldsymbol{\theta}', \boldsymbol{\theta}^{\sim}) p(\boldsymbol{\theta}^{\sim}, \boldsymbol{\theta})}{\pi(\boldsymbol{\theta}) p(\boldsymbol{\theta}, \boldsymbol{\theta}^{\sim}) g(\boldsymbol{\theta}^{\sim}, \boldsymbol{\theta}') p(\boldsymbol{\theta}', \boldsymbol{\theta}^{*})}\right\}$
14:     $\phi \sim$ unif(0,1)
15:     if $\phi \leq \alpha$ then
16:         return $\boldsymbol{\theta}'$        ▷ Accept the newly proposed state
17:     else
18:         return $\boldsymbol{\theta}$        ▷ Reject the proposed state, register the current state
19:     end if
20: end function
```

5.5 XOR Classification

XOR is a binary classification task for datapoints in a two-dimensional space. The datapoints are plotted in Gaussian-style clouds with centroids that form a square pattern for which the centroids/clouds corresponding to opposite vertices of the square are labelled equivallently. This essentially presents a very basic "toy" dataset that is not linearly separable via a basic logistic regression task (see figure 5.5).

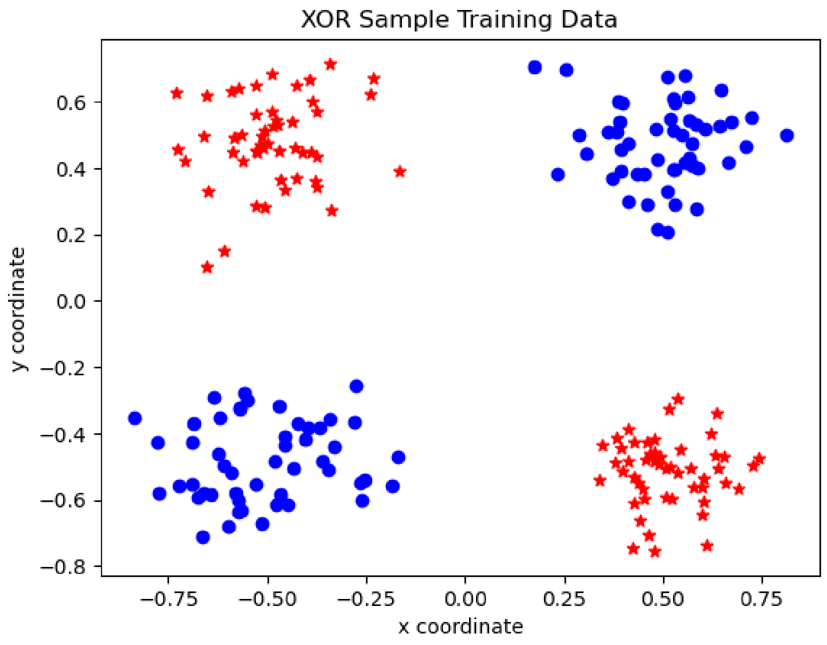


FIGURE 5.5: Sample training data for the XOR binary classification task

The data are not linearly separable, and a minimum of 2 hidden neurons are required to solve this task perfectly.

It is a known result within the space of neural computing that an ANN with a single hidden layer and a minimum of two hidden neurons is capable of perfectly classifying the data [77]. We therefore wonder whether a single-layer trans-dimensional BNN may represent this minimum capacity constraint in the marginal posterior distribution of an architecture parameter specifying the number of hidden neurons.

Network Definition

We implement a relatively basic RJBNN to examine the XOR classification task. One hidden layer with a variable k (the number of hidden neurons) is used. The likelihood is a categorical distribution with a binary input vector based on the output of the network with two output notes and a softmax activation function (this is for implementation convenience, and is equivalent to the typical BNN classifier definition of a Bernoulli likelihood with a probability parameter determined by the output of a single output node with a sigmoid activation function).

All of the weight and bias parameters are defined with independent, zero-centered standard Gaussian distributions. There are no hierarchical parameters affecting the variances, and as a classification task there is no need for a regression noise variance. k is drawn from a uniform discrete prior distribution ranging between 1 and 16.

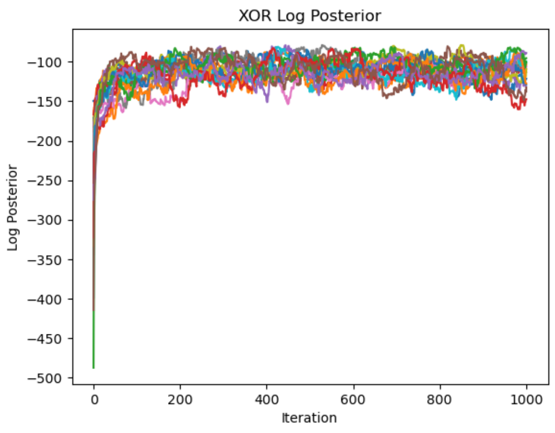
The program is implemented with 16 individual Markov chains running independently. One of each chain is initialized randomly based on an initial k value between 1 and 16, so that each possible neuron count is represented by a chain. 1000 iterations are sampled for each chain.

Results

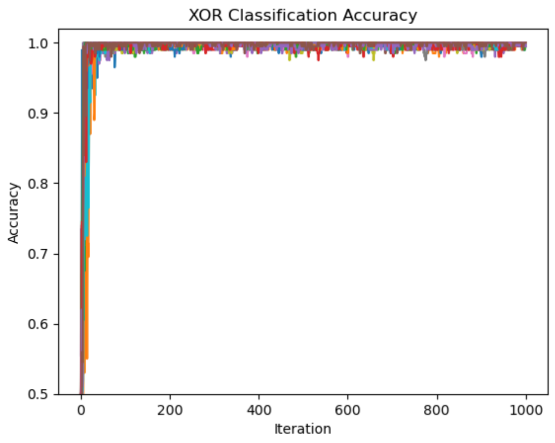
Figures 5.6a and 5.6b represent the success of the classifier. The log posterior demonstrates a quick burn-in period for each chain as the Markov Chains quickly sample areas of high posterior probability density. The classification accuracy only occasionally dips below 1.0 for each chain, as the problem definition allows for slightly noisy solutions. Marginalizing over the samples and taking the expected class labels based on a MCI of the posterior predictive distribution results in perfect classification accuracy on a test set.

The estimated marginal distribution of the k parameter is represented in figure 5.6c. Surprisingly, we see that any fewer than 4 hidden neurons are sampled infrequently. The mode appears to be around 7 neurons. The distribution appears quite smooth overall, and has a slightly longer tail to the right of the mode.

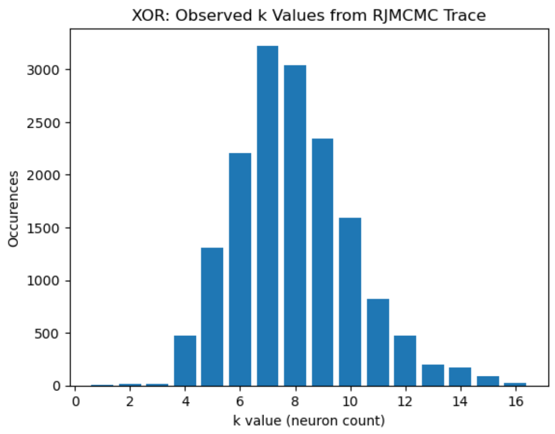
Figure 5.6d simply demonstrates the classification uncertainty for an extended region of the feature space. We see a great deal of certainty (darker red or blue) around the data clusters, and certain lighter colours or white where no training data is located. The results demonstrate non-ideal confidence behind the data point clusters. This suggests that certain modes of the posterior distribution have not been sampled, which may be expected given a short run. The resutls are not sufficient to make a claim about the calibration of the RJBNN in comparison to a standard BNN - such a discussion would be an interesting future research direction.



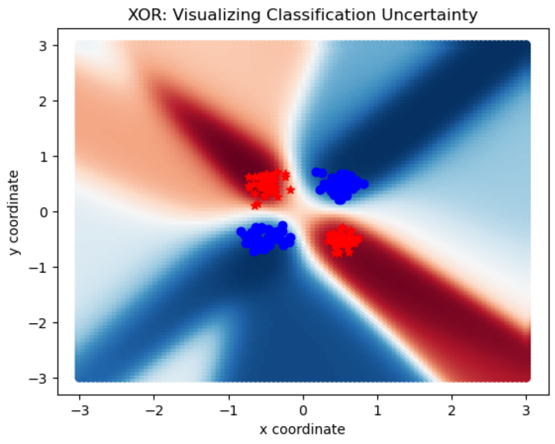
(A) Log Posterior



(B) Classification Accuracy



(C) Histogram of k Values



(D) Classification Heatmap

FIGURE 5.6: Results of RJNUTS inference for a single-layer BNN on the XOR binary classification task

5.6 Noisy XOR Classification

The results in section 5.5 demonstrated the expectation vs reality of an RJBNN applied to a classification task with a known constraint in terms of network size. It was expected that the samples obtained of the distribution of network architectures would favour those with two hidden neurons, being the minimum required to accurately separate and therefore perfectly classify the data. Instead, the estimate marginal posterior distribution suggested that some number around 7 hidden nodes presented the ideal capacity for solving the problem without using more nodes than necessary.

We now examine the behaviour of the same RJBNN when the XOR data is considerably noisier, such that even a network with 2 hidden nodes would be incapable of perfectly classifying the training data. Such a dataset would appear as in figure 5.7.

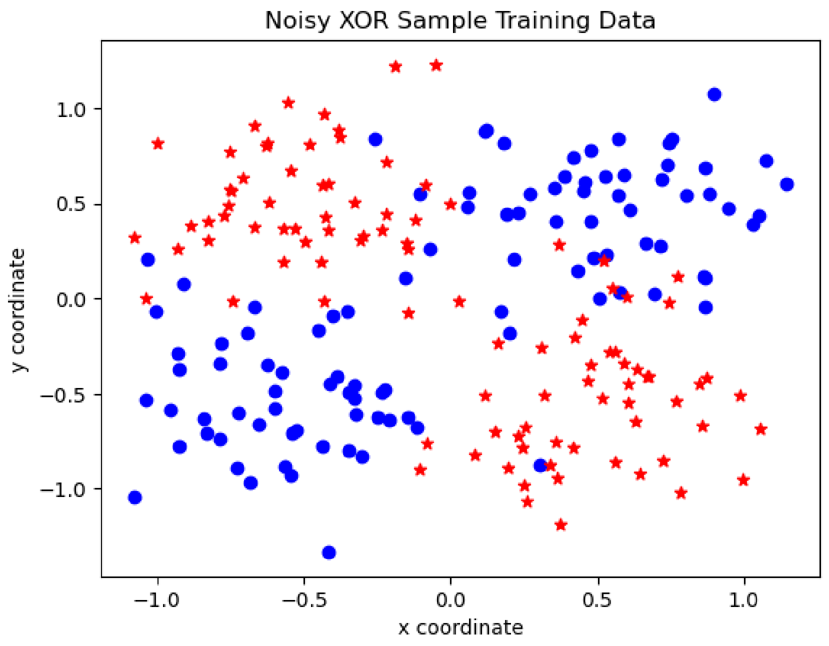


FIGURE 5.7: Sample training data for the noisy XOR binary classification task

The data are not linearly separable, even with a network with 2 hidden nodes.

Network Definition

The network is implemented exactly as in section 5.5 - only the training data has been adjusted. No changes are made to model priors, hyperparameters, or the training program.

Results

As with the standard XOR experiment, figures 5.8a and 5.8b demonstrate reasonable classification success via the RJBNN. The accuracy is considerably lower given the overlapping labelled training data. Individual samples fail to perfectly classify the network - nevertheless, MCI estimates of the posterior predictive distribution for a held-out test set result in 0.92 accuracy. This is a commendable result given the difficulty of the task.

We see in figure 5.8c that the estimated marginal posterior distribution of k follows the same shape as in the XOR task. The distribution is perhaps slightly shifted to the right, as the mode falls now on 8 neurons instead of 7. We might interpret this as the network suggesting additional capacity is needed to deal with the noisy data. This illustrates the main motivation behind using a RJBNN: the complexity of the data allows for specification of the network capacity. Practically speaking, we are marginalizing over a set of architectures, some of which we believe to be sufficient for solving the modelling task at hand.

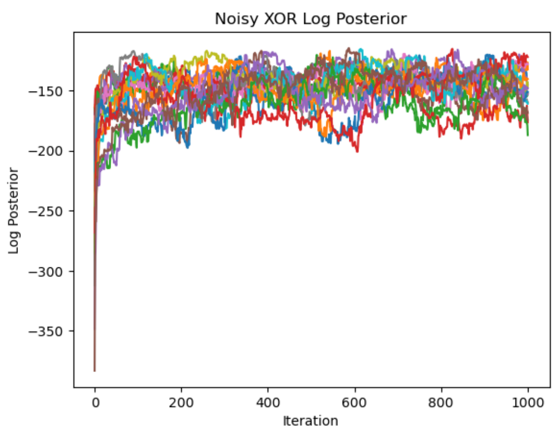
Interestingly, there is slightly better calibration with regards to the out-of-distribution regions of the feature space as in figure 5.8d. It may be that the noisy data motivated more varied solutions in terms of the MCMC sampling program. This is not empirically evaluated, and is strictly an observation based on the uncertainty visualization.

5.7 Related Works

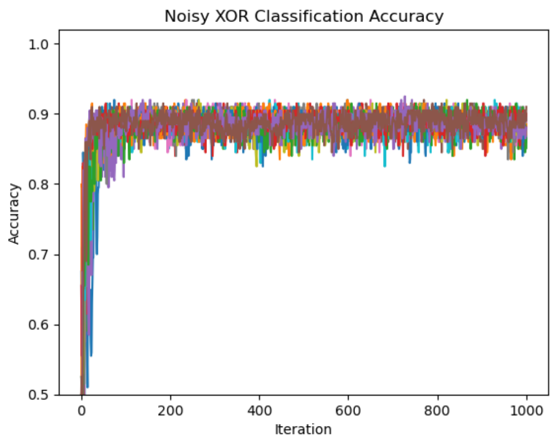
RJNUTS follows from a composite sampling method introduced with arbitrary fixed-dimensional Metropolis-Hastings proposals [76] and the use of HMC to update the state of fixed architecture BNNs [29]. RJHMC, the HMC pre-cursor to RJNUTS, is developed and applied in a geology setting [78] and a similar algorithm has combined Riemannian Manifold HMC [79] with RJMCMC [80].

RJMCMC has previously been used to learn the marginal posterior distribution for network width (number of nodes) in a BNN with a single hidden layer [81]. Only small networks of one hidden layer were used. The number of neurons is treated as the sole discrete architecture parameter, in comparison to select experiments in this work which tackle the optimal number of layers. Only regression problems are treated in the prior work; no classification task is examined. Using the RJNUTS delayed rejection sampler to scale the inference to larger problems is also introduced as a first for RJBNN training in this work.

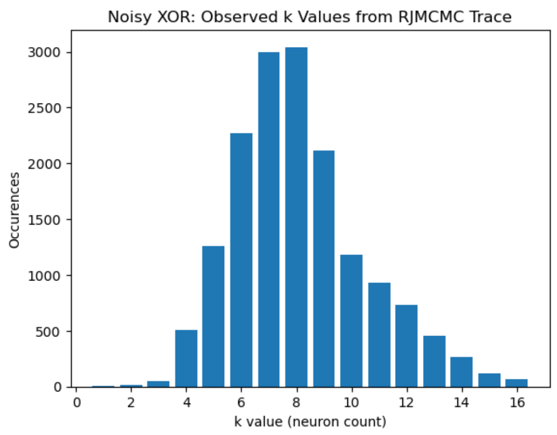
Bayesian learning of network architectures is examined using an SVI approach in [82]. SVI approaches are also used for simultaneous learning of BNN parameters and models (architectures) in [83].



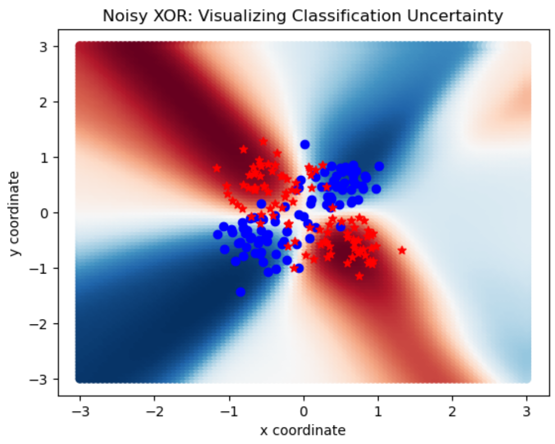
(A) Log Posterior



(B) Classification Accuracy



(C) Histogram of k Values



(D) Classification Heatmap

FIGURE 5.8: Results of RJNUTS inference for a single-layer BNN on the XOR binary classification task

Chapter 6

Experiment Results

We now demonstrate how RJMCMC can be used to extend a BNN model to a RJBNN which samples a distribution of BNN architectures.

Experiment definitions and results are presented in this chapter. Discussion focuses on demonstrating that the models obtain reasonable test set predictive capabilities, and that across-dimension proposals are accepted at a reasonable rate. Further discussion about the advantages and drawbacks to the RJBNN approach is presented in the following and final chapter of the thesis (7).

6.1 Overview

RJBNNs will be examined for a classification network and a regression network. For the classification problem, a network with one hidden layer ($\mathcal{L} = 1$) will be drawn from a distribution of network architectures with a varying number of hidden nodes $\mathcal{K}$ over a range of $k \in \boldsymbol{K}$ values, representing the number of neurons in the one hidden layer, where the support $\boldsymbol{K} = [1, \ldots, K_{max}]$. RJMCMC proposals are limited to neuron birth and neuron death moves as presented in section 5.2.2.

For the regression task, a network will be drawn with $\ell \in \boldsymbol{L}$ hidden layers, with the support $\boldsymbol{L} = [1, \ldots, L_{max}]$. Each layer has a fixed number of $\mathcal{K} = k$ neurons. RJMCMC proposals are limited to layer birth and layer death moves as presented in section 5.2.2.

Classification is the focus for the case of adding and removing neurons, while the regression network deals with a variable number of layers. These two experimental design variables seem to be independent of each other, and could be easily switched - as mentioned, regression has previously been assessed for the variable network width case [81].

6.2 Classification Network: OptDigits

The OptDigits training dataset [84] comprises 5620 hand-written digits between 0 and 9. Each image is represented as a 8x8 pixel image, with each pixel taking a value between 0 and 255 (Figure 6.1). The dataset is similar to the larger MNIST [85] dataset, which is popular for testing ML classification models. MNIST figures are 28x28 pixels, corresponding to 784 feature variables in a feedforward network. OptDigits figures are 8x8 pixels for a total of 64 feature variables. OptDigits samples are preferable here given the small networks that are to be assessed for trans-dimensional inference.

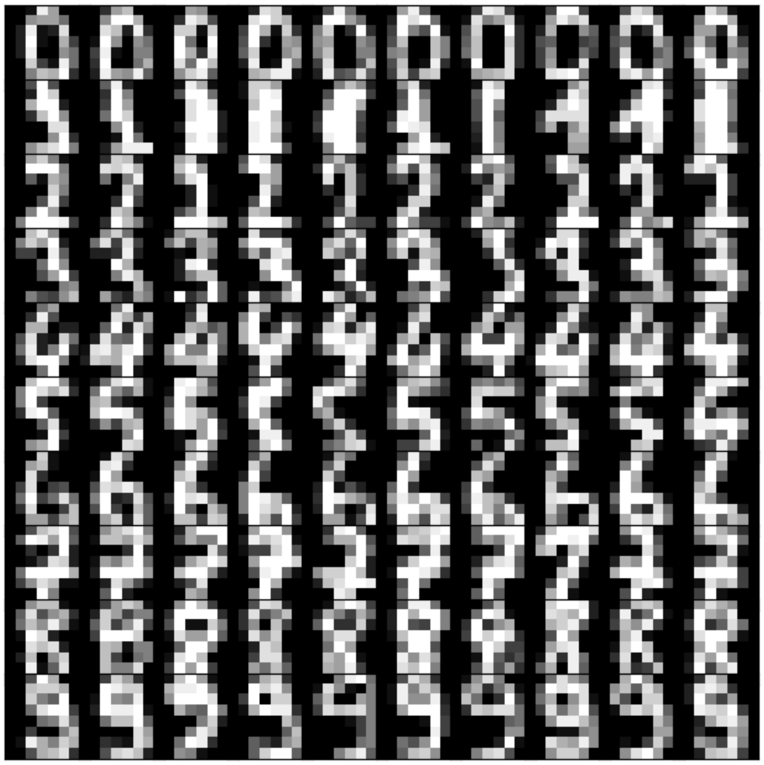


FIGURE 6.1: 100 random samples from the OptDigits hand-written digits training set, balanced by class label

For the RJBNN models, the full dataset is much too large. For the first experiment, we attempt classification of 5 classes. We proceed with a small balanced sample of 250 images, 50 of each digit 0 through 4, randomly selected from the full training set. A balanced set of 1000 images are similarly drawn at random from the test dataset. For the second experiment, all 10 classes are used, with 50 training samples for each class, and 200 test samples for a total of 500 and 2000 training and test samples respectively.

The goal is not to challenge state-of-the-art classification accuracy on the test set, but to demonstrate the features of the RJBNN on a classification problem. The metric of interest in this experiment, as is common in classification problems, is the network's classification accuracy on a test set of samples. Published results on MNIST and OptDigits are nearly perfect; we will not strive to improve on these. Test set accuracy will serve as a metric to assess whether the network is performing reasonably well.

Along with test set accuracy, we will be able to use the predictive uncertainty afforded to us by our Bayesian approach (via the posterior predictive distribution) to further calibrate the accuracy of our model (see section 7.1.1).

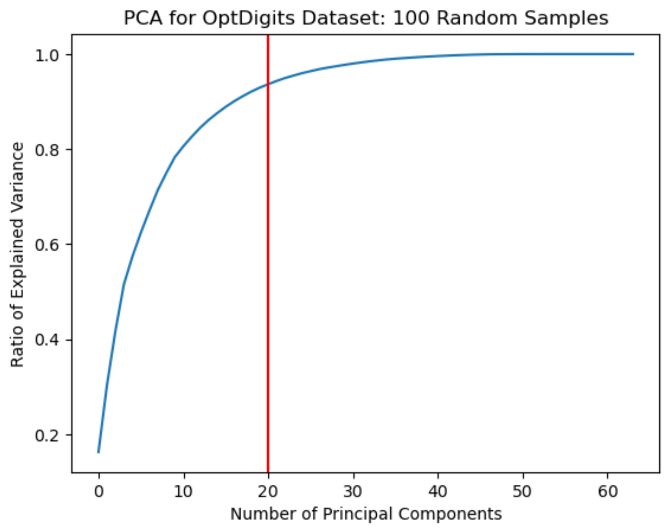


FIGURE 6.2: Ratio of preserved variance for up to 64 principal components on 100 OptDigits samples

The dimensionality of the OptDigits dataset implies a high computational demand for a feed-forward network. Each image corresponds to $8 * 8 = 64$ features, which in the case of an ANN results in an input layer size of 64. Given k neurons in the first hidden layer, the number of parameters in the first layer alone will be $64 * (k + 1)$ for the weights and biases, which quickly becomes untenable for a reasonable sized neural network. PCA as introduced in section 2.2.2 is therefore used to reduce the number of features. We choose to proceed with 20, capturing approximately 0.93 of the total variance of the randomly sampled balanced training set (figure 6.2).

We make one note on the output activation for the network design in these experiments. An optional tempering parameter a is introduced to the softmax equation (equation 2.6) to introduce flexibility to the concentration of class label outputs (equation 6.1).

$$g(z)_j = \frac{e^{az_j}}{\sum_{i=1}^{c} e^{az_i}} \tag{6.1}$$

6.2.1 OptDigits Experiment: 5 Classes

The network is proposed with one hidden layer. The hidden node count k is drawn from a discrete uniform distribution over a range of $[1, 64]$. The priors for network weight and biases are zero-centered Gaussians with the variance parameter set to 1.0. No precision hyperparameters were used for this experiment.

A balanced training set of 250 samples is randomly drawn from the training set. Classification accuracy is reported on a 1000 balanced samples from the test set.

Two runs were conducted. For experiment 10a, the softmax function is adjusted with a tempering weight of $a = 0.1$, and for experiment 10b $a = 0.5$.

The program is run on a high-performance computing unit with 16 individual chains sampling the RJBNN in parallel. Each chain with is initialized with $4 * i$ hidden neurons, where i is the chain number indicator (1 through 16). 1000 samples are recorded for each chain.

In terms of improving chain initialization, the RJBNN for each chain is randomly sampled 1000 times prior to running the RJNUTS program, and the chain with the best log-posterior score is used to limit the required burn-in and avoid numerical issues.

6.2.2 OptDigits Experiment: 10 Classes

The network is proposed with one hidden layer. The hidden node count k is drawn from a discrete uniform distribution over a range of $[1, 128]$. The priors for network weight and biases are zero-centered Gaussians with the variance parameter set to 1.0. No precision hyperparameters were used for this experiment.

A balanced training set of 250 samples is randomly drawn from the training set. Classification accuracy is reported on a 1000 balanced samples from the test set.

Two runs were conducted. For experiment 10a, the softmax function is adjusted with a weight of $a = 0.1$, and for experiment 10b the weight is $a = 0.5$.

The program is run on a high-performance computing unit with 16 individual chains sampling the RJBNN in parallel. Each chain with is initialized with $8 * i$ hidden neurons, where i is the chain number indicator (1 through 16). 1000 samples are recorded for each chain.

In terms of improving chain initialization, the RJBNN for each chain is randomly sampled 1000 times prior to running the RJNUTS program, and the chain with the best log-posterior score is used to limit the required burn-in and avoid numerical issues.

6.2.3 Assessment of Network Predictions

For a Markov chain corresponding to samples from the RJBNN model, we may generate predicted labels and corresponding training and test accuracy scores for each associated network. These are presented as plots (figures 6.3a, 6.3b, 6.3c, 6.3d) representing the accuracy over the trajectory of the chain. The results present evidence that the sampler was implemented correctly, and that we may proceed with analysis of the RJBNN output with reasonable confidence.

However, to assess only one network - even the most optimal as determined by the highest test accuracy - is to throw away much of the output of Bayesian inference. We may instead marginalize the predictive distribution over all parameterizations as returned by the program, and obtain expectations for each target class label. Full results of the predictive accuracy for the four experiments are reserved for section 7.1 in the final chapter of this thesis.

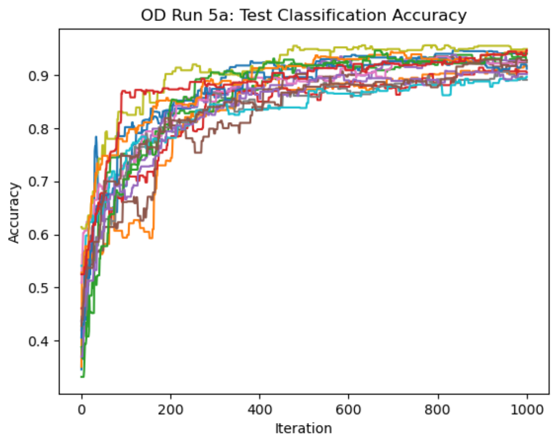
6.2.4 Estimated Hidden Layer Width

The parameter of interest in classification experiments for the OptDigits dataset is k, the number of nodes in the hidden layer of the network. Assessing the counts of different values of k over the trajectory of the trans-dimensional Markov chain [1] returns an approximation of the discrete marginal posterior distribution of network architectures under this variable network width parameter. We hypothesize the following features of this distribution prior to observing any data:

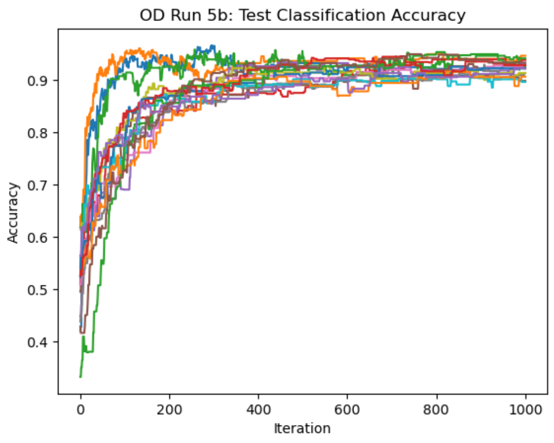
1. The distribution is discrete. This is implicit from the uniform discrete prior distribution over k
2. The distribution should have a distinct mode that corresponds to the optimal network width based on the problem definition
3. Occam's razor should be implicit in the counts of possible k values:
    (a) We expect a sharp drop-off in counts of k-values smaller than the mode, indicating insufficient network capacity for the given problem
    (b) We expect a gradual drop-off in k-values exceeding the mode, indicating that sufficient network capacity has been exceeded and that additional parameters are adding an unnecessary expense to the log-posterior score

Figures (6.4a, 6.4b, 6.4c, 6.4d) represent the counts of k values returned from the Markov chains for OptDigits experiment 5A and 5B (section 6.2.1) and OptDigits experiment 10A and 10B (section 6.2.2) respectively.

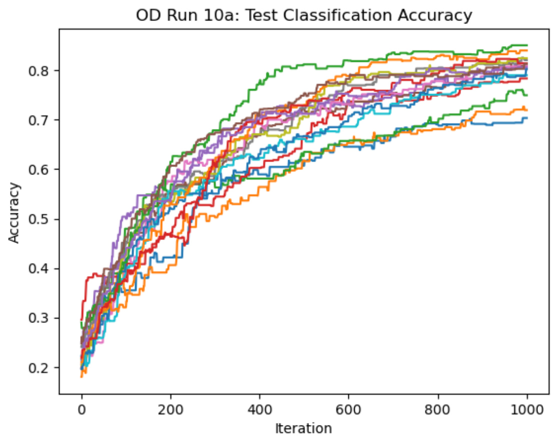
[1]Following burn-in; after the Markov chain is expected to have reached the stationary distribution



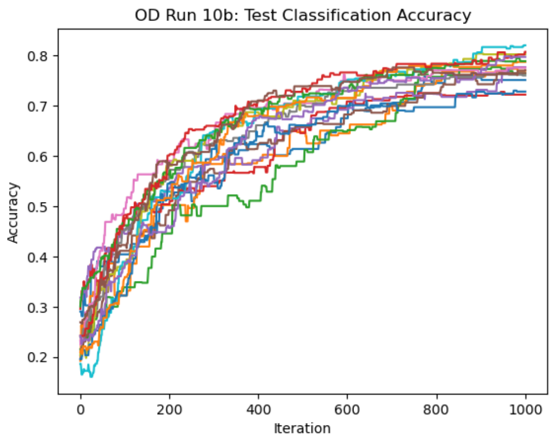
(A) Experiment 5a



(B) Experiment 5b

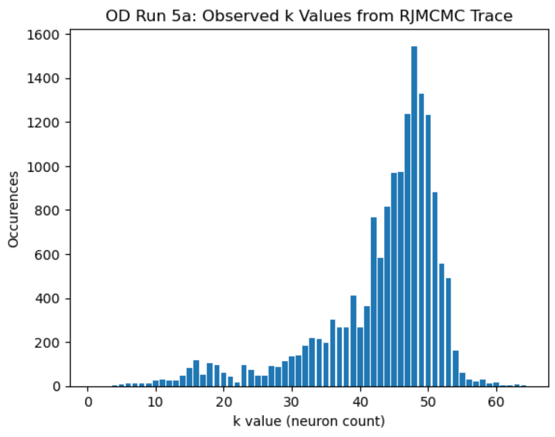


(C) Experiment 10a

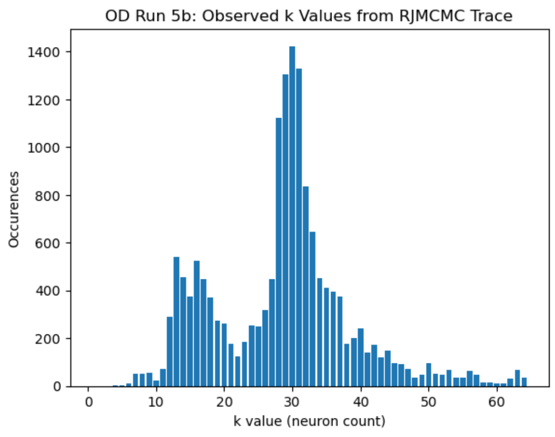


(D) Experiment 10b

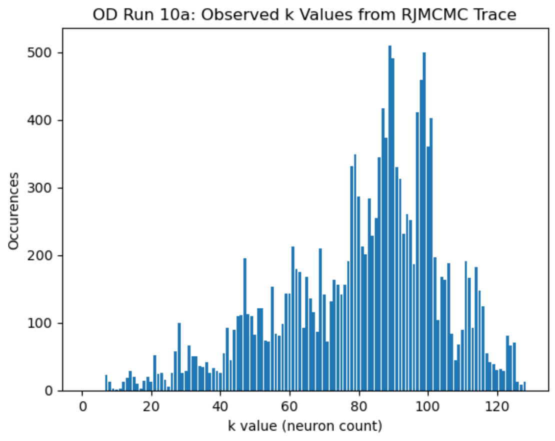
FIGURE 6.3: Observed frequency of k: hidden node counts for networks defined by all iterations of all chains of the RJNUTS sampling program



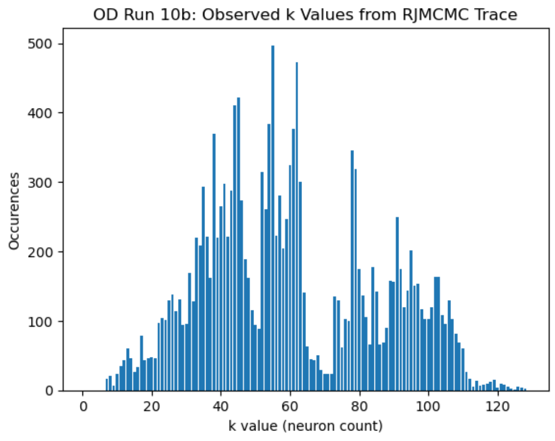
(A) Experiment 5a



(B) Experiment 5b



(C) Experiment 10a



(D) Experiment 10b

FIGURE 6.4: Observed frequency of k: hidden node counts for networks defined by all iterations of all chains of the RJNUTS sampling program

6.3 Regression Network: Boston Housing

The Boston Housing dataset [86] consists of 506 observations of 14 attributes corresponding to U.S. census data for neighbourhoods in Boston. It is commonly assessed in statistical literature as a method for benchmarking regression models, such that the median housing price index is modelled depending on the remaining 13 attributes. Full details of all datasets are included in Appendix A.

Of particular interest to our RJBNN model, the Boston Housing dataset was one assessed by Neal in his seminal work on BNNs [29]. It therefore receives attention in other BNN works, including the recent examination of hyperparameter granularity by Javid et al. [47]. Both works used similar sizes of small networks to the architectures that we will sample, and the results from these papers may serve as benchmarks for assessing the predictive capabilities of our RJBNNs. The metric used in this regression task is the root mean square error (RMSE) between the predicted median housing prices and the observed values for a randomly partitioned test set.

6.3.1 Boston Housing Experiment: 2 Node Width

The network is proposed with 2 hidden nodes per layer. The layer count ℓ is drawn from a discrete uniform distribution over a range of $[1, 8]$. The priors for network weight and biases are zero-centered Gaussians with the variance parameter set to 1.0. No precision hyperparameters were used for this experiment.

A training set of 256 samples is randomly drawn from the 512 observations. Accuracy is reported as RMSE for the test set of the remaining 256 samples.

Two runs were conducted. The likelihood function over the observed data is the product of independent Gaussian distributions centered on the network output with a variance parameter set to 1.0 for run A, and 0.8 for run B.

The program is run on a high-performance computing unit with 16 individual chains sampling the RJBNN in parallel. For each possible layer count of 1-8, two chains are randomly initialized. 1000 samples are recorded for each chain.

In terms of improving chain initialization, the RJBNN for each chain is randomly sampled 1000 times prior to running the RJNUTS program, and the chain with the best log-posterior score is used to limit the required burn-in and avoid numerical issues.

6.3.2 Boston Housing Experiment: 4 Node Width

The network is proposed with 4 hidden nodes per layer. The layer count ℓ is drawn from a discrete uniform distribution over a range of $[1, 4]$. The priors for network weight and biases are zero-centered Gaussians with the variance parameter set to 1.0. No precision hyperparameters were used for this experiment.

A training set of 256 samples is randomly drawn from the 512 observations. Accuracy is reported as RMSE for the test set of the remaining 256 samples.

Two runs were conducted. The likelihood function over the observed data is the product of independent Gaussian distributions centered on the network output with a variance parameter set to 1.0 for run A, and 0.8 for run B.

The program is run on a high-performance computing unit with 16 individual chains sampling the RJBNN in parallel. For each possible layer count of 1-4, four chains are randomly initialized. 1000 samples are recorded for each chain.

In terms of improving chain initialization, the RJBNN for each chain is randomly sampled 1000 times prior to running the RJNUTS program, and the chain with the best log-posterior score is used to limit the required burn-in and avoid numerical issues.

6.3.3 Assessment of Network Predictions

Each individual trace represents a fully-specified neural network model for representing the median housing price $\boldsymbol{y}$ based on the feature data $\boldsymbol{x}$. With our Bayesian approach, we marginalize over a set of these trace observations to obtain an approximate expectation for the targets. We may marginalize by chain, conditioned on the ℓ parameter, or the entire set of observations returned by the program. We reserve these observations for the discussion of network predictive accuracy in 7.1.

The health of the individual chains may be monitored by assessing the test error for each iteration of the sampler. We expect these results to be reasonably close but ultimately inferior compared to the marginalized test error. The plots also suggest at which iteration the chains have completed the burn-in. Figures 6.5a, 6.5b, 6.5c, and 6.5d present the test error plots for each chain of the four experiments. The results present evidence that the sampler was implemented correctly, and that we may proceed with analysis of the RJBNN output with reasonable confidence.

6.3.4 Estimated Network Depth

Similar to the argument regarding k in the case of the experiments for the OptDigits dataset in section 6.2.4, the histogram of ℓ values from the Markov chain's trajectory represents an approximation of the discrete marginal posterior distribution of network architectures based on the variable network depth parameter. We entertain similar hypotheses to those postulated for the network width in the OptDigits experiments:

1. The distribution is discrete. This is implicit from the uniform discrete prior distribution over ℓ
2. The distribution should have a distinct mode that corresponds to the optimal network depth based on the problem definition
3. Occam's razor should be implicit in the counts of possible ℓ values:
   (a) We expect a sharp drop-off in counts of ℓ-values smaller than the mode, indicating insufficient network capacity for the given problem
   (b) We expect a gradual drop-off in ℓ-values exceeding the mode, indicating that sufficient network capacity has been exceeded and that additional parameters are adding an unnecessary expense to the log-posterior score

Figures 6.6a, 6.6b, 6.6c, and 6.6d represent the counts of ℓ values returned from the Markov chains for Boston Housing experiment A (section 6.3.1) and Boston Housing experiment B (section 6.3.2) respectively.

6.4 Acceptance Probability

The acceptance probability of across-dimension moves is a key metric in determining whether our implementation of the RJNUTS inference algorithm was effective for exploring varying network architectures. We're looking for non-negligible results ($> 1\%$) to ensure that we have at least approximate marginal posterior distributions of architecture parameters.

Network	**Across-Dimension**	**Within-Dimension**
OD5a	11.2%	82.5%
OD5b	9.8%	64.3%
OD10a	7.8%	67.2%
OD10b	5.6%	57.6%
BH2a	4.5%	66.1%
BH2b	4.9%	65.8%
BH4a	1.1%	63.4%
BH4b	1.4%	64.8%

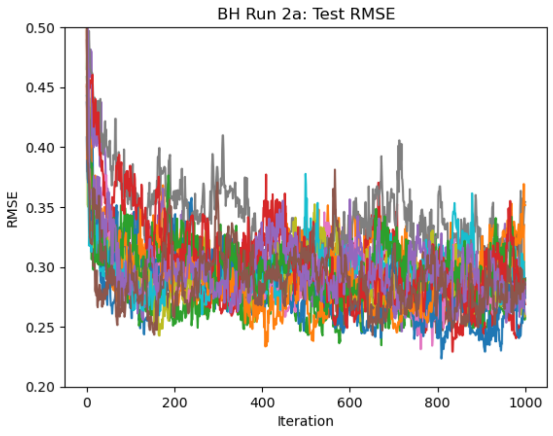
TABLE 6.1: Experiment proposal acceptance probabilities

Table 6.1 displays the proposal acceptance rates, expressed as percentages, for the eight experiments. We use these results to confirm that RJNUTS was implemented successfully, and further discuss the model inference implications in section 7.2.

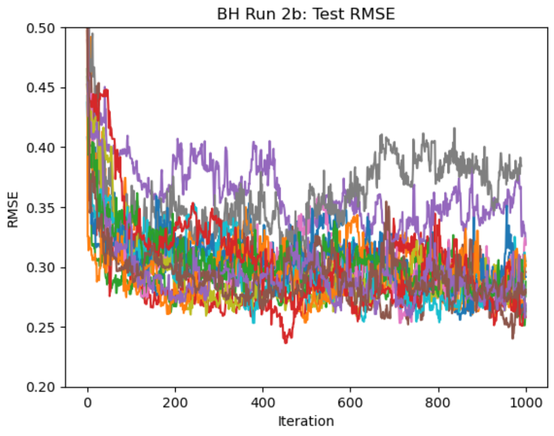
6.5 Summary of Experiments

Experiment definitions and select results have been presented for a classification and regression task using RJBNN models. The precise implementation details were motivated by topics explored in the preceding chapters. The experiments themselves were conducted to test whether or not RJBNN inference would work in a computationally feasible amount of time.

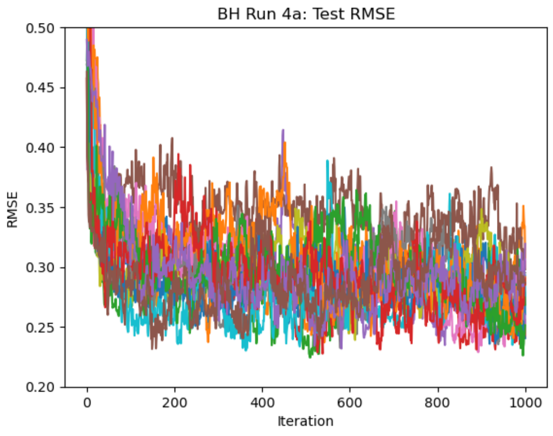
With this in mind, the results published in this chapter are only those which demonstrate effective predictive capabilities of the network, and the across-dimension "jumping" behaviour that implies the RJMCMC inference to be working. A full analysis and discussion of results of the RJBNN inference programs are explored in the next and final chapter of this report.



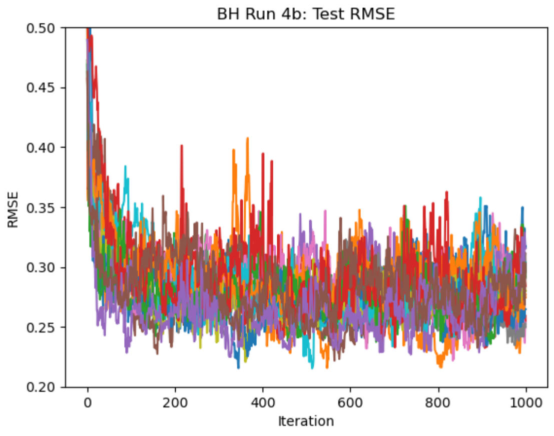
(A) Experiment 2a



(B) Experiment 2b

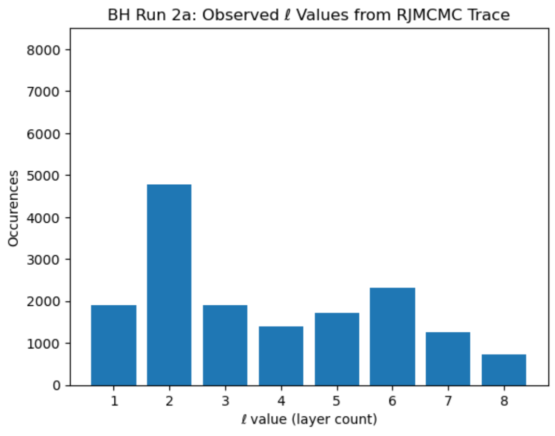


(C) Experiment 4a

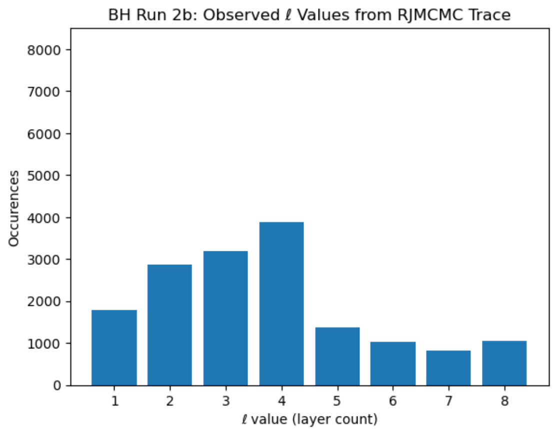


(D) Experiment 4b

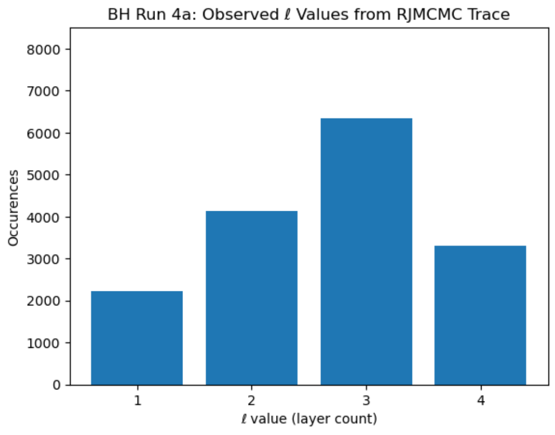
FIGURE 6.5: Test error: RMSE for output of test set observations fitted by networks defined by all iterations of all chains of the RJNUTS sampling program



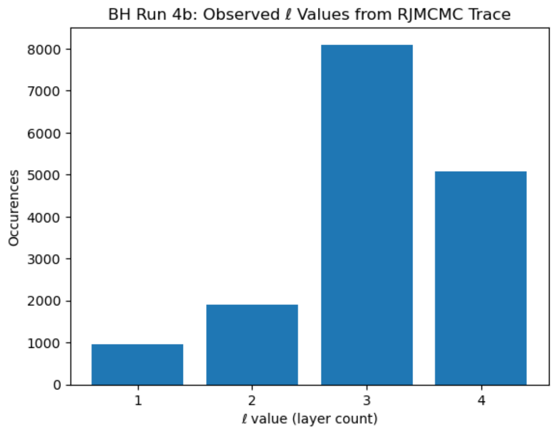
(A) Experiment 2a



(B) Experiment 2b



(C) Experiment 4a



(D) Experiment 4b

FIGURE 6.6: Observed frequency of ℓ: layer counts for networks defined by all iterations of all chains of the RJNUTS sampling program

Chapter 7

Discussion

RJMCMC can be theoretically defined for a BNN with variable architecture. One of the questions that was proposed in the design of this thesis is whether one could be practically implemented to any degree of success. What would qualify as a successful implementation is open to interpretation, but certainly the discussion would involve an assessment of the results regarding the two products of Bayesian inference discussed in chapter 3:

1. The predictive power of the model based on the posterior predictive distribution
2. An inferential understanding of the model described by the approximate posterior distribution

In this chapter, we will first discuss the predictive capabilities of the networks as a reference point for how well the networks performed from a machine learning perspective. Any serious ML model, classical or Bayesian, should be capable of achieving results comparable to competing models on datasets or experiments similar in scope. After that, we will assess whether RJNUTS has resulted in a reasonable estimate of the marginal posterior distributions of architecture parameters and network parameterizations. Final words on the relevance of this project and future opportunities suggested by the results conclude the chapter and report.

7.1 Results from an Optimization Perspective

The marginalization approach of BNNs offers the potential for gains in predictive accuracy compared to standard neural networks. [87]. A potential hypothesis might therefore concern whether extending these models to sample different architectures will introduce yet more accuracy to our posterior predictions.

The results suggest that predictive capabilities of the small RJBNNs examined in chapter 6 are comparable to standard BNNs. We refer to the publication of Javid et al. [47] in

Network	**Marginal Test RMSE**	**Layer Count Mode**	**Optimal Layer Count [47]**	**Optimal Layer Count RMSE [47]**
RJBNN 2A	0.236	2	1	0.275
RJBNN 2B	0.245	4	1	0.275
RJBNN 4A	0.223	3	1	0.271
RJBNN 4B	0.215	3	1	0.271

TABLE 7.1: Predictive RMSE for Boston Housing Experiments

terms of the Boston Housing experiments. Table 7.1 displays a comparison of the results for various architectures examined in our experiments and in theirs.

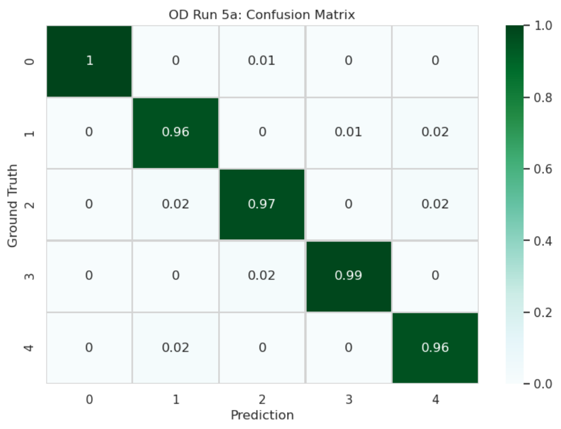
Results were also reasonable for the OptDigits dataset. 100% test accuracy would be preferable, but the networks performed well considering the small training sets and the PCA feature transformation. Confusion matrices presented in figures 7.1a, 7.1b, 7.1c, and 7.1d also demonstrate that test errors were not condensed to any one class, indicating that the networks learned to distinguish between all 5 or 10 classes. The results were particularly good for the experiments on a subset of 5 of the 10 classes, for the digits 0-4. Results of test set accuracy for these experiments are presented in table 7.2, column "Top 1".

7.1.1 Uncertainty Consideration in Predictive Accuracy

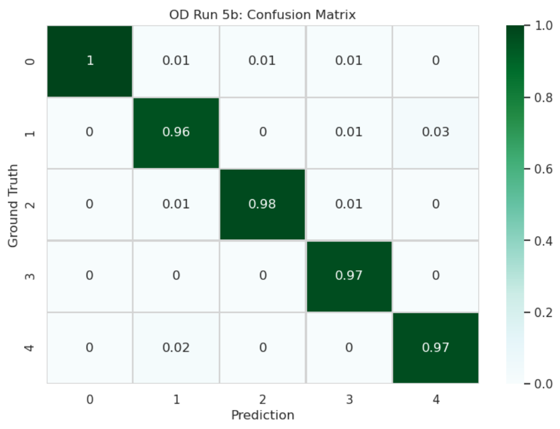
A convenient feature of favouring a Bayesian approach is that we have a measure of uncertainty in our posterior predictive distribution for any test data sample. We refer specifically to the OptDigits classification task for this discussion.

For any test sample that has been classified or interpolated incorrectly (as in figure 7.2), we may be interested in the confidence of the RJBNN's predictions for that sample. Focusing on an individual sample, we can generate the predictions from each individual sample network from the posterior distribution as represented by states of the 16 Markov chains. A bar graph of these predictions for the first delinquent training sample is displayed in figure 7.3.

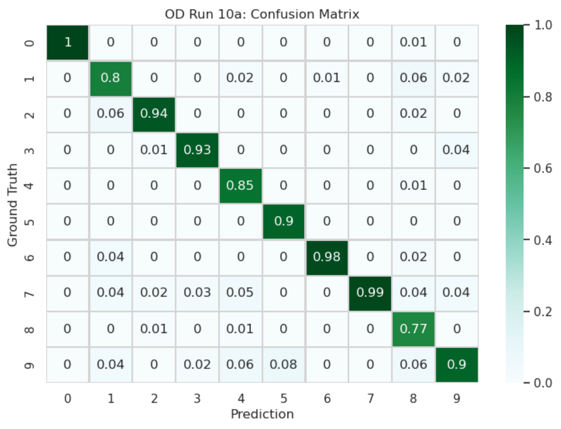
We see that only 22 percent of the networks predict the ground truth label of 1, while 26 percent of the networks incorrectly predict that the sample is a 9. Taking the mode of the marginal posterior predictive distribution for this sample, as we have reported in the predictive test set accuracy for our experiments, results in a misclassification. We are however under no obligation to accept the mode as the entire story. For one, the predictive certainty is low for the mode - the model has effectively less than 30% confidence in this misclassification. Compare this to a typical test sample that has been correctly classified as in figure 7.4, for which the predictive confidence in the correct class label is above 50%.



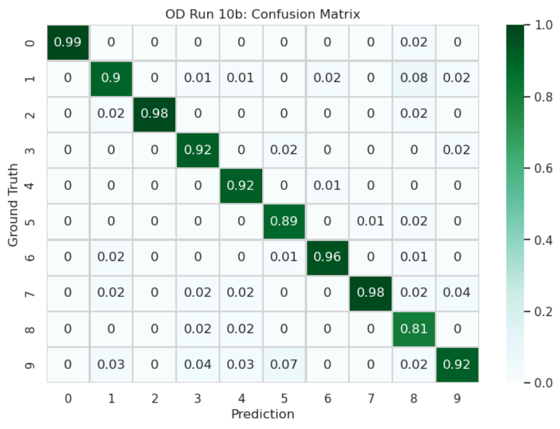
(A) Experiment 5a



(B) Experiment 5b



(C) Experiment 10a



(D) Experiment 10b

FIGURE 7.1: Confusion matrices: predictions versus ground truth for test samples in the OptDigits RJBNN classification experiment

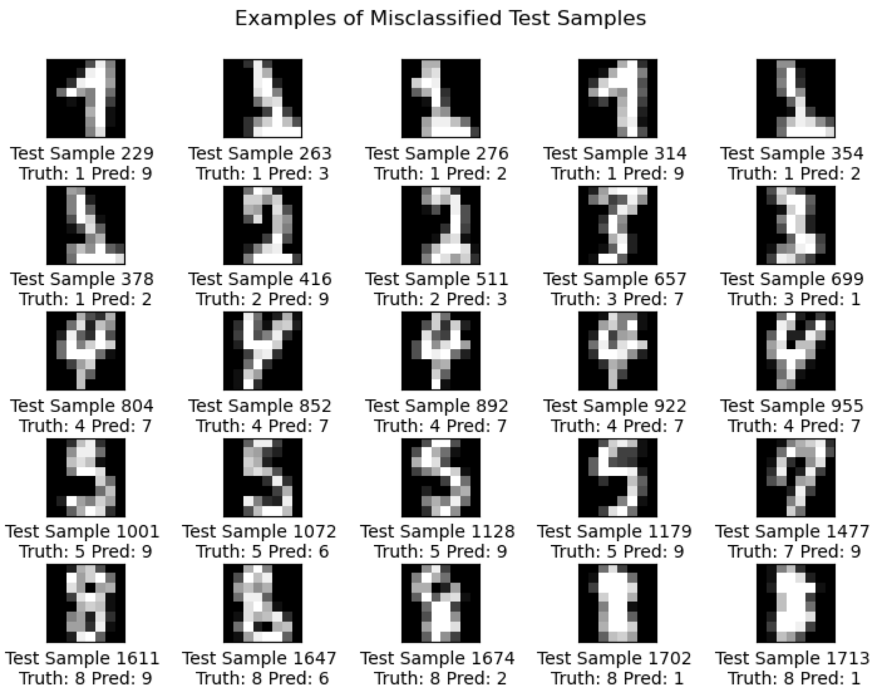


FIGURE 7.2: Misclassified test samples for OptDigits experiment 10a

Table 7.2 demonstrates the improvements that can be made to the test set accuracy by considering multiple predicted labels from the predictive distribution for each target (columns: top 2, top 3). We may also only consider predictions above a certain confidence threshold, and regard the *precision* (accurately classified samples out of those above the confidence threshold) and the *recall* (accurately classified targets above the confidence threshold out of all accurately classified targets) as measures of the predictive capabilities of our network.

	Marginal Test Set Accuracy			**Precision (P) and Recall (R)**	
Network	**Top 1**	**Top 2**	**Top 3**	**Threshold 0.2**	**Threshold 0.4**
OD 5a	97.50%	99.30%	99.90%	P: 97.5 R: 100%	P: 99.45% R: 92.82%
OD 5b	97.80%	99.30%	99.80%	P: 97.8 R: 100%	P: 99.34% R: 92.43%
OD 10a	90.65%	96.50%	98.25%	P: 92.16% R: 98.73%	P: 100% R: 50.47%
OD 10b	92.60%	97.30%	98.40%	P: 92.5% R: 100%	P: 97.73% R: 91.07%

TABLE 7.2: Test set accuracy results for OptDigits Experiments

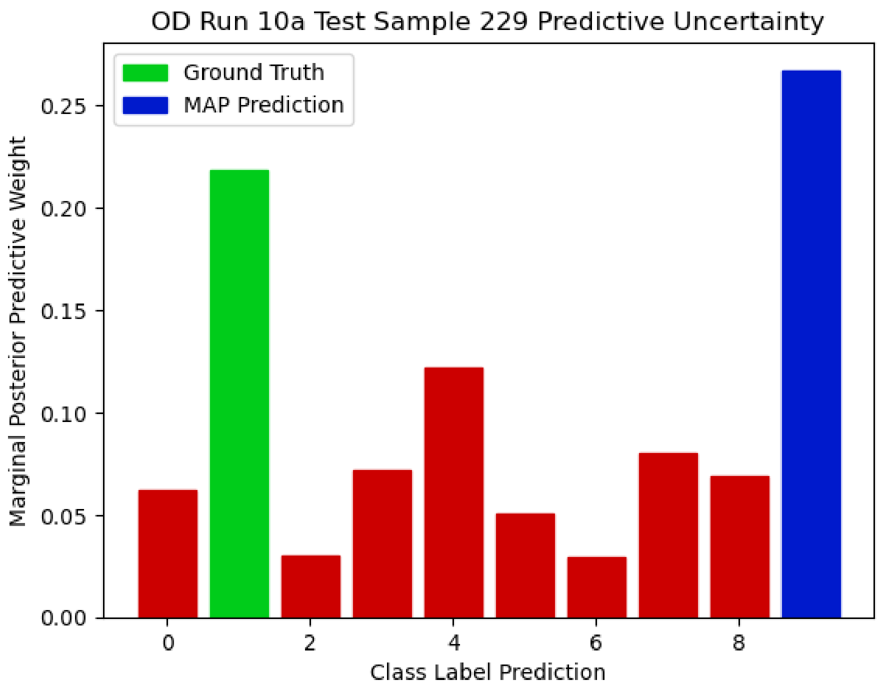


FIGURE 7.3: Bar graph of class label predictions for an individual misclassified test sample from OptDigits experiment 10a

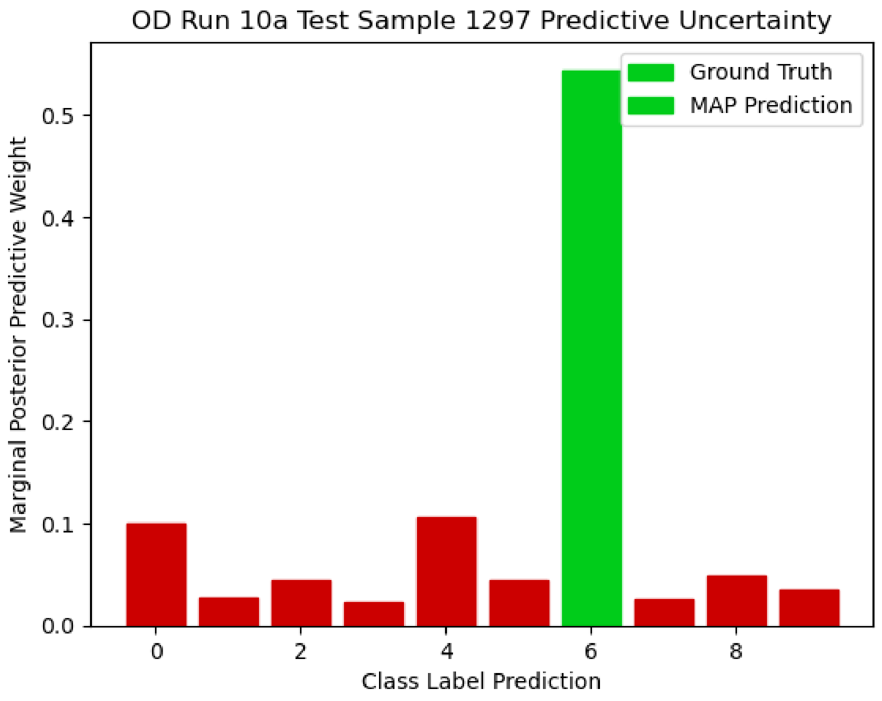


FIGURE 7.4: Bar graph of class label predictions for an individual correctly classified test sample from OptDigits experiment 10a

7.2 Results from a Model Inference Perspective

The novel product of our RJBNN inference program is a marginal posterior distribution over the model indicator for the network architecture. NAS approaches aim to optimize network design, but no fully Bayesian approach has been employed to produce an expectation for a model architecture.

Chain convergence

We first examine whether the our MCMC chains have converged - i.e, whether we can trust our estimates of the marginal posterior distributions for the parameters of interest. Reasonable predictive metrics on test sets as displayed in section 7.1 confirm that our models are well-inferred from the optimization standpoint that is typically of primary interest in an ML setting. From an inference perspective, there is somewhat less guarantee that our models have converged.

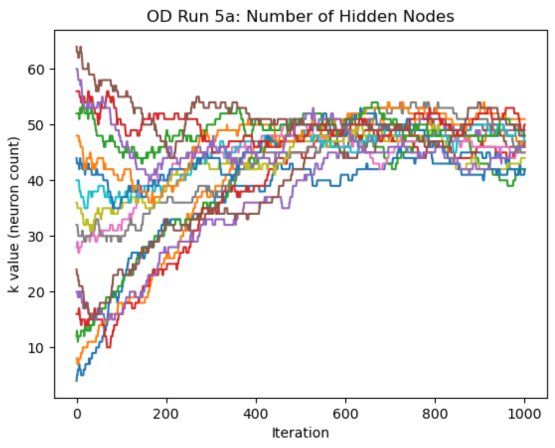
Referring to figures 7.5a, 7.5b, 7.5c, 7.5d for the OptDigits experiments, it does not necessarily appear as though the chains had an appropriate amount of time to burn-in and mix between network architecture specifications. Figures 7.6a, 7.6b, 7.6c, 7.6d suggest better mixing for the Boston Housing experiments, which may be a symptom of the smaller supports used for the variable layer depth parameter $\mathcal{L}$ compared to the variable layer width parameter $\mathcal{K}$.

Architecture parameter expectations

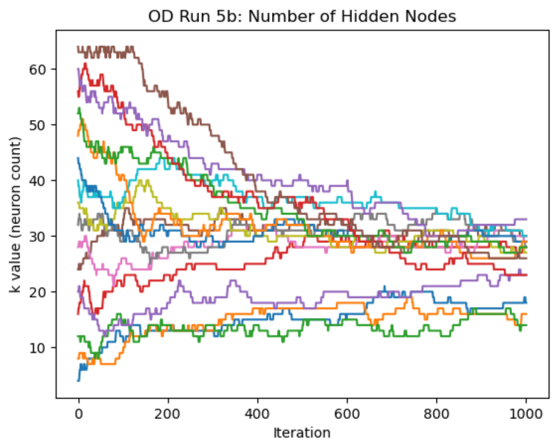
The number of hidden nodes and layers in a neural network are both discrete parameters. A MAP estimate for such a parameter will therefore be an integer value (the mode of the parameter samples) that directly corresponds to an optimal parameter value. On the other hand, expectations of discrete parameters are weighted averages of parameter samples, and need not be discrete. Table 7.3 displays these metrics for the RJBNN experiments.

Network	**Parameter**	**Mode (MAP)**	**Expectation**
OD 5a	$\mathcal{K}$	48	43.35
OD 5b	$\mathcal{K}$	30	28.73
OD 10a	$\mathcal{K}$	89	81.34
OD 10b	$\mathcal{K}$	55	60.31
BH 2a	$\mathcal{L}$	2	3.74
BH 2b	$\mathcal{L}$	4	3.73
BH 4a	$\mathcal{L}$	3	2.67
BH 4b	$\mathcal{L}$	3	3.07

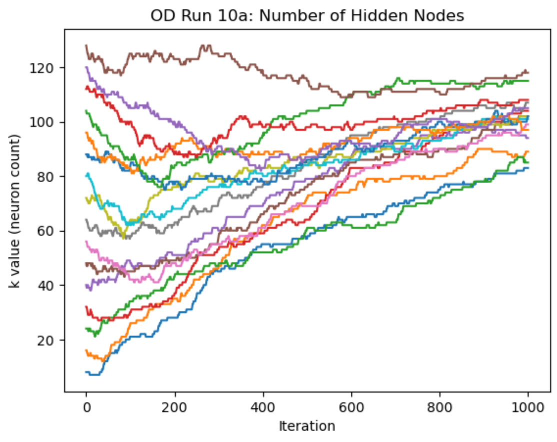
TABLE 7.3: MAP estimates and expectations for network architectures



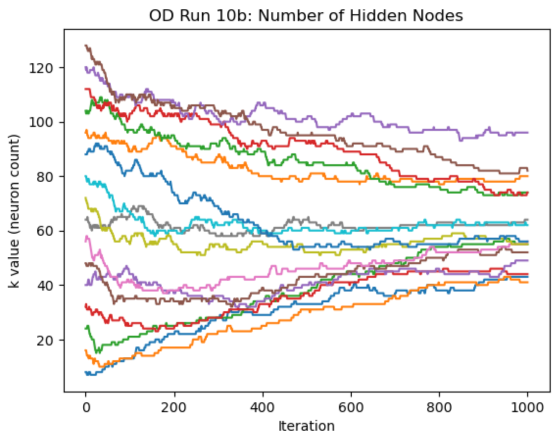
(A) Experiment 5a



(B) Experiment 5b

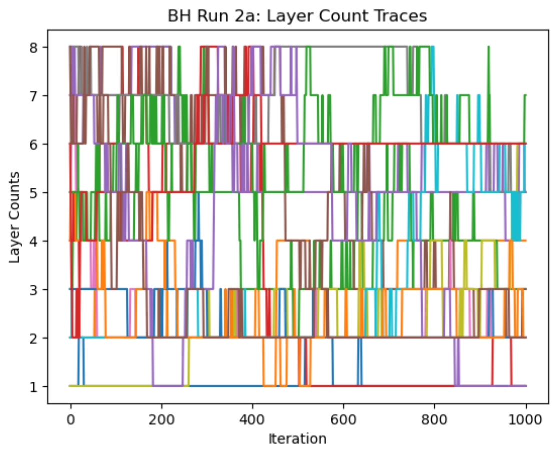


(C) Experiment 10a

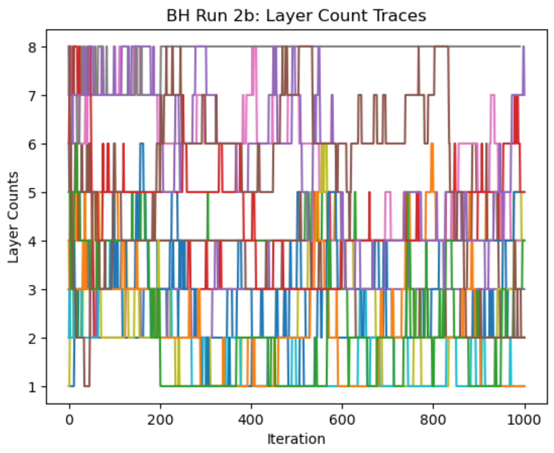


(D) Experiment 10b

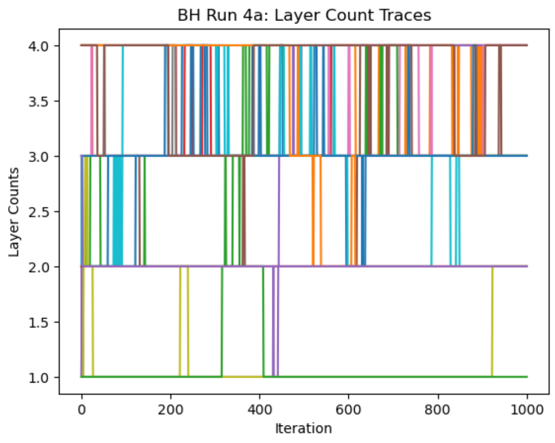
FIGURE 7.5: Node traces: network width by iteration for the 16 chains of the RJBNN sampling program



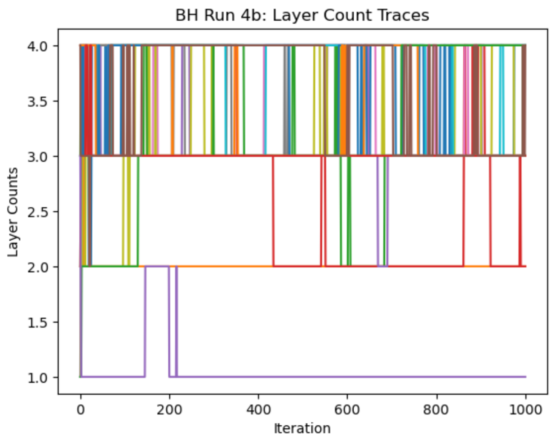
(A) Experiment 2a



(B) Experiment 2b



(C) Experiment 4a



(D) Experiment 4b

FIGURE 7.6: Layer traces: network depth by iteration for the 16 chains of the RJBNN sampling program

7.2.1 Occam's Razor

In his pioneering work on BNNs, David Mackay points out that the Bayesian approach automatically implies Occam's razor [41] - the premise that the simplest explanation that adequately explains the situation should be preferred to a more complicated one. The statistical modelling perspective interpretation of this idea is that a model capable of fitting data with fewer parameters should be preferred to some degree when compared to one with more parameters or a more complex structure.

Does the mode imply optimal architecture?

The XOR experiment in section 5.5 provided evidence against our hypothesis that the RJBNN would favour a minimal viable architecture. The mode of the number of hidden neurons was 7, which far exceeds the minimum requirement of 2. Networks with 2 hidden neurons were sampled infrequently in this experiment, suggesting additional capacity was favoured by the model definition and RJNUTS sampler.

We may also wonder whether the modal network architecture might correspond to better predictive accuracy. Table 7.4 presents evidence to the contrary. For each experiment on the Boston Housing dataset, the marginal predictive test RMSE over the network samples corresponding to the modal network architecture was higher than that of the best individual chain of samples across multiple network architectures.

Network	Marginal Test RMSE	Best Chain RMSE	Layer Mode	Layer Mode RMSE
RJBNN 2A	0.236	**0.235**	2	0.232
RJBNN 2B	0.245	0.249	4	**0.238**
RJBNN 4A	0.223	**0.211**	3	0.224
RJBNN 4B	0.215	**0.209**	3	0.215

TABLE 7.4: Best RMSE by Architecture Marginalization

Such a result suggests the RJBNN is not necessarily providing clues as to which architecture would fare best as a fixed-architecture BNN or classical ANN model. However, we seem to get an improvement to predictive accuracy by marginalizing over this parameter - a satisfying result for our trans-dimensional inference approach to BNN models.

Ultimately, our experiments did not find evidence that the RJBNN may guide the design of optimal architectures for equivalent fixed ANN model architectures. It is for this reason that we have not presented the RJBNN model from the perspective of NAS. Questions as to how the interplay of the model definition and the RJNUTS inference algorithm lead to the resultant approximate marginal posterior distribution of architectures is therefore a potential future research direction, which brings us to a discussion of other opportunities for continued research in this area.

7.3 Future Research Directions

The RJBNN has been proposed as a theoretically sound approach to robust inference for classification and interpolation tasks. Our results present evidence that the RJBNN model can be effectively implemented using RJNUTS inference for appropriately-sized networks. The question of scalability to modern DL applications, however, remains open. The complexities in designing a computationally sound RJBNN model for larger datasets are steep.

Along with these complexities comes the opportunity for future research directions. Several aspects of the design of RJBNN models and the RJNUTS algorithm may be examined for improved computational efficiency and model tuning. Several of these opportunities are highlighted in this section.

7.3.1 Proposal Design for RJBNNs

The computational complexity of the algorithm used for inference of RJBNN models is due almost entirely to the use of HMC via NUTS as a delayed rejection sampler. The motivation for this approach was discussed in section 5.3, in that we were not able in this work to design an across-dimension proposal that would result in a tenable acceptance rate for jumps to new network architectures. To design such an improved proposal and thereby reduce or remove the dependency on gradient-based delayed rejection sampling would dramatically improve the computational efficiency of trans-dimensional inference for BNNs.

Applying effort to RJBNN across-dimension proposal design presents a challenge, but also provides an opportunity to better understand neural networks. The functional relationship between two neural network architectures with differing numbers of hidden layers or neurons within those layers requires a deep analysis of the behaviour of the functional expressiveness of the networks. In metaphor, we would need to open the "black box" and better understand what makes for a well-specified network based on the parameterizations as opposed to prediction-based metrics on a testing set. This is a somewhat cyclical motivation: we need to understand neural networks better to design better RJBNNs, which in turn have been proposed here as an approach to help us understand and therefore design better ANNs.

A good starting point for this line of work would be to further investigate the recent attention to network morphisms [74] to see whether proposals for larger or smaller neural network architectures could be designed in the spirit of (approximately) preserving the functional representation of the state of the current architecture.

7.3.2 Speeding up RJNUTS Computation

The experiments presented in this thesis would not have been possible without HMC. The NUTS algorithm was chosen to ease the implementation of HMC by abstracting specification of the HMC hyperparameters away from the user. There is still room for improvement in how the HMC/NUTS components of the RJNUTS algorithm were implemented.

For one, datasets for the experiments were selected based on a size restriction. The Boston Housing training dataset contained 253 observations with 13 attributes. OptDigits was selected over the similar MNIST based on a more convenient feature size. A small training subset of 250 or 500 samples was used for the OptDigits training, to which PCA was applied to reduce the features from 64 to 20.

Favouring small datasets is not uncommon to Bayesian analyses. Each iteration of any MCMC approach requires an assessment of the full dataset to compute the likelihood as part of the posterior score. This limitation of MCMC continues to attract research attention, which has resulted in a number of approaches to "mini-batch" MCMC [88] or subsampling MCMC [89]. Such an approach allows a random subset of the full dataset to be examined at each iteration, with necessary adjustments to the acceptance probability and posterior distribution to maintain detailed balance and account for introduced bias.

Mini-batch MCMC was beyond the scope of the experiments presented here, but may be one important key to rendering RJBNNs more practically applicable to larger datasets.

7.3.3 Hyperparameter Selection of Network Parameters and the Likelihood

The hyperparameter schedule for the RJBNN models examined were proposed based on conventionality and convenience. There is therefore an opportunity to experiment more with the model indicators in the RJBNN model definition - the k and ℓ parameters which control the network architectures. We also used flat priors and relied on Occam's razor to favour the smallest sufficient neural network as an introduction to running inference of trans-dimensional BNNs, but more meaningful prior distributions could be proposed, with attention to improving the applicability of RJNUTS to detailed hierarchical hyperparameter schedules.

We also briefly mention the opportunity to explore approximate likelihoods [90] in running BNN inference. This thesis focused on the supervised learning tasks of classification and regression, which imply convenient likelihood definitions with only moderate flexibility. The regression network features a hyperparameter controlling the noise precision, and we relaxed the softmax with a tempering parameter (equation 6.1) for similar flexibility in the classification case, but approximate likelihoods may afford more flexibility for improved RJNUTS inference behaviour [36].

Approximate likelihoods would also represent a necessary feature for exploring the application of RJBNNs to unsupervised learning, including generative network models.

7.4 Summary of Contributions

We claim three contributions with regards to trans-dimensional inference for BNNs.

1. We have demonstrated the application of RJMCMC to BNNs for estimating the marginal posterior distribution over a range of network depths
2. We have demonstrated the application of RJMCMC to BNNs for estimating the marginal posterior distribution over a range of network widths
3. We have demonstrated the integration of the NUTS algorithm to improve the acceptance ratio of RJMCMC across-dimension proposals using a delayed-rejection sampler scheme

7.5 Conclusion

The work presented in this thesis may be viewed from the perspective of automation. We began with a description of statistical model selection as the basis for justifying the derivation of trans-dimensional inference via RJBNN models. Machine learning has been presented as an automated approach to modelling, and the Bayesian approach of inference as learning was reviewed to further extend this automation to the development of probability distributions over candidate model space. BNNs were the model of choice given known results about the functional generalizability and overwhelming popularity of deep ANN models, and have been extended to multi-dimensional models such that NN architecture need not be directly specified as part of the model design process.

The results demonstrate the theoretical possibility for the use of RJBNNs in supervised learning, but caveats regarding computational feasibility and practical scope exist. The experiments conducted in this thesis are indeed quaint, focusing on small networks and small datasets. Here we may draw a comparison to the timeline leading up to the deep learning revolution, where early models offered promising theory despite the practical intractability of the proposed architectures. Increases in computational capacity and innovations in ML theory eventually led to the deployment of DNNs in numerous academic and industrial applications, to the point that virtually all users of modern technology benefit in some regard from the research efforts made in DL.

This is to not suggest that RJBNNs should be a preferred approach to classification or regression tasks where predictive performance or uncertainty measures are the desired products of the model. The relevance of an RJBNN approach to a discussion of automation is simply the ability to abstract the exact architectural specification of a neural network design away from the practitioner, in situations where it may be important to formally express prior uncertainty regarding what depth or width of network may be required for the modelling task.

We hope that RJBNNs and similar approaches in AutoML and NAS may continue to be developed as part of the general trajectory towards automation, such that we may gain a better understanding of the ML methods in use today, and further develop new methods for insight into the increasing wealth of data available in the information age.

Appendix A

Datasets

XOR Point Clouds

Description: 2-dimensional, non-linearly separable, distinct point clouds with 2 class labels

Task: Binary classification

Source: Data is custom-generated for this experiment

Features: 2 - x/y coordinate space

Transformation Details: N/A

Number of Training Samples: 200

Number of Test Samples: 200

Noisy XOR Point Clouds

Description: 2-dimensional, non-linearly separable, overlapping point clouds with 2 class labels

Task: Binary classification

Source: Data is custom-generated for this experiment

Number of Features: 2 - x/y coordinate space

Transformation Details: N/A

Number of Training Samples: 200

Number of Test Samples: 200

OptDigits

Task: Multi-class classification (5 or 10 classes)

Description: Images of black and white handwritten digits 0-9 with values ranging from 0-255 for each pixel

Source: `https://archive.ics.uci.edu/ml/datasets.php`

Number of Features: 8 x 8 = 64

Transformation Details: Pixel values are normalized to values ranging between 0 and 1. PCA transformation is applied to reduce the number of features from 64 to 20.

Number of Training Samples:

- Experiments 5a, 5b - 250 (50 for each class 0-4)
- Experiments 10a, 10b - 500 (50 for each class 0-9)

Number of Test Samples: Number of Training Samples:

- Experiments 5a, 5b - 1000 (200 for each class 0-4)
- Experiments 10a, 10b - 2000 (200 for each class 0-9)

Boston Housing

Task: Regression - median housing price

Description: A dataset of census data for neighbourhoods in Boston, Massachusetts.

Source: `https://www.cs.toronto.edu/~delve/data/boston/bostonDetail.html`

Number of Features: 14 - 13 explanatory variables, and median housing price

Transformation Details: Each feature is standardized and normalized with mean 0 and variance 1.

Number of Training Samples: 256

Number of Test Samples: 256

Appendix B

Technical Implementation Details

Overview

All of the experiments for this thesis were implemented in the Julia programming language [91]. The probabilistic programming language Gen [92] was used to construct the RJBNN models. The NUTS algorithm was implemented based on code from the Turing probabilistic programming language [93].

Computing

High-performance computing resources for the experiments were provided courtesy of UNINETT Sigma2 AS (Sigma2).

All experiments were run on the *Saga* supercomputer. Memory allocations for the experiments were as follows:

- Boston Housing RJBNN: 1 node, 16 CPUs, 4GB RAM / CPU
  Training time: ~ 168 hours per experiment
- OptDigits RJBNN: 1 node, 16 CPUs, 4GB RAM / CPU
  Training time: $24 - 48$ hours per experiment
- XOR and Noisy XOR RJBNN: 1 node, 16 CPUs, 512MB RAM / CPU
  Training time: < 1 hour per experiment

Code

GitHub repositories are made available for the experiments conducted in this thesis:

Boston Housing experiments: `https://github.com/jberezow/BostonHousing`

OptDigits and XOR experiments: `https://github.com/jberezow/OptDigits`

Bibliography

[1] Peter J Green. “Trans-dimensional markov chain monte carlo”. In: *Oxford Statistical Science Series* (2003), pp. 179–198.

[2] David Donoho. “50 Years of Data Science”. In: *Journal of Computational and Graphical Statistics* 26 (Oct. 2017), pp. 745–766. DOI: 10.1080/10618600.2017.1384734.

[3] Yarin Gal. “Uncertainty in Deep Learning”. PhD thesis. University of Cambridge, 2016.

[4] Di Feng, L. Rosenbaum, and K. Dietmayer. “Towards Safe Autonomous Driving: Capture Uncertainty in the Deep Neural Network For Lidar 3D Vehicle Detection”. In: *2018 21st International Conference on Intelligent Transportation Systems (ITSC)* (2018). DOI: 10.1109/ITSC.2018.8569814.

[5] Terrence J. Sejnowski. “The unreasonable effectiveness of deep learning in artificial intelligence”. en. In: *Proceedings of the National Academy of Sciences* 117.48 (Dec. 2020). Publisher: National Academy of Sciences Section: Colloquium Paper, pp. 30033–30038. ISSN: 0027-8424, 1091-6490. DOI: 10.1073/pnas.1907373117. URL: https://www.pnas.org/content/117/48/30033 (visited on 05/26/2021).

[6] Steve Lawrence, C. Giles, and Ah Tsoi. “What Size Neural Network Gives Optimal Generalization? Convergence Properties of Backpropagation”. In: (Mar. 2001).

[7] A. C. Davison. *Statistical Models*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge: Cambridge University Press, 2003. ISBN: 978-0-521-77339-3. DOI: 10.1017/CBO9780511815850. URL: https://www.cambridge.org/core/books/statistical-models/8EC19F80551F52D4C58FAA2022048FC7 (visited on 05/26/2021).

[8] Herman J Adèr, G. J Mellenbergh, and D. J Hand. *Advising on research methods: a consultant's companion*. English. OCLC: 213401217. Huizen, Netherlands: Johannes van Kessel Pub., 2008. ISBN: 978-90-79418-01-5 978-90-79418-02-2.

[9] George Casella and Roger Berger. *Statistical Inference*. Duxbury Resource Center, June 2001. ISBN: 0534243126.

[10] R. A. Fisher. “Theory of Statistical Estimation”. In: *Mathematical Proceedings of the Cambridge Philosophical Society* 22.5 (1925), pp. 700–725. ISSN: 1469-8064. DOI: 10.1017/S0305004100009580. URL: https://www.cambridge.org/core/article/theory-of-statistical-estimation/7A05FB68C83B36C0E91D42C76AB177D4.

[11] Vladimir N. Vapnik. *Statistical Learning Theory*. Wiley-Interscience, 1998.

[12] Jie Ding, Vahid Tarokh, and Yuhong Yang. "Model Selection Techniques: An Overview". In: *IEEE Signal Processing Magazine* 35.6 (Nov. 2018). Number: 6 Publisher: IEEE, pp. 16–34. ISSN: 1053-5888. URL: https://resolver.caltech.edu/CaltechAUTHORS:20181128-150927005 (visited on 01/26/2021).

[13] Danilo Bzdok, Naomi Altman, and Martin Krzywinski. *Points of significance: statistics versus machine learning*. 2018.

[14] Ethem Alpaydin. *Introduction to Machine Learning*. The MIT Press, 2014. ISBN: 0262028182.

[15] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 3rd. USA: Prentice Hall Press, 2009. ISBN: 0136042597.

[16] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. "Overview of supervised learning". In: *The elements of statistical learning*. Springer, 2009, pp. 9–41.

[17] J. M. Benitez, J. L. Castro, and I. Requena. "Are artificial neural networks black boxes?" In: *IEEE Transactions on Neural Networks* 8.5 (1997), pp. 1156–1164. DOI: 10.1109/72.623216.

[18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". en. In: *Advances in Neural Information Processing Systems* 25 (2012), pp. 1097–1105. URL: https://proceedings.neurips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html (visited on 01/26/2021).

[19] Ashish Vaswani et al. "Attention is All you Need". In: *NIPS* (2017).

[20] Jonathan Long, Evan Shelhamer, and Trevor Darrell. "Fully Convolutional Networks for Semantic Segmentation". In: *CoRR* abs/1411.4038 (2014). arXiv: 1411.4038. URL: http://arxiv.org/abs/1411.4038.

[21] Kurt Hornik. "Approximation capabilities of multilayer feedforward networks". en. In: *Neural Networks* 4.2 (Jan. 1991), pp. 251–257. ISSN: 0893-6080. DOI: 10.1016/0893-6080(91)90009-T. URL: https://www.sciencedirect.com/science/article/pii/089360809190009T (visited on 03/02/2021).

[22] F. Rosenblatt. "The perceptron: A probabilistic model for information storage and organization in the brain." In: *Psychological Review* 65.6 (1958), pp. 386–408. ISSN: 0033-295X. DOI: 10.1037/h0042519. URL: http://dx.doi.org/10.1037/h0042519.

[23] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[24] Vinod Nair and Geoffrey Hinton. "Rectified Linear Units Improve Restricted Boltzmann Machines Vinod Nair". In: vol. 27. June 2010, pp. 807–814.

[25] Karl Pearson F.R.S. "LIII. On lines and planes of closest fit to systems of points in space". In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2.11 (1901), pp. 559–572. DOI: 10.1080/14786440109462720.

[26] Thomas Bayes and null Price. "An essay towards solving a problem in the doctrine of chances." In: *Philosophical Transactions of the Royal Society of London* 53 (Jan. 1763). Publisher: Royal Society, pp. 370–418. DOI: 10.1098/rstl.1763.0053. URL: https://royalsocietypublishing.org/doi/10.1098/rstl.1763.0053 (visited on 05/26/2021).

[27] Harold Jeffreys. *Theory of probability / by Harold Jeffreys*. English. 3rd ed. Clarendon Press Oxford, 1961, viii, 447 p. :

[28] Herbert Lee. "Model Selection for Neural Network Classification". In: *J. Classification* 18 (Feb. 2001), pp. 227–243. DOI: 10.1007/s00357-001-0017-y.

[29] Radford M. Neal. *Bayesian Learning for Neural Networks*. en. Ed. by P. Bickel et al. Vol. 118. Lecture Notes in Statistics. New York, NY: Springer New York, 1996. DOI: 10.1007/978-1-4612-0745-0. URL: http://link.springer.com/10.1007/978-1-4612-0745-0 (visited on 12/27/2020).

[30] Michael Betancourt. *Towards A Principled Bayesian Workflow*. Apr. 2020. URL: https://betanalpha.github.io/assets/case_studies/principled_bayesian_workflow.html#12_Computational_Faithfulness.

[31] Andrew Gelman. "Prior distributions for variance parameters in hierarchical models". In: *Bayesian Analysis* 1.3 (Sept. 2006). Publisher: International Society for Bayesian Analysis, pp. 515–534. ISSN: 1936-0975, 1931-6690. DOI: 10.1214/06-BA117A. URL: https://projecteuclid.org/journals/bayesian-analysis/volume-1/issue-3/Prior-distributions-for-variance-parameters-in-hierarchical-models-comment-on/10.1214/06-BA117A.full (visited on 05/27/2021).

[32] Robert E. Kass and Larry Wasserman. "The Selection of Prior Distributions by Formal Rules". In: *Journal of the American Statistical Association* 91.435 (Sept. 1996). Publisher: Taylor & Francis _eprint: https://www.tandfonline.com/doi/pdf/10.1080/01621459.1996.104 pp. 1343–1370. ISSN: 0162-1459. DOI: 10.1080/01621459.1996.10477003. URL: https://www.tandfonline.com/doi/abs/10.1080/01621459.1996.10477003 (visited on 05/27/2021).

[33] Gunnar Taraldsen and Bo Henry Lindqvist. "Improper Priors Are Not Improper". In: *The American Statistician* 64.2 (May 2010). Publisher: Taylor & Francis, pp. 154–158. ISSN: 0003-1305. DOI: 10.1198/tast.2010.09116. URL: https://amstat.tandfonline.com/doi/abs/10.1198/tast.2010.09116 (visited on 05/27/2021).

[34] Paul C. Lambert et al. "How vague is vague? A simulation study of the impact of the use of vague prior distributions in MCMC using WinBUGS". en. In: *Statistics in Medicine* 24.15 (2005). _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/sim.2112, pp. 2401–2428. ISSN: 1097-0258. DOI: https://doi.org/10.1002/sim.2112. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/sim.2112 (visited on 05/27/2021).

[35] Herbert K.H. Lee. “Default Priors for Neural Network Classification”. en. In: *Journal of Classification* 24.1 (June 2007), pp. 53–70. ISSN: 1432-1343. DOI: 10.1007/s00357-007-0001-2. URL: https://doi.org/10.1007/s00357-007-0001-2 (visited on 05/27/2021).

[36] Andrew Gelman, Daniel Simpson, and Michael Betancourt. “The Prior Can Often Only Be Understood in the Context of the Likelihood”. en. In: *Entropy* 19.10 (Oct. 2017). Number: 10 Publisher: Multidisciplinary Digital Publishing Institute, p. 555. DOI: 10.3390/e19100555. URL: https://www.mdpi.com/1099-4300/19/10/555 (visited on 05/27/2021).

[37] Tim Pearce, Felix Leibfried, and Alexandra Brintrup. “Uncertainty in Neural Networks: Approximately Bayesian Ensembling”. en. In: *International Conference on Artificial Intelligence and Statistics*. ISSN: 2640-3498. PMLR, June 2020, pp. 234–244. URL: http://proceedings.mlr.press/v108/pearce20a.html (visited on 03/02/2021).

[38] M. J. Bayarri and J. O. Berger. “The Interplay of Bayesian and Frequentist Analysis”. In: *Statistical Science* 19.1 (2004). Publisher: Institute of Mathematical Statistics, pp. 58–80. ISSN: 0883-4237. URL: https://www.jstor.org/stable/4144373 (visited on 05/27/2021).

[39] R. A. Fisher. “Inverse Probability”. en. In: *Mathematical Proceedings of the Cambridge Philosophical Society* 26.4 (Oct. 1930). Publisher: Cambridge University Press, pp. 528–535. ISSN: 1469-8064, 0305-0041. DOI: 10.1017/S0305004100016297. URL: https://www.cambridge.org/core/journals/mathematical-proceedings-of-the-cambridge-philosophical-society/article/inverse-probability/C9AB0A7C4566A3F9FCCEC489CA854814 (visited on 05/27/2021).

[40] Andrew Gelman and Jennifer Hill. *Data Analysis Using Regression and Multilevel/Hierarchical Models*. Analytical Methods for Social Research. Cambridge: Cambridge University Press, 2006. ISBN: 9780521867061. DOI: 10.1017/CBO9780511790942. URL: https://www.cambridge.org/core/books/data-analysis-using-regression-and-multilevelhierarchical-models/32A29531C7FD730C3A68951A17C9D983.

[41] David J. C. MacKay. “Bayesian Interpolation”. In: *NEURAL COMPUTATION* 4 (1991), pp. 415–447.

[42] Michael I. Jordan et al. “An Introduction to Variational Methods for Graphical Models”. en. In: *Machine Learning* 37.2 (Nov. 1999), pp. 183–233. ISSN: 1573-0565. DOI: 10.1023/A:1007665907178. URL: https://doi.org/10.1023/A:1007665907178 (visited on 03/02/2021).

[43] Matthew D. Hoffman et al. “Stochastic variational inference”. In: *ArXiv* abs/1206.7051 (2013).

[44] R. Wong. *Asymptotic Approximations of Integrals: Computer Science and Scientific Computing*. en. Google-Books-ID: EpHiBQAAQBAJ. Academic Press, May 2014. ISBN: 978-1-4832-2071-0.

[45] David J. C. MacKay. "A Practical Bayesian Framework for Backpropagation Networks". In: *Neural Computation* 4.3 (1992), pp. 448–472.

[46] Håvard Rue, Sara Martino, and Nicolas Chopin. "Approximate Bayesian inference for latent Gaussian models by using integrated nested Laplace approximations". en. In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 71.2 (2009). _eprint: https://rss.onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-9868.2008.00700.x, pp. 319–392. ISSN: 1467-9868. DOI: `https://doi.org/10.1111/j.1467-9868.2008.00700.x`. URL: `https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-9868.2008.00700.x` (visited on 03/03/2021).

[47] K. Javid et al. "Compromise-free Bayesian neural networks". In: *ArXiv* (2020).

[48] Thomas G. Dietterich. "Ensemble Methods in Machine Learning". In: *Multiple Classifier Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 1–15. ISBN: 978-3-540-45014-6.

[49] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. "Neural Architecture Search: A Survey". en. In: (), p. 21.

[50] Tom Charnock, Laurence Perreault-Levasseur, and François Lanusse. "Bayesian Neural Networks". In: *arXiv e-prints* 2006 (June 2020), arXiv:2006.01490. URL: `http://adsabs.harvard.edu/abs/2020arXiv200601490C` (visited on 03/02/2021).

[51] Anders Krogh and John A. Hertz. "A simple weight decay can improve generalization". In: *Proceedings of the 4th International Conference on Neural Information Processing Systems*. NIPS'91. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., Dec. 1991, pp. 950–957. ISBN: 978-1-55860-222-9. (Visited on 02/02/2021).

[52] Herbert K. H. Lee. "Priors for Neural Networks". en. In: *Classification, Clustering, and Data Mining Applications*. Ed. by David Banks et al. Studies in Classification, Data Analysis, and Knowledge Organisation. Berlin, Heidelberg: Springer, 2004, pp. 141–150. ISBN: 978-3-642-17103-1. DOI: `10.1007/978-3-642-17103-1_14`.

[53] David J. C. MacKay. "The Evidence Framework Applied to Classification Networks". In: *Neural Computation* 4.5 (1992), pp. 720–736. DOI: `10.1162/neco.1992.4.5.720`.

[54] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. USA: Oxford University Press, Inc., 1995. ISBN: 0198538642.

[55] Michael Betancourt. *A Conceptual Introduction to Hamiltonian Monte Carlo*. 2018. arXiv: `1701.02434 [stat.ME]`.

[56] Galin Jones. "Computational Statistics. Geof H. Givens and Jennifer A. Hoeting". In: *Journal of the American Statistical Association* 101 (Feb. 2006), pp. 856–857. DOI: `10.2307/27590758`.

[57] Steve Brooks et al. *Handbook of markov chain monte carlo*. CRC press, 2011.

[58] Geof H. Givens and Jennifer A. Hoeting. *Computational statistics*. 2nd ed. Hoboken, NJ, USA: John Wiley & Sons, 2012.

[59] R. Neal. "Non-reversibly updating a uniform [0, 1] value for Metropolis accept/reject decisions". In: *ArXiv* abs/2001.11950 (2020).

[60] Christopher Nemeth and Paul Fearnhead. "Stochastic Gradient Markov Chain Monte Carlo". In: *Journal of the American Statistical Association* 116.533 (Jan. 2021). Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/01621459.2020.1847120, pp. 433–450. ISSN: 0162-1459. DOI: 10.1080/01621459.2020.1847120. URL: https://doi.org/10.1080/01621459.2020.1847120 (visited on 05/28/2021).

[61] Nicholas Metropolis et al. "Equation of State Calculations by Fast Computing Machines". In: *The Journal of Chemical Physics* 21.6 (1953), pp. 1087–1092. DOI: 10.1063/1.1699114. eprint: https://doi.org/10.1063/1.1699114. URL: https://doi.org/10.1063/1.1699114.

[62] W. K. Hastings. "Monte Carlo sampling methods using Markov chains and their applications". In: *Biometrika* 57.1 (Apr. 1970), pp. 97–109. ISSN: 0006-3444. DOI: 10.1093/biomet/57.1.97. URL: https://doi.org/10.1093/biomet/57.1.97 (visited on 01/26/2021).

[63] S. Geman and D. Geman. "Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-6.6 (Nov. 1984). Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 721–741. ISSN: 1939-3539. DOI: 10.1109/TPAMI.1984.4767596.

[64] Peter J Green. "Reversible jump Markov chain Monte Carlo computation and Bayesian model determination". In: *Biometrika* 82.4 (1995), pp. 711–732.

[65] Peter J. Green and David I. Hastie. *Chapter 1 Reversible jump MCMC*. 2009.

[66] David I Hastie and Peter J Green. "Model choice using reversible jump Markov chain Monte Carlo". In: *Statistica Neerlandica* 66.3 (2012), pp. 309–338.

[67] Florian Wenzel et al. "How Good is the Bayes Posterior in Deep Neural Networks Really?" In: *arXiv:2002.02405 [cs, stat]* (July 2020). arXiv: 2002.02405. URL: http://arxiv.org/abs/2002.02405 (visited on 02/14/2021).

[68] Radford M Neal et al. "MCMC using Hamiltonian dynamics". In: *Handbook of markov chain monte carlo* 2.11 (2011), p. 2.

[69] S. Duane et al. "Hybrid Monte Carlo". In: *Physics Letters B* 195 (1987), pp. 216–222.

[70] Matthew D. Hoffman and Andrew Gelman. "The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo". In: *Journal of Machine Learning Research* 15.47 (2014), pp. 1593–1623. URL: http://jmlr.org/papers/v15/hoffman14a.html.

[71] Yurii Nesterov. “Primal-dual Subgradient Methods for Convex Problems”. In: *Mathematical Programming* 120 (Apr. 2009), pp. 221–259. DOI: 10.1007/s10107-007-0149-x.

[72] Alexandros Beskos et al. “Optimal tuning of the hybrid Monte Carlo algorithm”. In: *Bernoulli* 19.5A (Nov. 2013). Publisher: Bernoulli Society for Mathematical Statistics and Probability, pp. 1501–1534. ISSN: 1350-7265. DOI: 10.3150/12-BEJ414. URL: https://projecteuclid.org/journals/bernoulli/volume-19/issue-5A/Optimal-tuning-of-the-hybrid-Monte-Carlo-algorithm/10.3150/12-BEJ414.full (visited on 03/02/2021).

[73] Stan Development Team. *The Stan Core Library*. Version 2.18.0. 2018. URL: http://mc-stan.org/%201.

[74] Tao Wei et al. “Network Morphism”. en. In: *International Conference on Machine Learning*. ISSN: 1938-7228. PMLR, June 2016, pp. 564–572. URL: http://proceedings.mlr.press/v48/wei16.html (visited on 01/07/2021).

[75] Alireza Roodaki, Julien Bect, and Gilles Fleury. “Note on the computation of the Metropolis-Hastings ratio for Birth-or-Death moves in trans-dimensional MCMC algorithms for signal decomposition problems”. In: (Nov. 2011).

[76] Fahimah Al-Awadhi, Merrilee Hurn, and Christopher Jennison. “Improving the acceptance rate of reversible jump MCMC proposals”. In: *Statistics & Probability Letters* 69 (Aug. 2004), pp. 189–198. DOI: 10.1016/j.spl.2004.06.025.

[77] Sergios Theodoridis and Konstantinos Koutroumbas. *Pattern Recognition, Fourth Edition*. 4th. USA: Academic Press, Inc., 2008. ISBN: 1597492728.

[78] Mrinal Sen and Reetam Biswas. “Transdimensional seismic inversion using the reversible jump Hamiltonian Monte Carlo algorithm”. In: *GEOPHYSICS* 82 (May 2017), R119–R134. DOI: 10.1190/geo2016-0010.1.

[79] Mark Girolami and Ben Calderhead. “Riemann manifold Langevin and Hamiltonian Monte Carlo methods”. In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 73.2 (2011), pp. 123–214. DOI: https://doi.org/10.1111/j.1467-9868.2010.00765.x. eprint: https://rss.onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-9868.2010.00765.x. URL: https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-9868.2010.00765.x.

[80] J. Wyse, N. Friel, and M. Girolami. “Reversible jump Riemann Manifold Hamiltonian Monte Carlo”. In: 2012.

[81] Peter Müller and David Rios Insua. “Issues in Bayesian Analysis of Neural Network Models”. In: *Neural Computation* 10.3 (1998), pp. 749–770. DOI: 10.1162/089976698300017737. eprint: https://doi.org/10.1162/089976698300017737. URL: https://doi.org/10.1162/089976698300017737.

[82] Georgi Dikov and Justin Bayer. “Bayesian Learning of Neural Network Architectures”. en. In: *The 22nd International Conference on Artificial Intelligence and Statistics*.

ISSN: 2640-3498. PMLR, Apr. 2019, pp. 730–738. URL: http://proceedings.mlr.press/v89/dikov19a.html (visited on 05/31/2021).

[83] Aliaksandr Hubin and Geir Storvik. "Combining Model and Parameter Uncertainty in Bayesian Neural Networks". In: *arXiv:1903.07594 [cs, math, stat]* (May 2019). arXiv: 1903.07594. URL: http://arxiv.org/abs/1903.07594 (visited on 06/01/2021).

[84] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository*. 2017. URL: http://archive.ics.uci.edu/ml.

[85] Yann LeCun and Corinna Cortes. "MNIST handwritten digit database". In: (2010). URL: http://yann.lecun.com/exdb/mnist/.

[86] David Harrison and Daniel Rubinfeld. "Hedonic housing prices and the demand for clean air". In: *Journal of Environmental Economics and Management* 5 (Mar. 1978), pp. 81–102. DOI: 10.1016/0095-0696(78)90006-2.

[87] Andrew Gordon Wilson. *The Case for Bayesian Deep Learning*. 2020. arXiv: 2001.10995 [cs.LG].

[88] Tung-Yu Wu, Y. X. Rachel Wang, and Wing H. Wong. "Mini-batch Metropolis-Hastings MCMC with Reversible SGLD Proposal". In: *arXiv:1908.02910 [cs, stat]* (Aug. 2019). arXiv: 1908.02910. URL: http://arxiv.org/abs/1908.02910 (visited on 12/21/2020).

[89] Matias Quiroz et al. "Speeding Up MCMC by Efficient Data Subsampling". In: *Journal of the American Statistical Association* 114.526 (Apr. 2019). Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/01621459.2018.1448827, pp. 831–843. ISSN: 0162-1459. DOI: 10.1080/01621459.2018.1448827. URL: https://doi.org/10.1080/01621459.2018.1448827 (visited on 12/21/2020).

[90] Bradley Efron and Robert J. Tibshirani. *An Introduction to the Bootstrap*. Monographs on Statistics and Applied Probability 57. Boca Raton, Florida, USA: Chapman & Hall/CRC, 1993.

[91] Jeff Bezanson et al. "Julia: A Fast Dynamic Language for Technical Computing". In: *CoRR* abs/1209.5145 (2012). arXiv: 1209.5145. URL: http://arxiv.org/abs/1209.5145.

[92] Marco F. Cusumano-Towner et al. "Gen: A General-purpose Probabilistic Programming System with Programmable Inference". In: *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI 2019. Phoenix, AZ, USA: ACM, 2019, pp. 221–236. ISBN: 978-1-4503-6712-7. DOI: 10.1145/3314221.3314642. URL: http://doi.acm.org/10.1145/3314221.3314642.

[93] Hong Ge, Kai Xu, and Zoubin Ghahramani. "Turing: A Language for Flexible Probabilistic Inference". In: *Proceedings of the Twenty-First International Conference*

on Artificial Intelligence and Statistics. Ed. by Amos Storkey and Fernando Perez-Cruz. Vol. 84. Proceedings of Machine Learning Research. PMLR, Sept. 2018, pp. 1682–1690. URL: http://proceedings.mlr.press/v84/ge18b.html.