Priors and Algorithms for Bayesian Neural Networks

Charalambos Stavrou

Department of Computer Science
University College London

Supervised by:
Dr Ricardo Silva

This report is submitted to the University College London in partial fulfilment of the requirements for the degree of MSc in Computational Statistics and Machine Learning (CSML). It is substantially the result of my own work except where explicitly indicated in the text. The report may be freely copied and distributed provided the source is explicitly acknowledged.

September 2017

Abstract

We investigate how priors on the neural network (NN) parameters impact the model through the activation function. Specifically, how "sparsity" inducing priors such as Laplace, and Student T distribution priors differ from Normal priors in their impact on the neural network and ultimately the algorithm's performance. We review regularisation methods and dropout, but also provide a possible Bayesian interpretation of dropout a as variant of a mixture model. We examine the practicality of Bayesian neural networks (BNNs) by applying the algorithms to both classification (MNIST) and regression (variant of Mauna Loa) type of problems. The BNNs are then compared to feed-forward NNs fitted using back propagation with a variety of different optimisers. For approximating the BNN parameters, experiments have been conducted using, Hamiltonian (Hybrid) Monte Carlo (HMC), Stochastic Gradient Hamiltonian Monte Carlo (SGHMC), and Variational Inference (VI), in particular using the Variational Mean Field (VMF) approximation. In our analysis we explore BNNs of 1 hidden layer, 2 hidden layers and also a Bayesian convolutional NN (BCNN).

Keywords: *Bayesian neural networks, neural networks, Bayesian convolutional neural networks, Sparsity priors, HMC, SGHMC, Variational inference, VMF, Dropout.*

Acknowledgements

Firstly, I would like to thank my supervisor Dr Ricardo Silva, for all our helpful discussions and his guidance during this project which aided my understanding of the concepts that will be discussed. Finally, I would like to thank my family for their incredible support during my studies, without whom I would not have otherwise ventured into this Master's degree.

Contents

Notation

For the purposes of the study the notation will remain consistent. The notation might differ slightly in parts of the study, however this will be obvious from the context.

$\mathbf{C}$	matrix
$\mathbf{c}$	vector
c	scalar
$\mathbf{W}$	weight matrix
$\mathbf{H}$	hidden layer matrix
$\mathbf{b}$	bias vector
$\mathbf{X}$	input matrix (n rows, d columns)
$\mathbf{Y}$	matrix of true response/labels (n rows, k columns)
$\hat{\mathbf{Y}}$	output matrix (n rows, k columns)
$\mathbf{x_i}$	single input data point (vector of length d)
$\hat{\mathbf{y}}_\mathbf{i}$	single output point (vector of length k)
$\mathbf{y_i}$	single true response point (vector of length k)
$\mathcal{D}$	Dataset $\{\mathbf{X}, \mathbf{Y}\}$
$\mathbf{I_N}$	NxN identity matrix ($\forall i, j \in \{1, 2, \ldots, N\} : i_{ij} = 1$ if $i = j$, 0 if $i \neq j$)

Abbreviations

NN	Neural network
BNN	Bayesian neural network
CNN	Convolutional neural network
BCNN	Bayesian convolutional neural network
GP	Gaussian process
MC	Monte Carlo
MCMC	Markov chain Monte Carlo
VI	Variational inference
KL	Kullback-Leibler
SGD	Stochastic gradient descent
w.r.t.	With respect to
i.i.d.	Independent and identically distributed
s.t.	Such that
$\forall$	'For any' or 'for all'
"vice versa"	The other way around

Operators and functions[1]

$:=$	Defined as
$\Rightarrow$	Implies
$x \Leftrightarrow y$	x is equivalent to y
$\log(x)$	Natural logarithm of x
$\mathbf{X^T}$	The transpose of matrix $\mathbf{X}$
$\mathbf{b} \odot \mathbf{b}$	Elementwise product of the two matrices/vectors $\mathbf{b}$
$\nabla_{\mathbf{x}} \Leftrightarrow \frac{d}{d\mathbf{x}}$	Gradient vector w.r.t. $\mathbf{x}$
$\frac{\partial}{\partial \mathbf{x}}$	Vector of partial derivatives w.r.t. $\mathbf{x}$
$\mathbf{0}$	A matrix or vector whose entries are all equal to zero
$\mathbf{1}$	A matrix or vector whose entries are all equal to one
$\mathbf{x}_{\setminus \mathbf{i}} \Leftrightarrow \mathbf{x} \setminus x_i$	$(x_1, x_2, \ldots, x_{i-1}, x_{i+1}, \ldots, x_d)$ a vector with i^{th} entry removed
$e^{\mathbf{z}}$	Element-wise exponentiation of vector $\mathbf{z}$
$\text{relu}(\mathbf{Z})$	$\max(\mathbf{0}, \mathbf{Z})$, for a matrix $\mathbf{Z}$ the maximum is applied element-wise
$\text{softplus}(\mathbf{Z})$	$\log(\mathbf{1} + e^{\mathbf{Z}})$, for a matrix $\mathbf{Z}$ the logarithm applied element-wise
$\tanh(x)$	$\frac{\sinh(x)}{\cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$, which is the hyperbolic tangent
$\tanh(\mathbf{X})$	Hyperbolic tangent applied element-wise
$\text{softmax}(\mathbf{z})$	$\frac{\mathbf{e^z}}{\sum_c e^{z_c}}$, for a vector $\mathbf{z}$
$\text{softmax}(\mathbf{Z})$	$\begin{pmatrix} \mathbf{e^{z_1}} / \sum_c e^{z_{1c}} \\ \vdots \\ \mathbf{e^{z_n}} / \sum_c e^{z_{nc}} \end{pmatrix}$, for a matrix $\mathbf{Z}$ the function is applied to each row
$\mathbb{E}[x]_y \Leftrightarrow \langle x \rangle_y$	Expectation of x under the distribution y, i.e. under $p(y)$
$\lVert \mathbf{x} \rVert_1$	$\sum_{i=1}^n \lvert x_i \rvert$, for a vector $\mathbf{x}$ with length n
$\lVert \mathbf{x} \rVert_2^2$	$\sum_{i=1}^n x_i^2$, for a vector $\mathbf{x}$ with length n
$\Gamma(n)$	$(n-1)!$, for $n \in \mathbb{N}_{>0}$
$\Gamma(x)$	$\int_0^\infty z^{x-1} e^{-z} dz$, for $x \in \mathbb{R}_{>0}$
$\text{diag}(\mathbf{M})$	$(m_{11}, m_{22}, \cdots, m_{dd})$, for a (dxd) matrix $\mathbf{M}$
$\text{diag}(\mathbf{m})$	$\begin{pmatrix} m_{11} & 0 & \cdots & 0 \\ 0 & m_{22} & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \cdots & 0 & m_{dd} \end{pmatrix}$, for a vector $\mathbf{m}$ of length d

[1]Note that through the study there is slight abuse this gradient vector and vector of partial derivatives notation to mean also derivative w.r.t a matrix and matrix of partial derivatives w.r.t. a matrix.

$$\delta_{ij} = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{if } i \neq j \end{cases} \qquad \mathbb{1}_{\{\text{condition}\}} = \begin{cases} 1, & \text{if condition is TRUE} \\ 0, & \text{if condition is FALSE} \end{cases}$$

Distributions

$$\begin{aligned} \mathcal{N}_x(\mu, \sigma^2) &= \tfrac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} && \text{(Normal/Gaussian)} \\ \mathcal{N}_{\mathbf{x}}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) &= |2\pi\boldsymbol{\Sigma}|^{-\frac{1}{2}} \exp\left(-\tfrac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right) && \text{(Multivariate Normal/Gaussian)} \end{aligned}$$

The covariance matrix $\boldsymbol{\Sigma}$ is positive semi-definite, and the $|.|$ operator above refers to the determinant. If in the text we say that the weights w_i are "independent", then we mean that $\boldsymbol{\Sigma}$ has a diagonal structure, s.t. each of the w_i is sampled independently from a $\mathcal{N}_{w_i}(\mu_i, \sigma_{ii}^2)$, where σ_{ii}^2 are the diagonal entries of $\boldsymbol{\Sigma}$.

$$\begin{aligned} \text{Laplace}(x|\mu, b) &= \tfrac{1}{2b} e^{-\frac{|x-\mu|}{b}} && \text{(Laplace Distribution)} \\ \text{Laplace}(\mathbf{x}|\boldsymbol{\mu}, \mathbf{B}) &= \textstyle\prod_{i=1}^{d} \tfrac{1}{2b_{ii}} e^{-\frac{|x_i-\mu_i|}{b_{ii}}} && \text{(Multivariate Laplace)} \end{aligned}$$

For the purposes of this dissertation the Multivariate Laplace distribution we define possess independence across the samples x_i, and therefore $\mathbf{B}$ will always be diagonal. Notice also that $|.|$ above refers to the absolute value function. Please note that this is not the general multivariate Laplace definition found in other literature.

$$\begin{aligned} \text{StudentT}(x|\nu) &= \tfrac{\Gamma(\frac{\nu+1}{2})}{\sqrt{\pi\nu}\Gamma(\frac{\nu}{2})}\left[1 + \tfrac{x^2}{\nu}\right]^{-\frac{\nu+1}{2}} && \text{(Standard Student T distribution)} \\ \text{StudentT}(x|\nu, \mu, s) &= \tfrac{\Gamma(\frac{\nu+1}{2})}{\Gamma(\frac{\nu}{2})}\left(\tfrac{s}{\pi\nu}\right)^{\frac{1}{2}}\left[1 + \tfrac{s(x-\mu)^2}{\nu}\right]^{-\frac{\nu+1}{2}} && \text{(Scaled Student T distribution)} \end{aligned}$$

$$\text{Ber}(p) = p^x(1-p)^{1-x} = \begin{cases} p & \text{, if } x = 1 \\ 1-p & \text{, if } x = 0 \end{cases} \qquad \text{(Bernoulli distribution)}$$

$$\text{Categorical}(\boldsymbol{\pi}) = \begin{cases} \pi_1 & \text{, if } x = 1 \\ \vdots & \\ \pi_m & \text{, if } x = m \end{cases} \qquad \text{(Categorical Distribution)}$$

$$\text{s.t.} \quad \textstyle\sum_{i=1}^{m} \pi_i = 1$$

Alternatively,

$$\text{Categorical}(\boldsymbol{\pi}) = \exp\left\{ \begin{pmatrix} \log \pi_1 \\ \vdots \\ \log \pi_m \end{pmatrix}^T \begin{pmatrix} \mathbb{1}_{\{x=1\}} \\ \vdots \\ \mathbb{1}_{\{x=m\}} \end{pmatrix} \right\} \qquad \text{s.t.} \quad \textstyle\sum_{i=1}^{m} \pi_i = 1$$

Chapter 1

Introduction

Neural networks have dominated much of the machine learning applications and literature for the past decade, as they provide a very high degree of flexibility by being able to learn arbitrarily complicated functions. For such a growing field there is still a large gap in Bayesian applications to neural networks. Bayesian neural networks are the probabilistic alternative to neural networks, which provide the framework to explore uncertainty in the models. Bayesian neural networks also provide a natural method for model regularisation through the prior distribution and the use of ensembles of predictors, by averaging the predictions of several samples. An important distinction between BNNs and feed-forward NNs is that in the Bayesian setting we are no longer confined to simple models to avoid over-fitting as the prior helps to act as a regulariser, making them more compelling when the data is scarce.

An important advantage of BNNs as opposed to conventional NNs is that they provide a natural measure for the uncertainty of their predictions. For example the statement that a specific feed-forward NN is 95% accurate sounds appealing, but this gives no indication of the variance of the estimate. One very important use of BNNs may not be to make state-of-the-art models that predict with nearly perfect accuracy, but to build models that can help Statisticians to make informed decisions based on both the accuracy of the model and the uncertainty/variance of the predictor. For example, being told that surgery A has a 70% success rate and surgery B has a 65% success rate is very different to being told that surgery A has a 70% success rate give or take 30% and surgery B has a 65% success rate give or take 5%.

In this dissertation we will expand on the work of (Neal, 1995 [1]) using Hamiltonian Monte Carlo (HMC) to approximate the parameters of the Bayesian neural networks (BNNs). The goal of this dissertation is to understand how different priors impact the BNN performance and the resulting parameter approximations. Moreover, we will explore both the advantages and disadvantages of HMC in practice. We compare these to other methods of BNN parameter approximation, such as the Variational Inference (Peterson and Anderson, 1987 [24]; Hinton and van Camp, 1993 [23]; Jordan et al., 1999 [25]; Chen et al., 2014 [57]),

but also to feed-forward NNs.

This dissertation will focus at introducing both the feed-forward and Bayesian NN framework. Furthermore, we will examine methods of regularisation. Our main interest lies with dropout (Hinton et al., 2012 [34] and Srivastava et al., 2014 [35]), which currently serves as the state-of-the-art method of regularising NNs, but lacks the theoretical justification to its remarkable performance.

To set the background of this dissertation, the rest of this chapter will provide a brief introduction to Neural Networks (NNs), Convolutional Neural Networks (CNNs), and the underlying Bayesian approach to inference. We will conclude with an overview of what to expect throughout the different parts of this dissertation.

1.1 Neural Networks

The idea of Neural Networks dates back to (McCulloch and Pitts, 1943 [3]), where in an attempt to understand brain activity they introduced the first NN as single neuron with a two dimensional input and one dimensional (binary) output. Some more interesting ideas regarding the learning process for the weights/edges were proposed by (Hebb, 1949 [4]). The learning rule was to increase the weight on the edge "strengthening the synapse" connecting two neurons every time it was used, which was inspired by the way synapses are strengthened in the human brain.

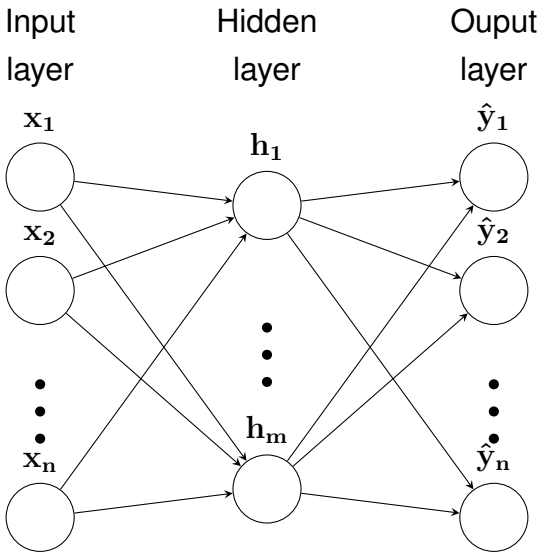

Figure 1.1 – *Above is a single (hidden) layer NN with inputs $\mathbf{X}$, outputs $\hat{\mathbf{Y}}$ and hidden layer $\mathbf{H}$. Note that for simplicity the arrows connecting the nodes represent all the operations performed from the previous layer to the current layer.*

Although early interest in the field stemmed from Psychologists and Neuroscientists, it did not take long for the teachings to intercept into other areas. Neurocomputing sprang

to life (Rosenblatt, 1957 [5] and 1961 [6]), with the first successful neurocomputer being built. The loss functions of the neural networks needed to be optimised somehow, with the popularisation of back-propagation (Rumelhart et al, 1986 [7]) the neural network field was reanimated.[1]. The Neural Networks we will be focusing on in this study are significantly deeper than their ancestral form. Many more have worked on neural networks since then developing state-of-the-art models for almost any task, from learning how to play video games, to identifying objects or even in discovering new drugs.

Let,

$$\mathbf{X} = \begin{pmatrix} \mathbf{x_1} \\ \mathbf{x_2} \\ \vdots \\ \mathbf{x_n} \end{pmatrix}, \qquad \hat{\mathbf{Y}} = \begin{pmatrix} \hat{\mathbf{y}}_\mathbf{1} \\ \hat{\mathbf{y}}_\mathbf{2} \\ \vdots \\ \hat{\mathbf{y}}_\mathbf{n} \end{pmatrix}, \qquad \mathbf{H} = \begin{pmatrix} \mathbf{h_1} \\ \mathbf{h_2} \\ \vdots \\ \mathbf{h_m} \end{pmatrix}$$

The hidden layer represents a non-linear transformation performed by the "activation function", typically a Rectified Linear Unit (ReLU), defined in the Notation section.

The activation function gets its name from the fact that it activates the neuron, acting as switch to turn a neuron on/off depending on a specific threshold, which in this case is zero. The NN in figure 1.1 is defined mathematically below:

$$\mathbf{f}(\mathbf{X}; \boldsymbol{\Theta}) = \hat{\mathbf{Y}}, \qquad \text{where} \quad \boldsymbol{\Theta} = \{\mathbf{W_0}, \mathbf{b_0}, \mathbf{W_1}, \mathbf{b_1}\}$$

$\mathbf{W_0}$ is the (d x m) weight matrix and $\mathbf{b_0}$ is the (1 x m) bias vector connecting the input layer to the output layer. We will refer to these as the first layer of parameters. Similarly $\mathbf{W_1}$ is the (m x k) weight matrix and $\mathbf{b_1}$ is the (1 x k) bias vector connecting the hidden layer with the output layer, which we will refer to as the second layer of parameters.

$$\mathbf{H} = \text{relu}\left(\mathbf{X}\mathbf{W_0} + \mathbf{b_0}\right)$$

For classification:

$$\hat{\mathbf{Y}} = \text{softmax}\left(\mathbf{H}\mathbf{W_1} + \mathbf{b_1}\right)$$

For regression:

$$\hat{\mathbf{Y}} = \mathbf{H}\mathbf{W_1} + \mathbf{b_1}$$

For the regression problem we propose the use of the squared loss function in order to perform optimisation, whereas for the classification problem we will use the cross-entropy loss. Note that back-propagation on the graph of the NN is as simple as computing the derivatives on the different modules in the graph. The "modules" in this context refer to operands e.g. the linear module (linear layer in the network), the ReLU module, e.t.c. In

[1] Note that back-propagation was formerly known as automatic-differentiation (Linnainmaa, 1970 [8]).

fact breaking the derivatives into different parts makes the computation faster as we do not have to recompute shared derivatives. Commonly used packages and software (Tensorflow, Theano) can compute these derivatives automatically as long as the model graph is defined correctly.

$$\text{Squared Loss} = \left\|\mathbf{Y} - \hat{\mathbf{Y}}\right\|_2^2 = \sum_{i=1}^{n} \left(y_i - f(\mathbf{x_i}; \Theta)\right)^2 \tag{1.1}$$

$$\text{Cross-entropy Loss} = -\sum_{i=1}^{n}\sum_{c=1}^{k} y_{ic} \log(\hat{y}_{ic}) = -\sum_{i=1}^{n}\sum_{c=1}^{k} y_{ic} \log(f(\mathbf{x_i}; \Theta)) \tag{1.2}$$

where, $\forall$ j $\in \{1, 2, ..., k\}$

$$y_{ij} = \begin{cases} 1, & \text{if the i}^{\text{th}} \text{ observation belongs to class j} \\ 0, & \text{otherwise} \end{cases}$$

Note that for NNs of 1 hidden layer or more we can no longer be optimised directly and therefore optimisation involves some approximation methods such as “Stochastic gradient descent” (SGD)[2], where the data used to estimate the gradient is fed batch by batch over several epochs (cycles over the data). The SGD algorithm can be found in the Appendix § A.1.1. In practice SGD can have very slow convergence and the solutions of the algorithm are usually local minima inconsistent with the global minimum, and therefore more sophisticated methods need to be applied for improved convergence. “Momentum” (Rumelhart et al., 1986 [7]) is a better approximation method that leads to more accurate updates by accumulating momentum in the direction of most decrease in the gradient providing an additional push in order to overcome local minima, this algorithm can be found in the Appendix § A.1.2. An improved variant of momentum is “Nesterov’s accelerated gradient” (Nesterov, 1983 [10]), the algorithm can be found in the Appendix § A.1.3.

Both SGD and Momentum unfortunately are sensitive to the value of the learning rate, which is a hyper-parameter that needs to be set prior to running the algorithm. If the learning rate is too low, then convergence can be painfully slow. Too high learning rates mean that the algorithm would never converge as each of the updates are too large, resulting in overshooting the global minimum every time. Ideally one would want to set an adaptive learning rate that is initially quite large and decreases gradually as the approximation approaches the true minimum. For this reason a more reliable and state-of-the-art gradient approximation method, is “Adam” (Kingma et al., 2015 [11]) and this will be the preferred method for this study, although the other optimisers will also be explored in Chapter 5.

[2] SGD was originally named “Adaline” by (Widrow, 1960 [9]).

For an arbitrary weight matrix $\mathbf{W}$ and bias vector $\mathbf{b}$, if $\mathbf{Z} = \mathbf{HW} + \mathbf{b}$, then:

$$\hat{\mathbf{y}}_\mathbf{i} = \text{softmax}(\mathbf{z}_\mathbf{i}) \quad , \quad \hat{y}_{im} = softmax(\mathbf{z}_\mathbf{i})_m = \frac{\exp(z_{im})}{\sum_{c=1}^{k} \exp(z_{ic})}$$

The first derivative of interest to us is $\frac{\partial L}{\partial \mathbf{Z}} = \hat{\mathbf{Y}} - \mathbf{Y}$, the derivation can be found in the Appendix § B.1.1.

Derivative	Solution	Section
$\frac{\partial L}{\partial \mathbf{H}}$	$\mathbf{W^T}(\hat{\mathbf{Y}} - \mathbf{Y})$	Appendix (§ B.1.2)
$\frac{\partial L}{\partial \mathbf{b}}$	$(\hat{\mathbf{Y}} - \mathbf{Y})$	Appendix (§ B.1.3)
$\frac{\partial L}{\partial \mathbf{W}}$	$\mathbf{H^T}(\hat{\mathbf{Y}} - \mathbf{Y})$	Appendix (§ B.1.4)

Figure 1.2 – *Useful derivatives required for back-propagation in NNs.*

Note that the above derivatives were defined for an arbitrary an weight matrix $\mathbf{W}$ and input matrix $\mathbf{H}$ and they can be trivially adapted to any inputs and/or weight matrices.

Finally, assuming that we are using a ReLU non-linearity (Fukushima, 1975 [12]). Let $\mathbf{H} = \text{relu}(\mathbf{M}) = \max(\mathbf{0}, \mathbf{M})$, then:

$$\frac{\partial L}{\partial \mathbf{M}} = \mathbf{W^T}(\hat{\mathbf{Y}} - \mathbf{Y}) \odot \mathbb{1}_{\{\mathbf{M}>\mathbf{0}\}}$$

The above result is derived in the Appendix § B.1.5.

Alternatively, we could be interested in using the smooth version of ReLU known as Softplus (Dugas et al., 2001 [13]). Let $\mathbf{H} = \text{softplus}(\mathbf{M}) = \log(\mathbf{1} + e^{\mathbf{M}})$, then:

$$\frac{\partial L}{\partial \mathbf{M}} = \mathbf{W^T}(\hat{\mathbf{Y}} - \mathbf{Y}) \odot \frac{\mathbf{1}}{\mathbf{1} + \mathbf{e}^{-\mathbf{M}}} \qquad \text{(see below for details)}^3$$

The above result is derived in the Appendix § B.1.5.

1.2 Convolutional Neural Networks

These are NNs most commonly applied to image datasets for object detection or identification, and pattern recognition. The idea behind CNNs dates back to (Hubel and Wiesel, 1962 [31]) who conducted experiments on cats, observing that cats' visual cortex is arranged into "receptive fields"[4] which helped them to detect a localised section of stimuli. This was incorporated into an algorithm for pattern recognition by (Fukushima et al., 1982 [32]). Many

[3] All operations in $\frac{\mathbf{1}}{\mathbf{1}+\mathbf{e}^{-\mathbf{M}}}$ are performed element-wise.

[4] Receptive fields are now most commonly referred to as "kernels" or "filters", depending on the literature.

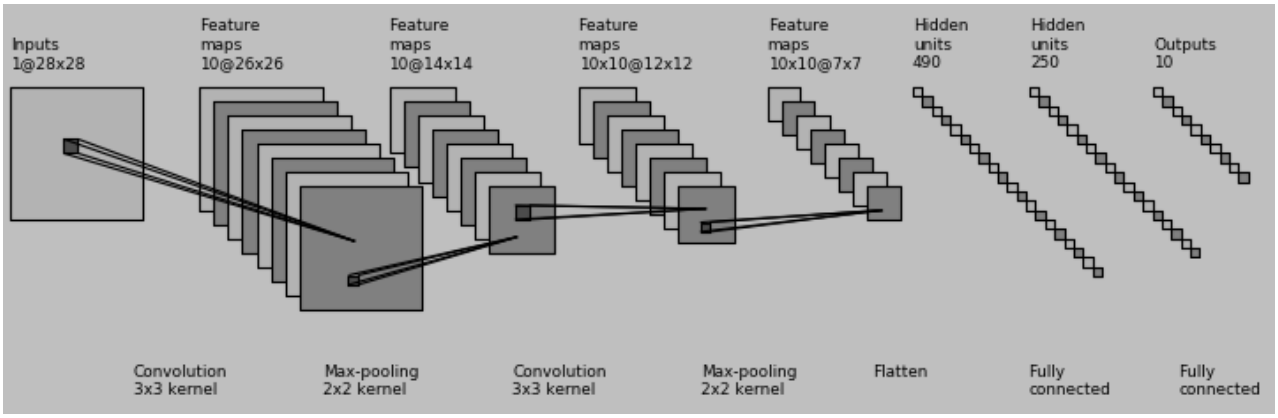

Figure 1.3 – *Graphical representation of the operations in the CNN used in this dissertation. The above CNN has a 3x3 convolution layer followed by a 2x2 max-pooling, followed by another 3x3 convolution layer followed by another 2x2 max-pooling. The model then has a fully-connected/flattened layer, and finally a hidden layer with 250 hidden units. The figure was generated by adapting the code from (Ding, 2016 [52]).*

years later a fully functional CNN model was applied by (Lecun et al., 1998 [33]) for object identification. Since then CNNs have been the best performing models at virtually any image task as demonstrated by their success in numerous Machine Learning competitions.

The ample success of these networks stems from their "translation invariance" property, which refers to the object appearance being independent of its location within the image itself. Intuitively the number "3" will still be a number "3" regardless where it can be found in the image. This is a very powerful property, but in order to fully understand how it arises in CNNs we would require an understanding of the model architecture.

The convolution is the first ingredient in the CNN. The "kernel/receptive field", slides over the image and maps each instance of the window to a different single entry in the feature map (in our example from figure 1.3 the kernel size is 3x3). This mapping can be found by summing over all the entries of the element-wise product of the kernel ($\mathbf{K}_f$) and the weight matrix $\mathbf{W}^{(c)}$ plus a bias term, where f is the instance of the kernel over the image and c is the feature map (channel of the output). Mathematically,

$$y_{kl}^{(c)} = \sum_{i=1}^{3}\sum_{j=1}^{3} x_{k+i-1,l+j-1} w_{ij}^{(c)} + b_c$$

where $c \in \{1, 2, ..., 10\}$, and $k, l \in \{1, 2, ...26\}$ for the first convolution in the above example, because we slide the kernel one pixel at a time, going from left to right and then down one row and repeating.

The next ingredient in the CNN is the "max-pool" mapping, which takes the maximum over a receptive field. In our specific example the kernel size is 2x2, and we slide the receptive field 2 pixels at a time so that none of the receptive fields are overlapping. The purpose of this pooling layer is to reduce the spatial resolution of the images, in particular halving the

resolution of the image[5]. Notice that the weight matrix $\mathbf{W}^{(c)}$ is only dependent on the feature map c and thus constant for all receptor fields, this in combination with the max-pool layer gives rise to the translation invariance in CNNs.

1.3 The Bayesian Approach

Bayesian inference will play a vital role in this dissertation thus making it crucial to understand the key concepts of Bayesian Modelling. A more detailed discussion will be provided in Chapter 2, where we will explore the concepts in the context of Bayesian Neural Networks. Moving away from the point estimate methods explored in the previous sections, and towards a probabilistic modelling approach we will formalise the tools that will form the skeleton for Bayesian Inference.

Let $\mathcal{D} = \{\mathbf{X}, \mathbf{Y}\} = \{(\mathbf{x_1}, \mathbf{y_1}), (\mathbf{x_2}, \mathbf{y_2}), \ldots, (\mathbf{x_n}, \mathbf{y_n})\}$. We are interested in finding the most likely distribution to have generated the data $\mathbf{Y}$. Assume that $\mathbf{y} = f(\mathbf{x}; \boldsymbol{\theta})$, we would be interested in finding the parameters $\boldsymbol{\theta}$ most likely to have generated the data $\mathbf{Y}$. In order to do this we need to make an assumption about the **likelihood distribution** $p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\theta})$. For example, in the classification setting we might assume that the likelihood has a softmax distribution on the outputs of the function $f(\mathbf{x}; \boldsymbol{\theta})$.

$$p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}) = \mathrm{softmax}(f(\mathbf{x}; \boldsymbol{\theta})) = F_{\mathbf{y}}(\mathbf{x}; \boldsymbol{\theta})$$

$$p(y = c|\mathbf{x}, \boldsymbol{\theta}) = \frac{\exp(f_c(\mathbf{x}, \boldsymbol{\theta}))}{\sum_{c' \in \mathcal{C}} \exp(f_{c'}(\mathbf{x}, \boldsymbol{\theta}))}$$

where $\mathcal{C}$ is the set of all classes in our data.

When trying to find the best possible model a key quantity of interest is the **posterior** distribution $p(\boldsymbol{\theta}|\mathcal{D})$, which tells us how likely the specific parameters are given the data.

$$p(\boldsymbol{\theta}|\mathcal{D}) = p(\boldsymbol{\theta}|\mathbf{Y}, \mathbf{X}) = \frac{p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\mathbf{Y}|\mathbf{X})} \propto p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\theta})p(\boldsymbol{\theta}) \tag{1.3}$$

Where $p(\boldsymbol{\theta})$ is said to be the **prior distribution** on the parameters.[6] In the i.i.d. data

[5] After each convolution we add a 1 pixel wide "padding" of pixels of zeros around the perimeter of the image to return the image back to its original size before the max-pooling is applied. For example the (26x26) image resulting after the first convolution is returned to (28x28) before the max-pool is applied.

[6] In order to be fully Bayesian we would also need to assume that the prior distribution on the parameters also has a hyperprior $p(\boldsymbol{\theta}|\gamma)$ distribution determining the value of the hyper-parameters of $\boldsymbol{\theta}$. The posterior distribution in (1.3) would become $p(\boldsymbol{\theta}, \gamma|\mathcal{D}) \propto p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\theta}, \gamma)p(\boldsymbol{\theta}|\gamma)p(\gamma)$.

setting (1.3) would simplify to:

$$p(\boldsymbol{\theta}|\mathcal{D}) \propto p(\boldsymbol{\theta}) \prod_{i=1}^{n} p(\mathbf{y_i}|\mathbf{x_i}, \boldsymbol{\theta}) = p(\boldsymbol{\theta}) \prod_{i=1}^{n} F_{\mathbf{y_i}}(\mathbf{x_i}; \boldsymbol{\theta})$$

The denominator $p(\mathbf{Y}|\mathbf{X})$ is the normalising term which we don't usually need to concern ourselves about until the end, but for completeness:

$$p(\mathbf{Y}|\mathbf{X}) = \int_{\boldsymbol{\theta} \epsilon \boldsymbol{\Theta}} p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\theta}) p(\boldsymbol{\theta}) d\boldsymbol{\theta}$$

Many times we receive a new data point $\mathbf{x_{n+1}}$ and we are interested in predicting the most probable $\mathbf{y_{n+1}}$. The best prediction we can make is to take the expectation under the posterior distribution.

$$\hat{\mathbf{y}}_{\mathbf{n+1}} = \mathbb{E}\left[F_{\mathbf{y_{n+1}}}(\mathbf{x_{n+1}}; \boldsymbol{\theta})\right]_{\boldsymbol{\theta}|\mathbf{Y},\mathbf{X}} = \int_{\boldsymbol{\theta} \epsilon \boldsymbol{\Theta}} F_{\mathbf{y_{n+1}}}(\mathbf{x_{n+1}}; \boldsymbol{\theta}) p(\boldsymbol{\theta}|\mathbf{Y}, \mathbf{X}) d\boldsymbol{\theta} \tag{1.4}$$

1.4 Dissertation Structure

The dissertation is aimed at providing a self-contained study of the discussed models, along with some of the intuition for the alternative approaches. Therefore, the dissertation is partitioned into five key chapters. Chapter 2 will introduce BNNs and discuss different prior functions but also exhibit graphically the surfaces of the drawn functions for different BNN architectures. Chapter 3 is aimed at providing an understanding of several methods of Approximate Inference, that will ultimately be applied to BNNs. Chapter 4 will discuss regularisation methods, both from the Frequentist and the Bayesian point of view and delve into Dropout and my interpretation of it in the Bayesian framework. Chapter 5 will present all the results of different Frequentist models and some novel Bayesian models. Finally, Chapter 6 will give a summary of the key findings and talk about further work that can be done in this field. The Appendix contains, algorithms and derivations that are not in the main body of the dissertation, along with the software used.

Chapter 2

Priors, and Bayesian Neural Networks

In Bayesian neural networks (BNNs) the prior distribution is aimed to capture prior assumed knowledge about the specific problem, or lack of knowledge in the case where an uninformative prior is used. Understanding the problem at hand is of paramount importance, in order to make a sensible choice of prior distribution.

When trying to classify the MNIST digits dataset one might prefer to use a sparsity inducing prior distribution such as the Laplace, Cauchy or Student-T distributions. These distributions are very spiky and are believed to be better suited for modelling sparse datasets such as MNIST. In other datasets, Normally distributed priors might prove advantageous. More importantly, prior distributions should be dataset and model specific as they incorporate our beliefs about the structure of the data.

In his thesis Neal (1995, [1]) showed the functions sampled from several NNs with different prior distributions (Student-T distribution and Cauchy). In this Chapter we will expand the drawn function space by considering different architectures, such as different activation functions and prior distributions, and compare these with similar drawn functions from (Neal, 1995). The thesis mainly used the "tanh" activation function, whereas in this dissertation we will also explore "ReLU" and "softplus", as they are thought to suffer less from vanishing gradients than "tanh" or "sigmoid" activation functions.

2.1 Bayesian Neural Networks (BNNs)

Before we begin, we will revisit the machinery required for Bayesian inference on BNNs. If we recall in the Introduction § 1.3 we let $p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}) = \mathrm{softmax}(f(\mathbf{x}; \boldsymbol{\theta}))$. In the classification

case we assume that the data have a Softmax likelihood:

$$\text{softmax}(f(\mathbf{x};\boldsymbol{\theta})) = F_{\mathbf{y}}(f(\mathbf{x};\boldsymbol{\theta}))$$

where $f(\mathbf{x};\boldsymbol{\theta})$ is the output of a NN with $\boldsymbol{\theta}$ being the parameters of the NN.

In the regression case we assume that the data have a Gaussian likelihood:

$$p(\mathbf{y}|\mathbf{x},\boldsymbol{\theta}) = \mathcal{N}_{\mathbf{y}}(f(\mathbf{x};\boldsymbol{\theta}), \tau\mathbf{I_n})$$

where τ is the model variance known as "model precision", and $f(\mathbf{x};\boldsymbol{\theta})$ is the output of the NN. Note that the variance-covariance matrix for the likelihood is assumed to be diagonal, which makes the calculations easier by making the assumption that the data is i.i.d., this does not necessarily have to be the case. As a rule of thumb, if we have no reason to suspect that the data are dependent, we always assume the simpler model.

We can then define the posterior distribution[1]:

$$p(\boldsymbol{\theta}|\mathcal{D}) = \frac{1}{Z}p(\mathbf{Y}|\mathbf{X},\boldsymbol{\theta})p(\boldsymbol{\theta})$$

Assuming that the parameters are independent of each other we have that:

$$p(\boldsymbol{\theta}|\mathcal{D}) = \frac{1}{Z}p(\mathbf{Y}|\mathbf{X},\mathbf{W_0},\mathbf{b_0},\mathbf{W_1},\mathbf{b_1},\ldots,\mathbf{W_m},\mathbf{b_m})\prod_{j=0}^{m}p(\mathbf{W_j})p(\mathbf{b_j})$$

where m is the number of hidden layers in the NN, and Z is the "model evidence"[2]:

$$Z = p(\mathbf{Y}|\mathbf{X}) = \int_{\mathbf{W_0}}\int_{\mathbf{b_0}}\cdots\int_{\mathbf{W_m}}\int_{\mathbf{b_m}} p(\mathbf{Y}|\mathbf{X},\boldsymbol{\theta})\prod_{j=0}^{m}p(\mathbf{W_j})p(\mathbf{b_j})d\mathbf{W_m}d\mathbf{b_m}\ldots d\mathbf{W_0}d\mathbf{b_0}$$

Both the posterior distribution and the model evidence are analytically intractable in most cases, apart from some simple cases such as Bayesian linear regression with a conjugate prior distribution. The target distribution for Bayesian inference is the posterior distribution, its analytical intractability consequently necessitates the use of the approximate inference methods which will be discussed in more detail Chapter 3.

[1] Note that for the duration of the dissertation we will not be fully Bayesian in our approach, and the hyperparameters will be fixed (chosen by a grid-search method).

[2] Assuming the data are i.i.d, then we can further decompose $p(\mathbf{Y}|\mathbf{X},\boldsymbol{\theta}) = \prod_{i=1}^{n} p\left(\mathbf{y_i}|f(\mathbf{x_i};\boldsymbol{\theta})\right)$

2.2 Priors

It has long be thought that the structure of the prior distributions has an impact on sparsity promotion. The benefit of sparsity promotion is the "pruning" effect on the parameters. One example of early work in this field is Automatic Relevance Determination (Mackay, 1992 [55], Neal, 1995 [1]) which aids in applying Occam's razor into the modelling process. This regularisation aspect of prior distributions is highly attractive in the field of Machine Learning, as it provides a natural safeguard against over-fitting that is essentially incorporated into the model architecture. Many others have studied Bayesian sparsity since then, (Gerven et al., 2009 [53], Jylänki et al., 2014 [21], Simpson et al., 2017 [37], Ingraham and Marks, 2017 [54]).

The practicality of incorporating the model regularisation and pruning into the prior distribution has motivated the more detailed exploration of prior distributions in the sections that will follow. We will explore the impact of different prior distributions on the drawn functions of neural networks.

2.3 Drawn functions

This section will explore the impact of different model architectures, such as varying degrees of model depth, width and the use of different activation functions on the drawn function. The code used to create the graphics in this chapter can be found in (Stavrou, 2017 [56]) [3] with instructions on how to use it.

2.3.1 1 hidden layer BNN

We begin analysing the effect of different prior distributions on the 1 hidden layer NN architectures. The Bayesian linear regression case is not as exciting, as that is the trivial case where we will always draw a flat plane regardless of the prior distribution and is therefore not shown, as it is a linear combination of the inputs. For the duration of this chapter, we will refer to the parameters of the model as "weights", not distinguishing between bias terms and weight matrices, this is only to make the explanation more concise.

Figure (2.1) is quite interesting as it shows that using a Normal prior for the weights leads approximately to a Gaussian Process (GP), a result argued also by (Neal, 1995). A more fascinating result is that for the Laplace priors, using low variance priors results in a drawn function that represents a less smooth GP. Notice that for a higher variance the drawn function becomes very spiky, a consequence of the heavier tails of the Laplace distribution in

[3] The link to the specific folder for this chapter on the Github repository is: https://github.com/cstavrou/Priors-and-Algorithms-for-Bayesian-Neural-Networks/

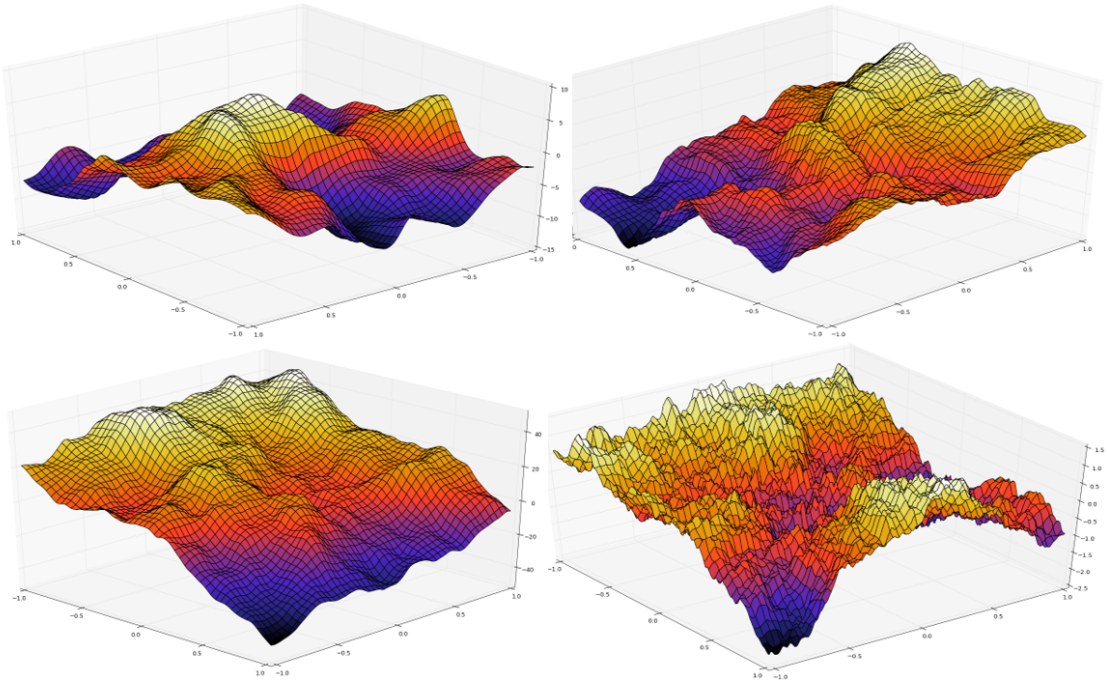

Figure 2.1 – *Drawn functions from 1 hidden layer BNNs with 10000 hidden units each, evaluated by taking the average drawn function from 1000 drawn samples. The top left plot has a* $\mathcal{N}(\mathbf{0}, \frac{10^2}{d_j}\mathbf{I}_{d_j})$ *prior over the weights, the bottom left plot has a* $\mathcal{N}(\mathbf{0}, \frac{20^2}{d_j}\mathbf{I}_{d_j})$ *prior over the weights. Where* d_j *is the number of inputs from the previous layer (the number of columns of the units being multiplied by the weight matrix. The top right is a plot with a* $Laplace(\mathbf{0}, \frac{5^2}{d_j}\mathbf{I}_{d_j})$ *prior on the weights, and the bottom right plot has a* $Laplace(\mathbf{0}, \frac{10^2}{d_j}\mathbf{I}_{d_j})$ *prior distribution on the weights. All the networks have tanh non-linearities for the hidden layer, and the input region is -1 to +1.*

comparison to the Normal distribution.

We will next consider using another BNNs with unscaled standard Student T prior distributions on the weights, all of which are independent and identically distributed. The Student T distribution again has heavier tails than the Normal distribution, which implies that there is a higher probability of a weight taking a large value than for the Normally distributed weights, which tend to be more centred around the mean. Decreasing the degrees of freedom in the Student T distribution results in the tails becoming heavier, as the distribution has lower certainty for the mean value. We will also explore the drawn functions when using a "spike and slab" (Mitchell and Beauchamp, 1988 [27]; Ishwaran and Rao, 2005 [28]) prior distribution over the weights. The spike and slab prior has been used by many in the past to perform Bayesian variable selection. It works by assigning some of the weights to be exactly equal to zero with some probability π, which results in the spike, with probability $(1-\pi)$ the weights are drawn from a wide distribution, typically a Gaussian with large variance, known as the slab,

$$\mathcal{SS}_w(\pi, \sigma^2) = p(w|\pi, \sigma) = \pi\delta(w-0) + (1-\pi)\mathcal{N}_w(0, \sigma^2)$$

where $\delta(w-0)$ is the dirac delta function.

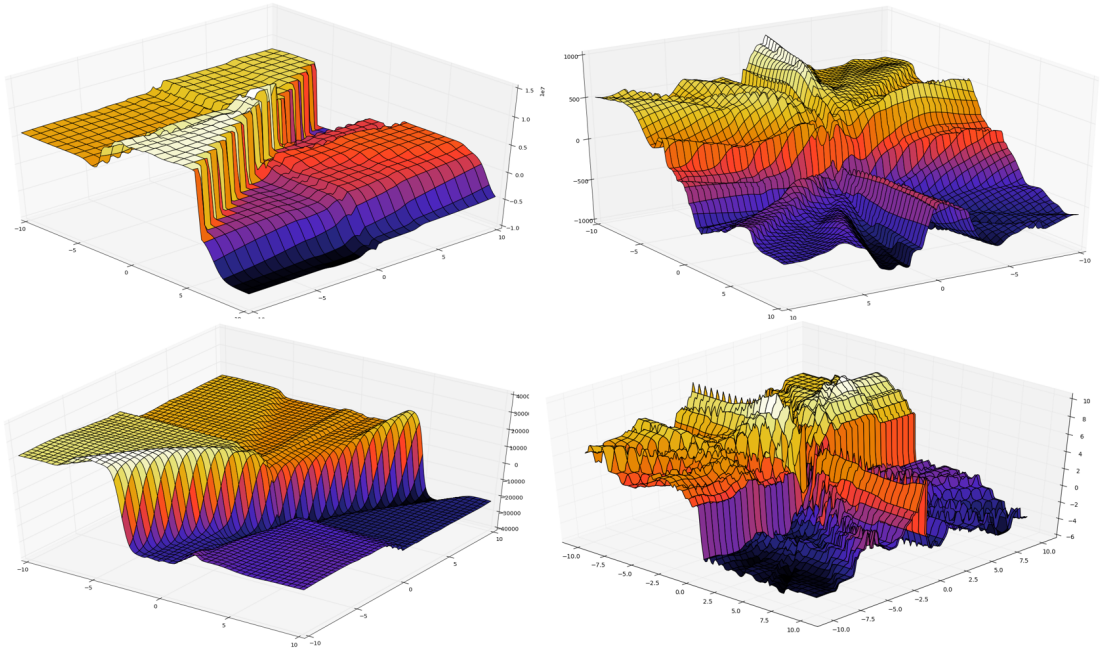

Figure 2.2 – *Drawn functions from 1 hidden layer BNNs with 10000 hidden units each, evaluated by taking the average drawn function from 1000 drawn samples. The top left plot displays a BNN with a prior function over each of the weights that is i.i.d. standard Student T with 0.5 degrees of freedom (ν). The bottom left plot has i.i.d. priors for each of the weights of Student T distribution with $\nu = 1$, which is also known as the Cauchy distribution. The top right plot has i.i.d. priors over the weights which are Student T distributed with $\nu = 1.5$. In the bottom right plot, the weights are i.i.d. with spike and slab prior distribution, with $\pi = 0.2$ and $\sigma = \frac{30}{d_j}$. All the networks have tanh non-linearities for the hidden layer, and the input region is -10 to +10.*

The drawn functions in figure (2.2) are considerably different from the ones we observed for the Normal and the Laplace prior distributions over the weights. For the lower degrees of freedom the function forms a relatively smoothed plane with large spikes leading to another plateau. The drawn functions using a Student T distribution for a prior appear to have a similar structure. The Student T prior appears to be a smoother, but very similarly shaped to the drawn function for the Spike and Slab prior.

In addition to using tanh non-linearities, ReLU and Softplus non-linearities were also used for the 1 hidden layer BNNs, their plots are much smoother planes. For the plots please refer to the Appendix § C.1.1. Both the ReLU and Softplus non-linearities seem to absorb the extensive curvature in the surfaces of the drawn functions. This helps to provide some graphical intuition to the reader as to why ReLU tends to be so widely used in practice, for it is believed to suffer less from exploding gradients than other rectifiers.

2.3.2 2 hidden layer BNN

We will now delve into slightly deeper neural networks, the term “deeper” refers to the networks possessing more hidden layers. For deep learning standards these networks are still considered to be quite shallow and are therefore not yet at the stage requiring special treatment to avoid vanishing gradients, or large singular values in the Jacobian. We would expect that as we increase the depth of the models, the samples from the prior may become more varying (Duvenaud et al., 2014 [30]). In [30] it was observed that increasing the depth of the models resulted in one dominating singular value, which they explained to be the result of the heavy-tailed Jacobian distribution. The state-of-the-art networks in (Duvenaud et al., 2014 and He et al., 2015 [29]) leverage direct skip connections between the inputs and the hidden layers in order to hinder these adverse effects arising from increasingly deep models.

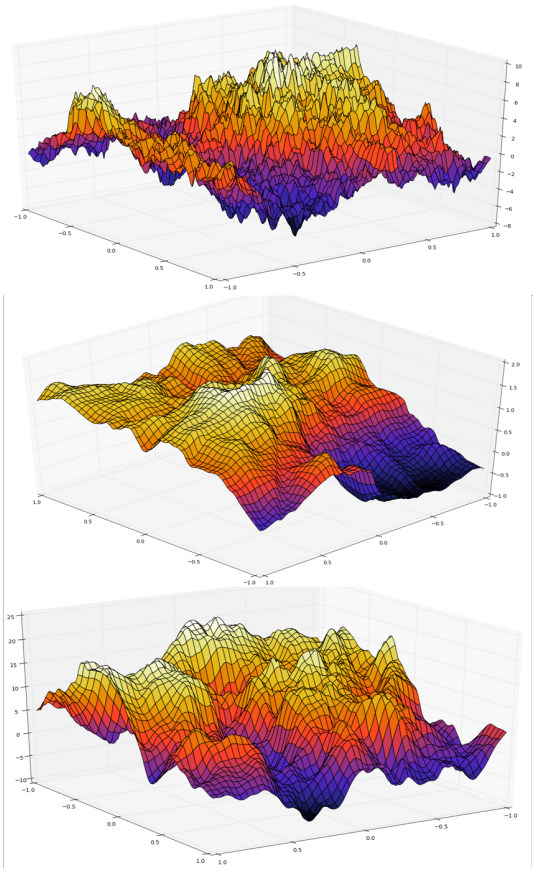

Figure 2.3 – *Drawn functions from 2 hidden layer BNNs with 1000 hidden units for each of the hidden layers (same for all drawn functions). The drawn functions are evaluated by taking the average of 1000 samples. The upper plot has a* $\mathrm{Laplace}(\mathbf{0}, \frac{10^2}{d_j}\mathbf{I}_{d_j})$ *prior over the weights. The middle plot has a* $\mathrm{Laplace}(\mathbf{0}, \frac{5^2}{d_j}\mathbf{I}_{d_j})$ *prior over the all the weights. The bottom plot has a* $\mathcal{N}(\mathbf{0}, \frac{10^2}{d_j}\mathbf{I}_{d_j})$ *prior distribution over the weights. Where* d_j *is the number of inputs from the previous layer (the number of columns of the units being multiplied by the weight matrix. All the networks have tanh non-linearities for both of the hidden layers, and the input region is -1 to +1.*

Similarly to the 1 hidden layer networks seen in figure (2.1) the drawn function from the model with the Normally distributed prior over the weights looks very similar to a GP. A similar result is also observed for the Laplace priors for low scale values ($b = \frac{5^2}{d_j}$), whereas for the higher scale value of $b = \frac{10^2}{d_j}$, the high variance in the first hidden layer of the network appears to crowd out the signal from the second hidden layer leading to a very rough function surface. When swaping the tanh activation function to either ReLU or Softplus, we no longer see high variance of the prior distribution playing a large role in the shape of the drawn function (see figure 2.4).

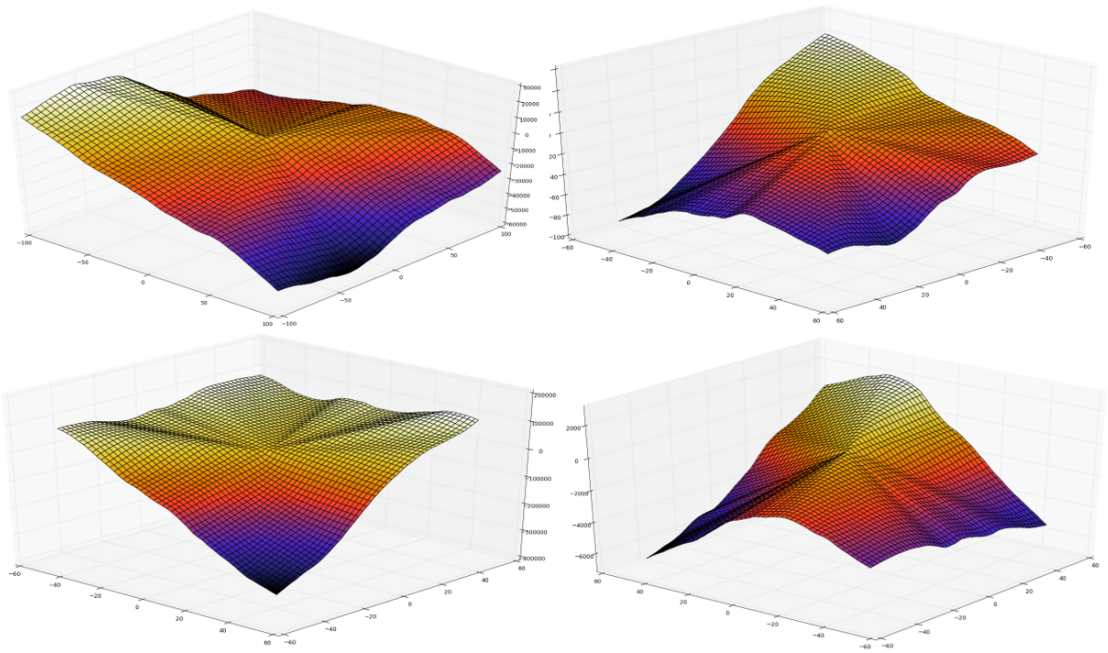

Figure 2.4 – *Drawn functions from 2 hidden layer BNNs with 10000 hidden units for each of the hidden layers (same for all drawn functions). The drawn functions are evaluated by taking the average of 100 samples. The top left plot has a* $\mathcal{N}(\mathbf{0}, \frac{10^2}{d_j}\mathbf{I}_{d_j})$ *prior distribution over the weights, whilst the bottom left plot has a* $\mathcal{N}(\mathbf{0}, \frac{20^2}{d_j}\mathbf{I}_{d_j})$ *prior distribution over the weights. The top right plot has a* $\text{Laplace}(\mathbf{0}, \frac{10^2}{d_j}\mathbf{I}_{d_j})$ *prior over the weights, whilst the bottom right plot has a* $\text{Laplace}(\mathbf{0}, \frac{20^2}{d_j}\mathbf{I}_{d_j})$ *prior over the weights. Where* d_j *is the number of inputs from the previous layer (the number of columns of the units being multiplied by the weight matrix. All the networks use ReLU activation functions for both of the hidden layers, and the input region is -60 to +60.*

A very similar plot can be observed for Softplus activation functions, which can be found in the Appendix § C.1.2. We believe the highly smooth surface is an artefact of the high variance of the inputs to the first hidden layer. In fact, changing the prior for the weights multiplying the inputs $\mathbf{X}$ to have a standard deviation of 1 in the Normal case, and a scale of 1 in the Laplace case, allows us to distinguish the impact of different activation functions on the samples from the prior distribution. From figure 2.5 it is obvious that both the ReLU and Softplus have a smoothing effect on the drawn functions. Although both ReLU and Softplus provide a mild curvature, if we observe closely, we can detect that the drawn functions using ReLU activatios have a marginally coarser surface. We believe that this is caused from the hard 0 minimum constraint on the outputs of the hidden layer, as opposed to the smoother progression towards 0 that Softplus provides.

We will now consider using Standard T prior distribution functions of varying degrees of freedom. Much like (Neal, 1995), we observe a plateau type structure. One would expect such models to be naturally good at classification tasks due to the formation of a step type structure. We could thus think of the different levels as different classes we would want our function to predict. However, one can already see a potential problem with having such prior

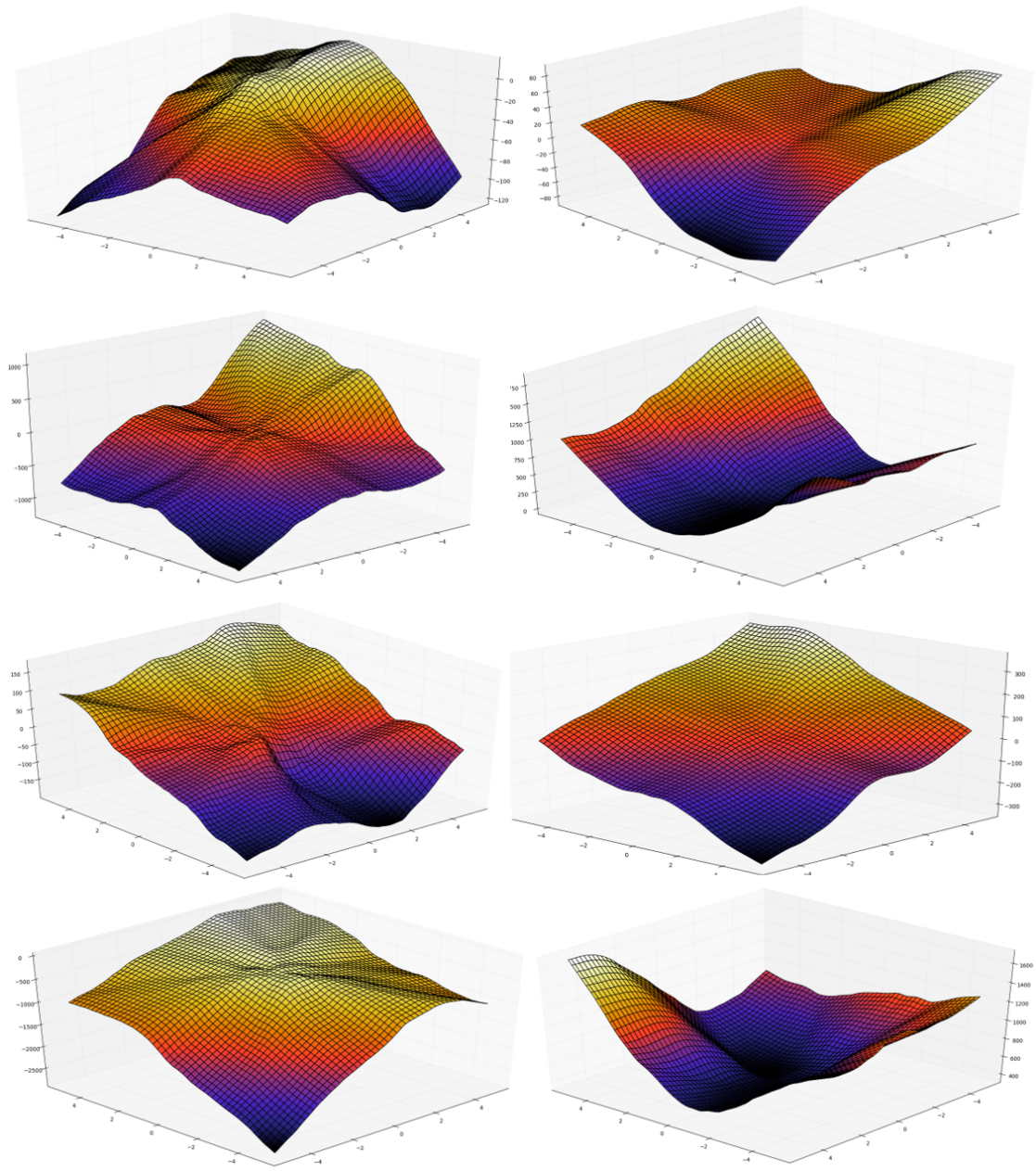

Figure 2.5 – *Drawn functions from 2 hidden layer networks with 1000 hidden units for each layer. The functions plotted are the average of 1000 samples. For all the plots the first layer of weights is sampled from a distribution with variance 1 for the Normal prior networks and scale 1 for the Laplace prior networks respectively. All the plots on the left-hand side use ReLU activation functions for the hidden layers, and on the right, Softplus activation functions. The top two plots have all the weights for the remaining 2 hidden layers drawn independently from a* $\mathrm{Laplace}(0, 10^2/d_j)$ *distribution. The second row plots have all their weights drawn independently from a* $\mathrm{Laplace}(0, 20^2/d_j)$*. The third row plots have all the weights for the remaining two layers of weights drawn independently from a* $\mathcal{N}(0, 10^2/d_j)$ *distribution, and the final row from a* $\mathcal{N}(0, 20^2/d_j)$ *prior distribution. The input region for all the plots is -5 to +5.*

functions. Separating the different plateaus is a wall with almost infinite gradients. Depending on the likelihood function that will be used this might make it extremely difficult, if not impossible to learn the underlying function. The result in practice is that Student T distributions priors are very difficult to tune to provide stable results.

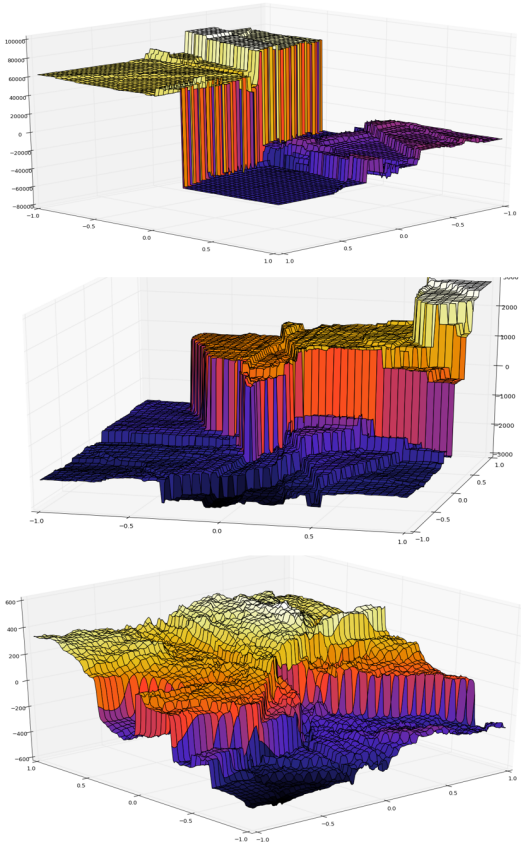

Figure 2.6 – *Drawn functions from 2 hidden layer BNNs with 1000 hidden units for each of the hidden layers (same for all drawn functions). The drawn functions are evaluated by taking the average of 1000 samples. The upper plot has a* $\mathrm{StudentT}(\nu = 0.5)$ *prior over the weights. The middle plot has a* $\mathrm{StudentT}(\nu = 1)$ *prior over the all the weights. The bottom plot has a* $\mathrm{StudentT}(\nu = 1.5)$ *prior distribution over the weights. All the networks have tanh non-linearities for both of the hidden layers, and the input region is -1 to +1.*

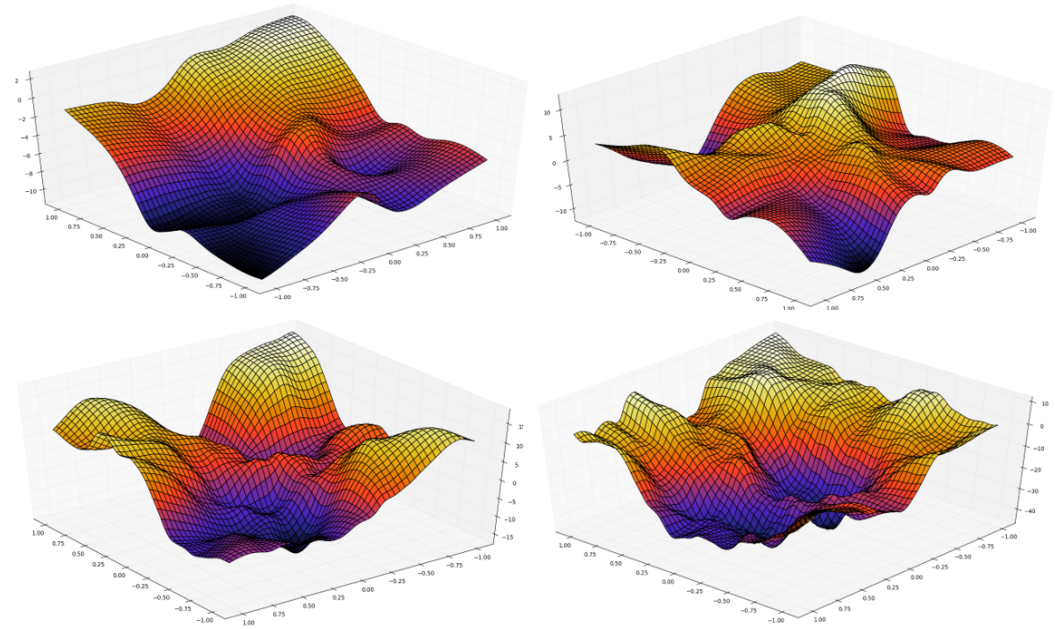

Figure 2.7 – *Drawn functions from 2 hidden layer BNNs with 1000 hidden units for each of the hidden layers, evaluated by taking the average of 1000 samples. All the plots have* $\sigma^2 = 1$ *for the Spike and Slab prior over the weights multiplying the inputs, the top row has* $\pi = 0.2$ *and bottom row has* $\pi = 0.5$*. The remaining layers of weights are sampled independently from; an* $\mathcal{SS}(0.2, 20^2/d_j)$ *prior for the top left plot, an* $\mathcal{SS}(0.2, 30^2/d_j)$ *for the top right plot, an* $\mathcal{SS}(0.5, 20^2/d_j)$ *for the bottom left plot, and an* $\mathcal{SS}(0.5, 30^2/d_j)$ *for the bottom right plot. All the networks use tanh non-linearities for their hidden layers, and the input region is -1 to +1.*

When using spike and slab priors (figure 2.7) we observed that the drawn functions had an intrinsic smooth structure. Unlike the functions drawn from pure Normal distributions (fig-

ures 2.1 and 2.3) the surfaces observed were less rough and possessed some curved slabs. This can be attributed to the property of spike and slab priors of assigning exactly 0 to the weights, and as a result of eliminating variables the drawn function tends to be smoother. We would expect such priors to perform well in scenarios where the data is sparse and thus want to assign a zero weight to uninformative inputs. This can be typical in image data, where large parts of the image are just the background.

2.3.3 2 hidden layer mixture of priors BNN

In this section, we will explore using a mixture of different priors for the 2 hidden layer networks. The combinations were designed to utilise the favourable qualities of the different prior distributions, for example the smoothness of the Normal distribution, and the sparsity promoting of the Student T or Laplace distributions.

We can identify that using a mixture of different priors allows for greater flexibility in the drawn function (figure 2.8). With the mixture of Normal and Student T priors with degrees of freedom $\nu = 0.5$ we can observe a plateau-like structure similar to the one we witnessed in figure 2.6, with the added benefit of the smooth surface, meaning that the drawn functions would no longer suffer from potentially infinite gradients. This result is the same for the higher variance Normal priors tested, that have a mildly coarser surface. As we increase the degrees of freedom of the Student T distribution we begin to loose the plateau-like behaviour and tend towards the functions observed when using fully Normal priors over the weights, an unsurprising result given the nature of the Student T distribution.

As in the Normal distribution case explored in figure 2.8 we have the plateau-like effect caused from the Student T distribution priors. For $\nu = 0.5$ we have sharper edges than those observed when using the Normal distribution priors for the first 2 layers of the networks. In general, the networks using the Laplace and Student T mixture can be related to the samples drawn when using a Normal and T distribution mixture, with a rougher surface.

To conclude, we expect that the surfaces of the drawn functions will ultimately have an impact on our ability to model different data more accurately. We suspect that the rougher functions would benefit mostly in sparser datasets where our training algorithm could be aimed at detecting abrupt signals in the data, e.g. in the MNIST digits dataset used in chapter 5 we have an input of 784 pixels, most of which are zero and few equal to 1 and thus the signal arising from the inputs is very acute. In contrast with regression of continuous inputs where the signal might arise from a smooth function, where smoother priors like the Normal distribution might make more sense.

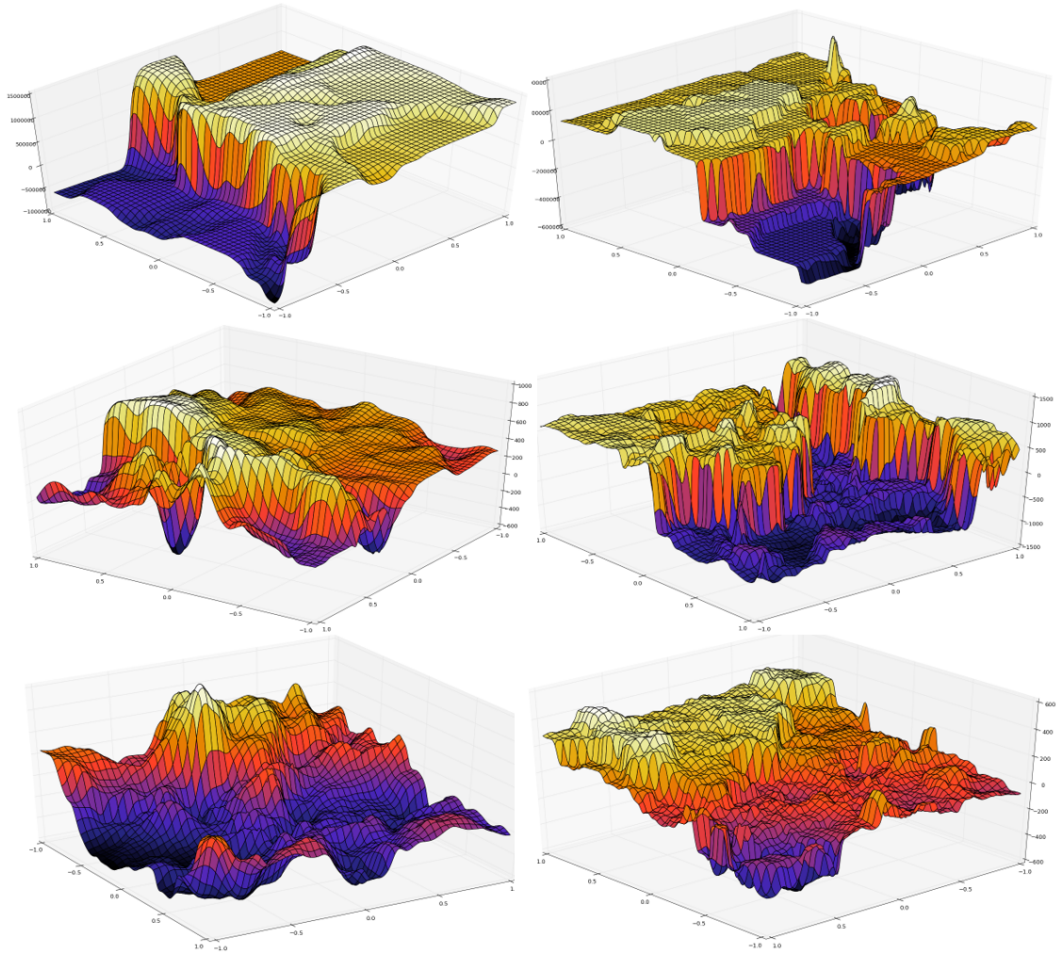

Figure 2.8 – *Drawn functions from 2 hidden layer BNNs with 1000 hidden units for each of the hidden layers, evaluated by taking the average of 1000 samples. The plots on the left have their weights drawn independently from a* $\mathcal{N}(0, 10^2/d_j)$ *for the first two layers (linear layer and first hidden layer). The ones on the right from a* $\mathcal{N}(0, 20^2/d_j)$*. The final layer of weights (from second hidden layer to the output) are drawn independently from a standard Student T with degrees of freedom 0.5, 1, and 1.5 for the first, second and third rows of plots respectively. All the networks use tanh non-linearities for their hidden layers, and the input region is -1 to +1.*

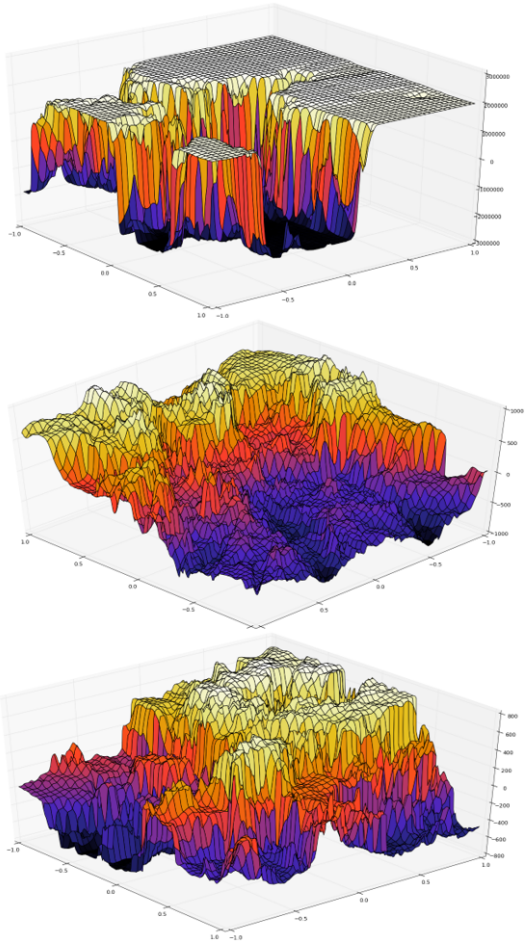

Figure 2.9 – *Drawn functions from 2 hidden layer BNNs with 1000 hidden units for each of the hidden layers, evaluated by taking the average of 1000 samples. The plots have their weights drawn independently from a* $\mathrm{Laplace}(0, 10^2/d_j)$ *for the first two layers (linear layer and first hidden layer). The final layer of weights (from second hidden layer to the output) are drawn independently from a standard Student T with degrees of freedom 0.5, 1, and 1.5 for the top, middle and bottom plots respectively. All the networks use tanh non-linearities for their hidden layers, and the input region is -1 to +1.*

Chapter 3

Methods of Approximate Inference

We have seen in § 2.1 that the computations required for BNNs very quickly become analytically intractable, and therefore we require the use of approximate methods. This chapter is intended to develop a sufficient understanding of the approximate inference machinery that will be used in the subsequent chapters of this work.

The approximate inference methods seen in this chapter are only a small selection of the current approximation methods applied to BNNs. Recent research has been focused on finding good approximations that scale well on larger datasets (Hernández-Lobato and Adams, 2015 [39], Blundell et al., 2015 [40]). Although, both '*Probabilistic Backpropagation*' and '*Bayes by Backprop*' are computationally efficient, they can lead to large approximation errors and an underestimation of the model uncertainty, and will therefore not be explored in this study.

3.1 MCMC definitions

In most cases it might be impossible to sample directly from the *target distribution*. MCMC facilitates approximate sampling from the target distribution. Essentially, this involves setting up a Markov chain, s.t. its invariant distribution is the target distribution. The target distribution is the distribution we are interested in sampling from, which is typically the posterior distribution.

This section will not provide a comprehensive introduction into MCMC, a more detailed exploration can be found in (Neal, 1993) and (Gilks et al., 1996). We will briefly introduce the vital components of MCMC using adaptations of the notation and definitions from (Neal, 1993 [14] and Bishop, 2006 [16]).

Definition 1. Markov Chain
A sequence $X_0, X_1, \ldots, X_n$ of random variables forms a *Markov Chain* if:

$$p(X_t|X_{t-1}, \mathcal{X}) = p(X_t|X_{t-1}) \quad , \quad \forall t \in \{1, 2, \ldots, n\} \quad , \quad \forall \mathcal{X} \subseteq \{X_0, \ldots, X_{t-2}\}$$

The joint distribution of the Markov chain is characterised by the *marginal distribution* on X_0 (initial distribution $p(X_0)$) and the *transition probability distribution*, i.e. the conditional distribution $p(X_t|X_{t-1})$, $\forall t \in \{1, 2, \ldots, n\}$.

$$p(X_n, X_{n-1}, \ldots, X_0) = p(X_0) \prod_{i=1}^{n} p(X_i|X_{i-1})$$

Let the transition probability for state x' at time $n+1$ from state x at time n be,

$$T_n(x, x') = p(X_{n+1} = x'|X_n = x)$$

Let the marginal distribution for state x at time $n+1$ to be,

$$p_{n+1}(x) = p(X_{n+1} = x) = \sum_{\tilde{x}} p_n(\tilde{x}) T_n(\tilde{x}, x)$$

Definition 2. Invariant distribution
We say that the distribution $\pi(x)$ is invariant w.r.t. the Markov Chain with transition probabilities $T_n(x, x')$ if,

$$\pi(x) = \sum_{\tilde{x}} \pi(\tilde{x}) T_n(\tilde{x}, x) \quad \forall n \in \mathbb{N}$$

Definition 3. Detailed Balance
If the transition state is selected according to probabilities from the invariant distribution π, then the probability of going from state x to x' is equal to the probability of going from state x' to x. In other words, detailed balance holds if for all x,

$$\pi(x) T(x, x') = \pi(x') T(x', x)$$

Clearly if detailed balance condition holds then the distribution π is an invariant

$$\sum_{x'} \pi(x') T(x', x) = \sum_{x'} \pi(x) T(x, x') = \pi(x) \underbrace{\sum_{x'} T(x, x')}_{=1} = \pi(x)$$

Definition 4. Ergodic Markov Chain
A Markov chain is said to be '*ergodic*' if regardless of the initial probabilities $p_0(x)$ the probabilities $p_n(x)$ converge to the stationary distribution as n tends to infinity, that is,

$$\lim_{n \to \infty} p_n(x) = \pi(x) \quad , \quad \forall x \in \mathcal{S}$$

where $\mathcal{S}$ is the '*state space*' of the Markov Chain. A sufficient condition for ergodicity is that with some non-zero probability starting from any state $x \in \mathcal{S}$ we can transition to any other state $x' \in \mathcal{S}$ within some finite number of steps k,

$$\exists k \in \mathbb{N} \text{ s.t. } p(X_k = x' | X_0 = x) = T^k(x, x') > 0 \quad , \quad \forall x, x' \in \mathcal{S}$$

where $T^k(x, x')$ corresponds to the transition probability kernel for k time steps ahead. If $T(.)$ is a transition probability matrix then it would correspond to raising the matrix to the power k.

Since we want our Markov chain to sample from the target distribution, arranging the Markov chain such that the target distribution is invariant is the first requirement. The reason for the invariance requirement is that we do not want the process of generating samples to change the distribution every time. The second requirement is that the Markov chain is also ergodic, as an ergodic Markov chain can only have one invariant distribution. This means that the sampling scheme should converge to the correct target distribution.

3.2 Metropolis Hastings

One class of MCMC is the Metropolis-Hastings (MH) algorithm (Hastings, 1970 [18]) which is a generalisation of the Metropolis algorithm (Metropolis et al., 1953 [17]).

Algorithm 1: Metropolis-Hastings algorithm

initialise $\mathbf{x^{(0)}} = (x_1^{(0)}, \ldots, x_d^{(0)})$; // Initial sample
for $t \in \{1, 2, \ldots, M\}$ **do**
  Sample $\mathbf{x}^* \sim q(\mathbf{x}|\mathbf{x^{(t-1)}})$; // Sample from proposal distribution
  $\alpha(\mathbf{x}^*, \mathbf{x^{(t-1)}}) = \min\left(1, \frac{\tilde{p}(\mathbf{x}^*) q(\mathbf{x^{(t-1)}}|\mathbf{x}^*)}{\tilde{p}(\mathbf{x^{(t-1)}}) q(\mathbf{x}^*|\mathbf{x^{(t-1)}})}\right)$; // Acceptance probability
  $u \sim U(0, 1)$; // Random sample from uniform(0,1) distribution
  if $u < \alpha(\mathbf{x}^*, \mathbf{x^{(t-1)}})$ **then**
    $\mathbf{x^{(t)}} = \mathbf{x}^*$; // Accept the sample
  else
    $\mathbf{x^{(t)}} = \mathbf{x^{(t-1)}}$; // Reject sample
  end
end

Figure 3.1 – Where $p(\mathbf{x}) = \tilde{p}(\mathbf{x})/Z$ is the target distribution and Z is the normalising factor. $q(\mathbf{x}|\mathbf{x}')$ is the proposal distribution, d are the number of dimensions of each sample, and M is the total number of samples.

Using the definition of $\alpha(.,.)$ and the symmetry of $\min(.,.)$ we have that,

$$
\begin{aligned}
T(\mathbf{x}, \mathbf{x}') &= q(\mathbf{x}'|\mathbf{x})\alpha(\mathbf{x}', \mathbf{x}) \\
&= q(\mathbf{x}'|\mathbf{x}) \min\left(1, \tfrac{p(\mathbf{x}')q(\mathbf{x}|\mathbf{x}')}{p(\mathbf{x})q(\mathbf{x}'|\mathbf{x})}\right)
\end{aligned}
$$

$$
\begin{aligned}
\Rightarrow p(\mathbf{x})T(\mathbf{x}, \mathbf{x}') &= p(\mathbf{x})q(\mathbf{x}'|\mathbf{x})\alpha(\mathbf{x}', \mathbf{x}) \\
&= \min\left(p(\mathbf{x}')q(\mathbf{x}|\mathbf{x}'), p(\mathbf{x})q(\mathbf{x}'|\mathbf{x})\right) \\
&= \min\left(p(\mathbf{x})q(\mathbf{x}'|\mathbf{x}), p(\mathbf{x}')q(\mathbf{x}|\mathbf{x}')\right) \\
&= p(\mathbf{x}')q(\mathbf{x}|\mathbf{x}')\alpha(\mathbf{x}, \mathbf{x}') \\
&= p(\mathbf{x}')T(\mathbf{x}', \mathbf{x})
\end{aligned}
$$

therefore MH satisfies detailed balance, which thereby guarantees that $p(\mathbf{x})$ is an invariant distribution. However, satisfying the ergodicity condition depends on the target and proposal distributions of the specific example.

3.3 Gibbs Sampling

A special case of the MH algorithm is Gibbs Sampling (Geman and Geman, 1984 [19]), which is useful when we cannot sample from the proposal distribution as required by MH, but can instead sample each of the dimensions of $\mathbf{x}$ separately from the conditional proposal distribution.

Suppose that our target distribution is $p(\mathbf{x})$ and that we wish to generate M samples.

Algorithm 2: Gibbs sampling algorithm

initialise $\mathbf{x^{(0)}} = (x_1^{(0)}, \ldots, x_d^{(0)})$; `// Initial sample`
for $t \in \{1, 2, \ldots, M\}$ **do**
  for $j \in \{1, 2, \ldots, d\}$ **do**
    Sample $x^* \sim p(x_j|\mathbf{x_{\setminus j}^{(t-1)}})$; `// Sample from conditional distribution`
    $x_j^{(t)} = x^*$; `// $j^{th}$ dimension of the $t^{th}$ sample`
  end
end

Figure 3.2 – Where $p(\mathbf{x})$ is the target distribution. $p(x_i|\mathbf{x}_{\setminus \mathbf{i}})$ is the conditional distribution for i^{th} dimension of $\mathbf{x}$, d are the number of dimensions of each sample, and M is the total number of samples.

Remark: Let us first note that when we sample x_i from the conditional distribution, the remaining values $\mathbf{x}_{\setminus \mathbf{i}}$ remain unchanged. Similarly when we sample x_i' the values of $\mathbf{x'}_{\setminus i}$ all remain unchanged, and obviously it follows that $\mathbf{x}_{\setminus \mathbf{i}} = \mathbf{x'}_{\setminus \mathbf{i}}$.

By applying Bayes rule and the remark above we can easily show that Gibbs sampling satisfies the detailed balance equation guaranteeing the invariance of the target distribution $p(\mathbf{x})$. Since,

$$\begin{aligned} p(\mathbf{x})T(\mathbf{x},\mathbf{x}') &= p(x_i,\mathbf{x}_{\setminus\mathbf{i}})p(x_i'|\mathbf{x}_{\setminus\mathbf{i}}) \\ &= p(x_i|\mathbf{x}_{\setminus\mathbf{i}})p(\mathbf{x}_{\setminus\mathbf{i}})p(x_i'|\mathbf{x}_{\setminus\mathbf{i}}) && \text{(Bayes rule)} \\ &= p(x_i|\mathbf{x}'_{\setminus\mathbf{i}})p(\mathbf{x}'_{\setminus\mathbf{i}})p(x_i'|\mathbf{x}'_{\setminus\mathbf{i}}) && \text{(Remark)} \\ &= p(x_i|\mathbf{x}'_{\setminus\mathbf{i}})p(x_i',\mathbf{x}'_{\setminus\mathbf{i}}) && \text{(Bayes rule)} \\ &= p(\mathbf{x}')T(\mathbf{x}',\mathbf{x}) \end{aligned}$$

as required. Although this guarantees invariance of the target distribution it does not guarantee ergodicity of the Markov chain. Neal [14] argues that a sufficient condition of guaranteeing the ergodicity of the Markov chain is that the conditional distributions are not zero anywhere. Thus by updating one component each step, within a finite number of steps the chain starting from any point can reach any other point in the space.

3.4 Hamiltonian (Hybrid) Monte Carlo

In practice MH and Gibbs sampling perform poorly for convoluted posterior distributions which are the target distributions required for Bayesian inference on BNNs. In particular, MH has a tendency to exhibit random walk behaviour, exploring the target distribution too slowly. As for Gibbs sampling, it might be impossible to sample from the conditional distributions required. Furthermore, decomposing a state into local components not overly dependent on each other might be infeasible.

A more sophisticated and dynamic class of MCMC is Hamiltonian (Hybrid) Monte Carlo, which was first applied in Physics by (Duane et al., 1987). This was later adapted in the Statistical framework by Neal (1993 [14] and 1995 [1]) from which we will leverage the notation and results for this section. HMC simulates the motion of an imaginary particle using Hamilton's equations of motion, where the position of the particle is the target distribution. For the purpose of clarity, this section will explain how HMC works and how it can be implemented. However, a more detailed interpretation of the subject can be found in Neal (1993, and 1995).

3.4.1 Hamiltonian definitions and equations

Definition 5. Potential energy
This is the potential energy $U(\mathbf{q})$ of the fictitious physical system, where the variables of

interest (model parameters) are represented by the *position* variable, $\mathbf{q}$.

$$U(\mathbf{q}) = -\log(\pi(\mathbf{q})\mathcal{L}(\mathbf{q}|\mathcal{D})) + C$$

where $\mathcal{D}$ is the data, C is a constant, $\pi(\mathbf{q})$ is the prior distribution and $\mathcal{L}(\mathbf{q}|\mathcal{D}))$ is the likelihood function[1] given the data. In other words,

$$\begin{aligned} U(\mathbf{q}) &= -\log(\underbrace{p(\mathbf{q}|\mathcal{D})}_{\text{posterior}}) + C \\ &= -\log(\pi(\mathbf{q})) - \underbrace{\ell(\mathbf{q}|\mathcal{D})}_{\text{log-likelihood}} + C \\ &= -\log(\pi(\mathbf{q})) - \textstyle\sum_{i=1}^{n} \log\left(p(\mathbf{y_i}|\mathbf{x_i}, \mathbf{q})\right) + C \qquad (n \text{ i.i.d. observations}) \end{aligned}$$

with $\mathbf{y_i}$ being the true *label* or *output* and $\mathbf{x_i}$ the true input *data* for the i^{th} observation.

Definition 6. Kinetic energy
The *kinetic energy* $K(\mathbf{p})$, of the fictitious physical system, where $\mathbf{p}$ is the *momentum* variable. The kinetic energy helps the imaginary particle to overcome regions of local approximation solutions and thus explores the target distribution more freely. Typically the kinetic energy is assumed to have a *quadratic* form,

$$K(\mathbf{p}) = \frac{1}{2}\mathbf{p^T}\mathbf{M^{-1}}\mathbf{p}$$

where $\mathbf{M}$ is a symmetric and positive-definite matrix, which for simplicity is typically diagonal.

$$K(\mathbf{p}) = \sum_{i=1}^{d} \frac{p_i^2}{2m_i}$$

We assumed a quadratic form of the kinetic energy, because it corresponds to the log density of a zero-mean multivariate Gaussian distribution with covariance matrix $\mathbf{M}$ plus a constant, which is very easy to sample from.

Definition 7. Hamiltonian function
The Hamiltonian function, $H(\mathbf{q}, \mathbf{p})$, is an energy function for the joint state $(\mathbf{q}, \mathbf{p})$,

$$H(\mathbf{q}, \mathbf{p}) = U(\mathbf{q}) + K(\mathbf{p})$$

which defines the joint distribution, $p(\mathbf{q}, \mathbf{p})$, over $\mathbf{q}$ and $\mathbf{p}$,

$$p(\mathbf{q}, \mathbf{p}) = \frac{1}{Z_H} \exp\left(\frac{-H(\mathbf{q}, \mathbf{p})}{T}\right) \tag{3.1}$$

[1] The likelihood function $\mathcal{L}(\mathbf{q}|\mathcal{D}) = p(\mathbf{Y}|\mathbf{X}, \mathbf{q})$.

where T is referred to as the *temperature*, which in the canonical case is equal to 1. Z_H is the normalising factor so that $p(\mathbf{q}, \mathbf{p})$ is a probability distribution.

Definition 8. Hamilton's equations of motion
These equations determine the change of the position and momentum over time t,

$$\frac{dq_i}{dt} = \frac{\partial H}{\partial p_i}$$
$$\frac{dp_i}{dt} = -\frac{\partial H}{\partial q_i}$$

for $i \in \{1, 2, \ldots, d\}$, where d are the number of dimensions of the variables of interest.

Using the definitions of the Hamiltonian function and the Kinetic energy,

$$\frac{d\mathbf{q}}{dt} = \mathbf{M^{-1}p} \Rightarrow \frac{dq_i}{dt} = \frac{p_i}{m_i} \quad , \quad \forall i \in \{1, 2, \ldots, d\}$$
$$\frac{d\mathbf{p}}{dt} = -\frac{\partial U}{\partial \mathbf{q}} \Rightarrow \frac{dp_i}{dt} = -\frac{\partial U}{\partial q_i} \quad , \quad \forall i \in \{1, 2, \ldots, d\}$$

3.4.2 Leapfrog method

In the situation where we are using a NN the Hamiltonians will no longer be integrable, therefore we require the use of a discrete-time approximation method known as the '*leapfrog method*'. This computes the discrete-time approximate updates to $\mathbf{p}$ and $\mathbf{q}$ at small time step increments ϵ, known as the '*step size*'. A single leapfrog step[2] is,

$$\begin{aligned}
p_i\left(t + \tfrac{\epsilon}{2}\right) &= p_i(t) - \tfrac{\epsilon}{2}\frac{\partial U(\mathbf{q}(t))}{\partial q_i} && \text{(Half-step momentum update)} \\
q_i(t + \epsilon) &= q_i(t) + \epsilon\frac{p_i\left(t+\frac{\epsilon}{2}\right)}{m_i} && \text{(Full-step position update)} \\
p_i(t + \epsilon) &= p_i\left(t + \tfrac{\epsilon}{2}\right) - \tfrac{\epsilon}{2}\frac{\partial U(\mathbf{q}(t+\epsilon))}{\partial q_i} && \text{(Half-step momentum update)}
\end{aligned}$$

[2]Note that if the kinetic energy function is different to the quadratic function proposed earlier, then the full-step update to the position variable has to be appropriately adjusted to the derivative $\frac{dq_i}{dt}$.

3.4.3 HMC and leapfrog algorithms

Algorithm 3: Leapfrog algorithm

Inputs: $\mathbf{q}, \mathbf{p}, \epsilon, L, \tau$

$\mathbf{p} = \mathbf{p} - \frac{\epsilon}{2}\frac{\partial U}{\partial \mathbf{q}}$; // Half-step momentum update

for $i \in \{1, 2, \ldots, L\}$ **do**

  $\mathbf{q} = \mathbf{q} + \frac{\epsilon}{\tau}\mathbf{p}$; // Full-step position update

  if $i < L$ **then**

    $\mathbf{p} = \mathbf{p} - \epsilon\frac{\partial U}{\partial \mathbf{q}}$; // Full-step momentum update except at the end

  end

end

$\mathbf{p} = \mathbf{p} - \frac{\epsilon}{2}\frac{\partial U}{\partial \mathbf{q}}$; // Half-step momentum update at end of trajectory

output: $\mathbf{q}, \mathbf{p}$

Figure 3.3 – Where $\mathbf{q}$ is the position variable vector or matrix (depends on the parameters of the model), $\mathbf{p}$ is momentum variable and ϵ is the step size, and L is the number of leapfrog steps. Assumes that $K(\mathbf{p}) = \mathbf{p}^{\mathbf{T}}\mathbf{M}^{-1}\mathbf{p}$, for simplicity can assume $\mathbf{M} = \tau\mathbf{I}$.

Algorithm 4: HMC algorithm

Inputs: $n_{samp}, \epsilon, L, \tau$

initialise $\mathbf{q}^{(\mathbf{0})}$; // Initialise the position variables

for $i \in \{1, 2, \ldots, n_{samp}\}$ **do**

  $\mathbf{p}^{(\mathbf{i-1})} \sim \mathcal{N}(\mathbf{0}, \tau\mathbf{I})$; // Sample momentum from multivariate Gaussian

  $(\mathbf{q}^{(\mathbf{i})}, \mathbf{p}^{(\mathbf{i})}) = \mathrm{leapfrog}(\mathbf{q}^{(\mathbf{i-1})}, \mathbf{p}^{(\mathbf{i-1})}, \epsilon, L, \tau)$; // Perform leapfrog updates

  $\mathbf{p}^{(\mathbf{i})} = -\mathbf{p}^{(\mathbf{i})}$; // Negate momentum update to symmetrise the proposal

  $\alpha \sim U(0, 1)$; // Sample random number from Uniform(0,1)

  if $\alpha > \min\left(1, e^{-H(\mathbf{q}^{(\mathbf{i})}, \mathbf{p}^{(\mathbf{i})}) + H(\mathbf{q}^{(\mathbf{i-1})}, \mathbf{p}^{(\mathbf{i-1})})}\right)$ **then**

    $\mathbf{q}^{(\mathbf{i})} = \mathbf{q}^{(\mathbf{i-1})}$; // Reject proposal

    $\mathbf{p}^{(\mathbf{i})} = \mathbf{p}^{(\mathbf{i-1})}$; // Reject proposal

  end

end

output: $\{\mathbf{q}^{(\mathbf{i})}, \mathbf{p}^{(\mathbf{i})}\}_{i=1}^{n_{samp}}$

Figure 3.4 – Where $\mathbf{q}^{(\mathbf{i})}$ is the i^{th} sample of the position variable vector or matrix (depends on the parameters of the model), $\mathbf{p}^{(\mathbf{i})}$ is i^{th} sample of the momentum variable and ϵ is the step size, and L is the number of leapfrog steps. $\mathrm{leapfrog}()$ is the leapfrog algorithm defined in figure 3.3, and n_{samp} is the total number samples. Assumes that $K(\mathbf{p}) = \mathbf{p}^{\mathbf{T}}\mathbf{M}^{-1}\mathbf{p}$, for simplicity can assume $\mathbf{M} = \tau\mathbf{I}$.

The HMC algorithm shown above is a somewhat simplistic representation as it involves inference on only one parameter matrix/vector. In BNNs the posterior distribution (target

distribution) involves several parameters from which we want to sample from using HMC. However, this would be a trivial extension of the above algorithm performing the steps for each of the parameters, whilst holding the other parameters fixed.

3.4.4 Properties of the Hamiltonian system

Now that we have described how to perform HMC it is important to look at why HMC is a viable class of MCMC.

Remark
From the joint distribution $p(\mathbf{q}, \mathbf{p})$ over $\mathbf{q}$ and $\mathbf{p}$ in (3.1), we have that,

$$p(\mathbf{q}, \mathbf{p}) = \frac{1}{Z_H} \exp\left(\frac{-H(\mathbf{q}, \mathbf{p})}{T}\right) = \frac{1}{Z_H} \exp\left(\frac{-U(\mathbf{q})}{T}\right) \exp\left(\frac{-K(\mathbf{p})}{T}\right)$$

therefore $\mathbf{q}$ and $\mathbf{p}$ are independent.

Property 1. Conservation of the Hamiltonian
Through the progression of the system the value of $H(\mathbf{q}, \mathbf{p})$ remains constant,

$$\begin{aligned} \frac{dH}{dt} &= \sum_{i=1}^{d} \left(\frac{\partial H}{\partial p_i} \frac{dp_i}{dt} + \frac{\partial H}{\partial q_i} \frac{dq_i}{dt} \right) && \text{(chain rule)} \\ &= \sum_{i=1}^{d} \left(\frac{\partial H}{\partial q_i} \frac{\partial H}{\partial p_i} - \frac{\partial H}{\partial p_i} \frac{\partial H}{\partial q_i} \right) \\ &= 0 && \text{(Remark)} \end{aligned}$$

Property 2. Volume preservation
The volume $\mathbf{V}$ in the space $(\mathbf{q}, \mathbf{p})$ is preserved, meaning that the change in the volume is constant.

$$\mathbf{V} = \left(\frac{d\mathbf{q}}{dt}, \frac{d\mathbf{p}}{dt} \right)$$

$$\begin{aligned} \nabla \left(\frac{d\mathbf{q}}{dt}, \frac{d\mathbf{p}}{dt} \right) &= \frac{\partial}{\partial \mathbf{q}} \cdot \frac{d\mathbf{q}}{dt} + \frac{\partial}{\partial \mathbf{p}} \cdot \frac{d\mathbf{p}}{dt} && \text{(chain rule)} \\ &= \frac{\partial}{\partial \mathbf{q}} \cdot \frac{\partial H}{\partial \mathbf{p}} - \frac{\partial}{\partial \mathbf{p}} \cdot \frac{\partial H}{\partial \mathbf{q}} \\ &= \frac{\partial^2 H}{\partial \mathbf{q} \partial \mathbf{p}} - \frac{\partial^2 H}{\partial \mathbf{p} \partial \mathbf{q}} \\ &= 0 && \text{(Remark)} \end{aligned}$$

From the above two properties of the Hamiltonian dynamics we can see that $p(\mathbf{q}, \mathbf{p})$ is left invariant. To show that detailed balance holds we will use, but not prove two facts from (Neal, 1993 [14]),

1. The leapfrog scheme is **time-reversible**
2. The leapfrog scheme **preserves** the phase space **volume**

Proof that detailed balance holds for the canonical distribution (T = 1)
Let $\mathcal{S} = (\mathbf{q}, \mathbf{p})$ and $\mathcal{S}' = (\mathbf{q}', \mathbf{p}')$ denote some regions in the phase space. Suppose that after a sequence of L leapfrog steps of size ϵ we go from the region $\mathcal{S}$ in the phase space to the region $\mathcal{S}'$. We need to show that,

$$p(\mathcal{S})T(\mathcal{S}, \mathcal{S}') = p(\mathcal{S}')T(\mathcal{S}', \mathcal{S})$$

Since the leapfrog scheme preserves phase space volume, then if the volume of the phase space $\mathcal{S}$ is equal to V, the volume of the phase space $\mathcal{S}'$ must also equal to V.

$$\begin{aligned} p(\mathcal{S})T(\mathcal{S}, \mathcal{S}') &= \tfrac{1}{Z} V e^{-H(\mathcal{S})} \underbrace{\min\left(1, e^{-H(\mathcal{S}')+H(\mathcal{S})}\right)}_{p(\text{accept state } \mathcal{S}')} \\ p(\mathcal{S}')T(\mathcal{S}', \mathcal{S}) &= \tfrac{1}{Z} V e^{-H(\mathcal{S}')} \underbrace{\min\left(1, e^{-H(\mathcal{S})+H(\mathcal{S}')}\right)}_{p(\text{accept state } \mathcal{S})} \end{aligned}$$

Case 1. $H(\mathcal{S}) = H(\mathcal{S}')$
One can trivially see that the detailed balance equation holds.

Case 2. If $H(\mathcal{S}) > H(\mathcal{S}')$, then

$$p(\mathcal{S})T(\mathcal{S}, \mathcal{S}') = \frac{1}{Z} V e^{-H(\mathcal{S})}$$

Since the leapfrog scheme is reversible we can perform the scheme backwards in time,

$$\begin{aligned} p(\mathcal{S}')T(\mathcal{S}', \mathcal{S}) &= \tfrac{1}{Z} V e^{-H(\mathcal{S}')} e^{-H(\mathcal{S})+H(\mathcal{S}')} \\ &= \tfrac{1}{Z} V e^{-H(\mathcal{S})} \\ &= p(\mathcal{S})T(\mathcal{S}, \mathcal{S}') \end{aligned}$$

as required.

Case 3. If $H(\mathcal{S}) < H(\mathcal{S}')$, then

$$\begin{aligned} p(\mathcal{S}')T(\mathcal{S}', \mathcal{S}) &= \tfrac{1}{Z} V e^{-H(\mathcal{S}')} \\ p(\mathcal{S})T(\mathcal{S}, \mathcal{S}') &= \tfrac{1}{Z} V e^{-H(\mathcal{S})} e^{-H(\mathcal{S}')+H(\mathcal{S})} \\ &= \tfrac{1}{Z} V e^{-H(\mathcal{S}')} \\ &= p(\mathcal{S}')T(\mathcal{S}', \mathcal{S}) \end{aligned}$$

as required.

Similarly to MH and Gibbs sampling, ergodicity of the chain is once again problem dependent.

3.5 Variational Inference

As previously, our goal is to infer the true posterior distribution $p(\boldsymbol{\Theta}|\mathcal{D})$, where $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, ...\}$ (the set of all parameters), which cannot be done analytically. Variational inference (Jordan et al. 1999 [25]) involves the introduction of an approximating distribution parametrised by λ. The approximating/variational distribution $q_\lambda(\boldsymbol{\Theta})$ is constructed so that it can be easily inferred. The idea is that the approximating distribution is a close to the true distribution as possible, so we want to minimise the KL divergence (Kullback and Leibler, 1951 [26]) with respect to λ. Let us define the KL divergence to be,

$$\mathrm{KL}[q_\lambda(\boldsymbol{\Theta}) \parallel p(\boldsymbol{\Theta}|\mathcal{D})] = \int q_\lambda(\boldsymbol{\Theta}) \log\left(\frac{q_\lambda(\boldsymbol{\Theta})}{p(\boldsymbol{\Theta}|\mathcal{D})}\right) \mathbf{d}\boldsymbol{\Theta}$$

Hence we have that,

$$\begin{aligned}
\mathrm{KL}[q_\lambda(\boldsymbol{\Theta}) \parallel p(\boldsymbol{\Theta}|\mathcal{D})] &= \int q_\lambda(\boldsymbol{\Theta}) \log\left(\tfrac{q_\lambda(\boldsymbol{\Theta})}{p(\mathbf{Y}|\mathbf{X},\boldsymbol{\Theta})p(\boldsymbol{\Theta})}\right) \mathbf{d}\boldsymbol{\Theta} + \mathrm{KL}[q_\lambda(\boldsymbol{\Theta}) \parallel p(\boldsymbol{\Theta})] + c \\
&= -\int q_\lambda(\boldsymbol{\Theta}) \log p(\mathbf{Y}|\mathbf{X},\boldsymbol{\Theta}) \mathbf{d}\boldsymbol{\Theta} + \mathrm{KL}[q_\lambda(\boldsymbol{\Theta}) \parallel p(\boldsymbol{\Theta})] + c \\
&= -\textstyle\sum_{i=1}^{n} \int q_\lambda(\boldsymbol{\Theta}) \log p(\mathbf{y}_i|\mathbf{x}_i,\boldsymbol{\Theta}) \mathbf{d}\boldsymbol{\Theta} + \mathrm{KL}[q_\lambda(\boldsymbol{\Theta}) \parallel p(\boldsymbol{\Theta})] + c
\end{aligned}$$

For simplicity, only the variational mean field approximation will be considered (Peterson and Anderson, 1987 [24]), where $q_\lambda(\boldsymbol{\theta})$ is fully factorised, i.e. assuming independence of all the weight and bias scalars in every layer of the model,

$$q_\lambda(\boldsymbol{\Theta}) = \prod_{\boldsymbol{\theta} \in \boldsymbol{\Theta}} \left\{ \prod_{i,j \in (\mathcal{I}_\theta, \mathcal{J}_\theta)} q(\theta_{ij}) \right\}$$

where $(\mathcal{I}_\theta, \mathcal{J}_\theta)$ is the set of all nodes for the parameter $\boldsymbol{\theta}$. Factorising over everything, makes the computations very simple and quick, although the method struggles to reach accurate results for more complicated posterior distributions. Now as for the prediction using this methodology we define the predictive distribution $p(\hat{\mathbf{y}}|\mathbf{x}, \mathcal{D})$,

$$p(\hat{\mathbf{y}}|\mathbf{x}, \mathcal{D}) \approx \int p(\hat{\mathbf{y}}|\mathbf{x}, \boldsymbol{\Theta}) \hat{q}_\lambda(\boldsymbol{\Theta}) \mathbf{d}\boldsymbol{\Theta}$$

where $\hat{q}_\lambda(\boldsymbol{\Theta})$ is the argmin of the KL divergence objective. Due to the inability of this method to perform well on deeper networks we will only use this method of approximate inference in very few models. We will not explore any further mathematical details of this method, as they can be found in (Peterson and Anderson, 1987 [24], Hinton and van Camp, 1993 [23], and Jordan et al., 1999, [25]) in a much more detailed and complete form.

3.6 Prediction

As previously discussed, the predictive distribution is $p(\hat{\mathbf{y}}|\mathbf{x}, \mathcal{D})$, which is the expectation under the posterior distribution. However, an explanation regarding the estimation of those quantities is yet to be covered, as we cannot typically analytically evaluate the required integrals. We will now show the method of Monte Carlo integration used to make the approximations.

3.6.1 MC integration for VI

We can average all the predictions using the estimated samples obtained by sampling from the approximate posterior distribution, to obtain an approximation of the true predictions,

$$\begin{aligned}\frac{1}{T}\sum_{s=1}^{T} p(\hat{\mathbf{y}}|\mathbf{x}, \hat{\boldsymbol{\Theta}}_{\boldsymbol{s}}) \quad &\underset{T\to\infty}{\longrightarrow} \int p(\hat{\mathbf{y}}|\mathbf{x}, \boldsymbol{\Theta})\hat{q}_{\lambda}(\boldsymbol{\Theta})\mathbf{d}\boldsymbol{\Theta} \\ &\approx \int p(\hat{\mathbf{y}}|\mathbf{x}, \boldsymbol{\Theta})p(\boldsymbol{\Theta}|\mathcal{D})\mathbf{d}\boldsymbol{\Theta} \\ &= p(\hat{\mathbf{y}}|\mathbf{x}, \mathcal{D})\end{aligned}$$

where $\boldsymbol{\Theta}_s \overset{i.i.d.}{\sim} q_{\lambda}(\boldsymbol{\Theta})$, $\forall s = 1, 2, ..., T$.

3.6.2 MC integration for MCMC

We can average all the predictions using the estimated samples obtained by sampling from the approximate posterior distribution, to obtain an approximation of the true predictions,

$$\begin{aligned}\frac{1}{T}\sum_{s=1}^{T} p(\hat{\mathbf{y}}|\mathbf{x}, \hat{\boldsymbol{\Theta}}_{\boldsymbol{s}}) \quad &\underset{T\to\infty}{\longrightarrow} \int p(\hat{\mathbf{y}}|\mathbf{x}, \boldsymbol{\Theta})\hat{p}(\boldsymbol{\Theta}|\mathcal{D})\mathbf{d}\boldsymbol{\Theta} \\ &\approx \int p(\hat{\mathbf{y}}|\mathbf{x}, \boldsymbol{\Theta})p(\boldsymbol{\Theta}|\mathcal{D})\mathbf{d}\boldsymbol{\Theta} \\ &= p(\hat{\mathbf{y}}|\mathbf{x}, \mathcal{D})\end{aligned}$$

where $\boldsymbol{\Theta}_s \overset{i.i.d.}{\sim} \hat{p}(\boldsymbol{\Theta}|\mathcal{D})$, $\forall s = 1, 2, ..., T$. Where $\hat{p}(.)$ denotes the MCMC approximation to the true posterior distribution.

MC integration is well-defined, provided that $p(\hat{\mathbf{y}}|\mathbf{x}, \boldsymbol{\Theta})$ is absolutely continuous and thus integrable. Please note that in the rest of the dissertation we refer to the predictive approximations derived by using MC integration as MC estimates, and the MC integration as the MC estimator.

Chapter 4

Regularisation methods

Regularisation is broad area of methodologies that are primarily aimed at reducing the overfitting incurred by the model during the training stage. In machine learning language, overfitted models are said to not "generalise" well. This essentially means that the difference between the expected error and the empirical/training error is large and thus the model would perform well on the training set and poorly on the test set. Such a model would be inaccurate for out of sample predictions. Regularisation is thus a means of penalising over adaptation to the training data.

For Frequentists regularisation could be as simple as adding a penalty term on the size of the weights into the objective function, L1 and L2 regularisation are explored in more detail in section 4.1. A more sophisticated approach to regularisation in NNs is to use the method known as Dropout, the variants of this method will be explored in section 4.3. The Bayesian use of a prior distribution incorporates regularisation in the underlying modelling method, in fact some different prior distributions can have an equivalent regularisation representation in the Frequentist scheme. Bayesian linear regression with Gaussian priors can be equivalently expressed as L2 regularised linear regression. This equivalence will be explored in more detail in section 4.2.

4.1 Frequentist Regularisation

Let us consider the simple example of linear regression, with data $\mathcal{D} = \{\mathbf{y}, \mathbf{X}\}$, where $\mathbf{y}$ is an $(n \text{ x } 1)$ vector of the responses and $\mathbf{X}$ is a $(n\text{x}d)$[1] matrix of covariates, and $\mathbf{w}$ is the $(d\text{x}1)$ vector of weights, with last weight corresponding to that for the bias term. Suppose that we are interested in minimising the squared loss, then the objective function is

[1] For the simplicity of the calculations we assume that the last column of $\mathbf{X}$ is a column with all ones, which will represent the bias vector.

$$\min_{\mathbf{w}} \quad (\mathbf{y} - \mathbf{X}\mathbf{w})^{\mathbf{T}}(\mathbf{y} - \mathbf{X}\mathbf{w})$$

This objective function is likely to over-fit the data in the scenario where d is large.

L1 regularisation

L1 regularisation imposes an L1-norm penalty term $\|\mathbf{w}\|_1$ on the weights, so that they are inhibited from becoming too large and hence the new objective function is,

$$\min_{\mathbf{w}} \quad (\mathbf{y} - \mathbf{X}\mathbf{w})^{\mathbf{T}}(\mathbf{y} - \mathbf{X}\mathbf{w}) + \lambda \|\mathbf{w}\|_1 \tag{4.1}$$

where $\lambda > 0$ is the regularisation constant. Notice that in order to solve the above objective we would be required to either make a smooth/continuous approximation to the absolute value function, or to instead use one of the gradient approximation methods seen earlier, such as Adam, SGD or NAG. A possible smooth approximation could be,

$$|x| \approx \sqrt{x^2 + \epsilon} \quad \text{for } \epsilon > 0$$

where ϵ is a very small positive constant. The smaller ϵ is, the closer the approximation we would be to the absolute value function.

L2 regularisation

Similarly L2 regularisation imposes an L2-norm penalty term $\|\mathbf{w}\|_2^2$ on the weights, in order to constraint their absolute size. The objective function in this case is,

$$\underset{\mathbf{w}}{\operatorname{argmin}} \quad (\mathbf{y} - \mathbf{X}\mathbf{w})^{\mathbf{T}}(\mathbf{y} - \mathbf{X}\mathbf{w}) + \lambda \mathbf{w}^{\mathbf{T}}\mathbf{w} \tag{4.2}$$

for $\lambda > 0$. Since the objective is positive semi-definite there is a unique minimum[2], we can easily solve the objective by differentiating with respect to the vector $\mathbf{w}$ and setting the derivative equal to the zero vector $\mathbf{0}$, yielding the solution,

$$\hat{\mathbf{w}} \quad = \quad (\mathbf{X}^{\mathbf{T}}\mathbf{X} + \lambda \mathbf{I_d})^{-1}\mathbf{X}^{\mathbf{T}}\mathbf{y} \tag{4.3}$$

Notice that the value of λ would impact the solution we obtain and it is therefore a parameter that needs to be tuned. The most common approach of tuning this parameter is through cross-validation. This is a method for picking the λ with minimum cross-validation error, by minimising the MSE over several different subsets of held out portions of the data (validation sets) not used for training.

[2]The proof for this as argument is not provided as it is just a simple application of algebra and Mercer's theorem, and twice differentiating the objective function.

4.2 Bayesian Regularisation

In the Bayesian framework the prior function helps to inhibit the weights from growing too large and thus acts as a means for regularisation. In the simple context of linear regression we can actually show equivalence of Laplace priors with L1 regularisation, and Normal priors with L2 regularisation. Assuming that the data are i.i.d. Gaussian, then the likelihood for the data has the form,

$$p(\mathbf{y}|\mathbf{X},\mathbf{w}) \quad = \quad \frac{1}{Z_1}\exp\left(\frac{-\sigma^2}{2}(\mathbf{y}-\mathbf{X}\mathbf{w})^{\mathbf{T}}(\mathbf{y}-\mathbf{X}\mathbf{w})\right)$$

Let us assume that all the weights are i.i.d Laplace with mean 0 and scale 1, then the prior on the weights $p(\mathbf{w})$ has the form,

$$p(\mathbf{w}) = \frac{1}{Z_2}\exp(-\left\|\mathbf{w}\right\|_1)$$

Then the "Maximum a-posteriori" (MAP) estimate of the weights $\mathbf{w}^{\mathbf{MAP}}$ is defined below,

$$\begin{aligned}
\mathbf{w}^{\mathbf{MAP}} &= \underset{\mathbf{w}}{\operatorname{argmax}} \quad p(\mathbf{w}|\mathcal{D}) \\
&= \underset{\mathbf{w}}{\operatorname{argmax}} \quad \tfrac{1}{Z}\exp\left(\tfrac{-\sigma^2}{2}(\mathbf{y}-\mathbf{X}\mathbf{w})^{\mathbf{T}}(\mathbf{y}-\mathbf{X}\mathbf{w})\right)\exp(-\left\|\mathbf{w}\right\|_1) \\
&= \underset{\mathbf{w}}{\operatorname{argmax}} \quad \log\left(\tfrac{1}{Z}\exp\left(\tfrac{-\sigma^2}{2}(\mathbf{y}-\mathbf{X}\mathbf{w})^{\mathbf{T}}(\mathbf{y}-\mathbf{X}\mathbf{w})\right)\exp(-\left\|\mathbf{w}\right\|_1)\right) \\
&= \underset{\mathbf{w}}{\operatorname{argmax}} \quad \tfrac{-\sigma^2}{2}(\mathbf{y}-\mathbf{X}\mathbf{w})^{\mathbf{T}}(\mathbf{y}-\mathbf{X}\mathbf{w}) - \left\|\mathbf{w}\right\|_1 + C \\
&= \underset{\mathbf{w}}{\operatorname{argmin}} \quad \tfrac{\sigma^2}{2}(\mathbf{y}-\mathbf{X}\mathbf{w})^{\mathbf{T}}(\mathbf{y}-\mathbf{X}\mathbf{w}) + \left\|\mathbf{w}\right\|_1 \\
&= \underset{\mathbf{w}}{\operatorname{argmin}} \quad (\mathbf{y}-\mathbf{X}\mathbf{w})^{\mathbf{T}}(\mathbf{y}-\mathbf{X}\mathbf{w}) + \tfrac{2}{\sigma^2}\left\|\mathbf{w}\right\|_1
\end{aligned}$$

which is equivalent to the objective in (4.1), with $\lambda = \frac{2}{\sigma^2}$.

If we assume that the weights are i.i.d. $w_i \sim \mathcal{N}(0,1)$, $\forall i \in \{1,2,...,d\}$, then the prior distribution on the weights $p(\mathbf{w})$ has the form,

$$p(\mathbf{w}) = \frac{1}{Z_3}\exp(-\frac{1}{2}\mathbf{w}^{\mathbf{T}}\mathbf{w})$$

thus the MAP estimate for the parameters is,

$$\mathbf{w}^{\mathbf{MAP}} \quad = \quad \underset{\mathbf{w}}{\operatorname{argmin}} \quad (\mathbf{y}-\mathbf{X}\mathbf{w})^{\mathbf{T}}(\mathbf{y}-\mathbf{X}\mathbf{w}) + \tfrac{1}{\sigma^2}\mathbf{w}^{\mathbf{T}}\mathbf{w}$$

which is equivalent to the objective in (4.2), with $\lambda = \frac{1}{\sigma^2}$.

4.3 Dropout

We will now take a step back and think about the reason for regularisation, which is to prevent the extensive adaptation to the training data. Large NNs have an astronomical number of parameters that need to be learnt, making the threat of their co-adaptation highly probable, particularly in the absence of a sufficient amount of training data. As discussed, regularisation allows the model to perform better in unobserved cases by capturing the underlying patterns in the data, as opposed to the dataset specific noise (sampling noise).

Dropout (Hinton et al., 2012 [34]; Srivastava et al., 2014 [35]) works by corrupting the learning process with random noise, prohibiting the model from simply memorising the training data. Dropout has been shown to achieve state-of-the-art results in many machine learning tasks and benchmark datasets, making it the go to regularisation technique currently practised in Machine learning. MC Dropout may also be useful in providing good approximations the "true" posterior distribution, a result which follows from (Myshkov and Julier, 2016 [41]) analysis on synthetic data.

For a dataset $\mathcal{D} = (\mathbf{X}, \mathbf{y})$, where $\mathbf{X} \in \mathbb{R}^{n \times d}$ and $\mathbf{y} \in \mathbb{R}^{n \times 1}$. Linear regression minimising the squared loss finds $\hat{\mathbf{w}}$ such that,

$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmin}} \quad (\mathbf{y} - \mathbf{X}\mathbf{w})^{\mathbf{T}}(\mathbf{y} - \mathbf{X}\mathbf{w})$$

where $\mathbf{w} \in \mathbb{R}^{d \times 1}$ is the weight vector. Suppose that we apply dropout to the input data $\mathbf{X}$, s.t. we keep each x_{ij} with probability p. Let $\mathbf{B} \in \{0, 1\}^{n \times d}$, such that $b_{ij} \overset{i.i.d.}{\sim} \mathrm{Ber}(p)$. In the context of the linear regression objective minimisation,

$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmin}} \quad (\mathbf{y} - \mathbf{B} \odot \mathbf{X}\mathbf{w})^{\mathbf{T}}(\mathbf{y} - \mathbf{B} \odot \mathbf{X}\mathbf{w})$$

Taking the expectation under the probability distribution assumed for $\mathbf{B}$, we have

$$\begin{aligned} \left\langle \|\mathbf{y} - \mathbf{B} \odot \mathbf{X}\mathbf{w}\|^{\mathbf{2}} \right\rangle_{\mathbf{B} \sim \mathrm{Ber}(\mathbf{P})} &= \langle \mathbf{y}^{\mathbf{T}}\mathbf{y} - \mathbf{2}\mathbf{y}^{\mathbf{T}}(\mathbf{B} \odot \mathbf{X})\mathbf{w} + \mathbf{w}^{\mathbf{T}}(\mathbf{B} \odot \mathbf{X})^{\mathbf{T}}(\mathbf{B} \odot \mathbf{X})\mathbf{w} \rangle_{\mathbf{B} \sim \mathrm{Ber}(\mathbf{P})} \\ &= \mathbf{y}^{\mathbf{T}}\mathbf{y} - 2\mathbf{y}^{\mathbf{T}} \langle \mathbf{B} \rangle_{\mathbf{B} \sim \mathrm{Ber}(\mathbf{P})} \odot \mathbf{X}\mathbf{w} \quad + \quad M \\ &= \mathbf{y}^{\mathbf{T}}\mathbf{y} - 2p\mathbf{y}^{\mathbf{T}}\mathbf{X}\mathbf{w} \quad + \quad M \end{aligned}$$

where, $M = \mathbf{w}^{\mathbf{T}} \langle (\mathbf{B} \odot \mathbf{X})^{\mathbf{T}}(\mathbf{B} \odot \mathbf{X}) \rangle_{\mathbf{B} \sim \mathrm{Ber}(\mathbf{P})} \mathbf{w}$
We note that,

$$(\mathbf{B} \odot \mathbf{X})^{\mathbf{T}}(\mathbf{B} \odot \mathbf{X}) = \begin{pmatrix} \sum_{i=1}^{n} x_{i1}^2 b_{i1}^2 & \dots & \sum_{i=1}^{n} x_{id} x_{i1} b_{id} b_{i1} \\ \vdots & \ddots & \vdots \\ \sum_{i=1}^{n} x_{i1} x_{id} b_{i1} b_{id} & \dots & \sum_{i=1}^{n} x_{id}^2 b_{id}^2 \end{pmatrix}$$

Hence,

$$\left\langle (\mathbf{B} \odot \mathbf{X})^{\mathbf{T}} (\mathbf{B} \odot \mathbf{X}) \right\rangle_{\mathbf{B} \sim \mathrm{Ber}(\mathbf{P})} = \begin{pmatrix} \sum_{i=1}^{n} x_{i1}^2 \left\langle b_{i1}^2 \right\rangle & 0 & \dots & 0 \\ 0 & \ddots & & \vdots \\ \vdots & & & 0 \\ 0 & \dots & 0 & \sum_{i=1}^{n} x_{id}^2 \left\langle b_{id}^2 \right\rangle \end{pmatrix}$$

since $\forall i \in \{1, 2, ..., n\}$, $\langle b_{ij} b_{im} \rangle = \begin{cases} 0 & , \text{ if } j \neq m \\ p(1-p) + p^2 & , \text{ if } j = m \end{cases}$

by the independence of the Bernoulli random variables.

Our objective function thus reduces to,

$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmin}} \quad \|\mathbf{y} - p\mathbf{X}\mathbf{w}\|_2^2 + p(1-p) \|\mathbf{\Gamma}\mathbf{w}\|_2^2$$

Where $\mathbf{\Gamma} = \mathrm{diag}(\mathrm{diag}(\mathbf{X}^{\mathbf{T}}\mathbf{X}))^{1/2}$, and thus scaling the weight vector by the standard deviation of the corresponding d dimensions of the input data. This objective looks quite similar to the objective we found for L2 regularisation. If we let p tend towards 1 we can see that the regularisation constant tends towards 0 and since we tend towards retaining all the inputs the objective tends to the least squares objective. Equivalently for $\tilde{\mathbf{w}} = p\mathbf{w}$ we have,

$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmin}} \quad \|\mathbf{y} - \mathbf{X}\tilde{\mathbf{w}}\|_2^2 + \frac{1-p}{p} \|\mathbf{\Gamma}\tilde{\mathbf{w}}\|_2^2$$

Another way of applying Dropout (Konda et al., 2015 [36]), is to drop the units as before, but rescaling the active units by a factor of $1/p$ to compensative for the loss of magnitude. The objective in that case is,

$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmin}} \quad \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 + \frac{1-p}{p} \|\mathbf{\Gamma}\mathbf{w}\|_2^2$$

which can directly be related to a form of but not equivalent to L2 regularisation. Baldi and Sadowski (2013, [43]) argued that stochastic gradient descent performed on a model with dropout was similar to stochastic gradient descent on a regularised error function. The reason we say that it is not equivalent to L2 regularisation is because this is objective has a superior penalty term incorporating the variability of the data and the probability p of dropping a unit. N.B. that we showed how Bayesian linear regression with Gaussian priors can be equivalent to L2 regularisation, and therefore can be similar to the Dropout objective, but not equivalent.

Note that MC Dropout would then take the MC estimate of all the predictions generated by a number (N) of dropout fitted models. This is essentially an ensemble of models fitted using dropout. Alternatively, one can use the point estimate method of dropout, which is

to fit the weights of dropout models stochastically using gradient descent, and adjusting the predictions by a factor $1/p$. Although the two methods are not equivalent, in practice they provide quite similar results. MC dropout starts to become more accurate through an increasing number of samples, which can be computationally expensive.

4.4 Alternative view on Dropout

To keep with the theme of this chapter a Bayesian interpretation of what Dropout is in the context of linear regression will be attempted below. We have previously argued that Dropout is similar to concepts that we can grasp, such as L2 regularisation and thus some form of Bayesian linear regression with Gaussian priors, however varying in its penalty term. Let us revisit Bayesian linear regression and see the graphical model for this,

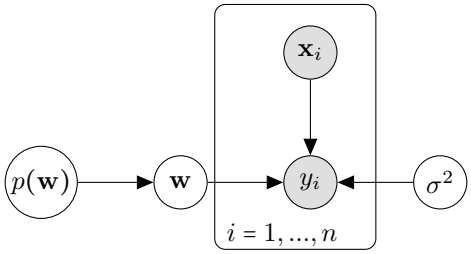

Figure 4.1 – Bayesian linear regression, where $\mathbf{w}, \mathbf{x}_i \in \mathbb{R}^d$, and we assume that the labels (y_i) are i.i.d. s.t. $y_i \sim \mathcal{N}(\mathbf{w^T x_i}, \sigma^2), \forall i \in \{1, 2, ..., n\}$. The prior distribution on the weights $p(\mathbf{w})$ could be a multivariate Gaussian $\mathcal{N}(\mathbf{0}, \mathbf{I})$.

Baldi and Sadowski (2014, [44]) discussed the "ensemble averaging properties" of linear networks with dropout and non-linear logistic networks. With this interpretation in mind we will show in what ways Dropout is dissimilar to the regularisation methods that we have previously seen. In particular, two possible interpretations of Bayesian Dropout will be provided as well as showing how the graphical model for Dropout based linear regression in a Bayesian framework differs from that of regular Bayesian linear regression.

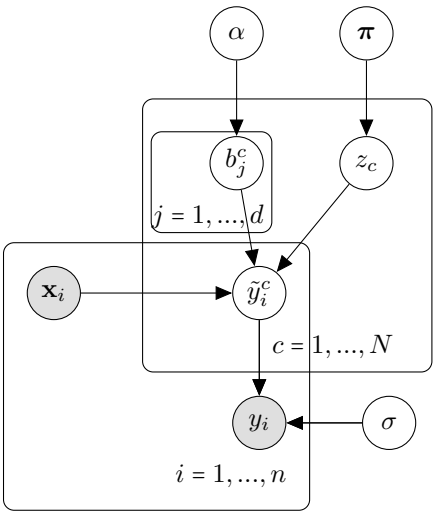

Figure 4.2 – (A) First Bayesian interpretation of Dropout, where $\mathbf{x}_i \in \mathbb{R}^d$, N is the number of Dropout repetitions used to form the Monte Carlo estimate for y_i. The categorical distribution z_c, determines the parameters of the distribution of $\tilde{y}_i^c$ (see below for details).

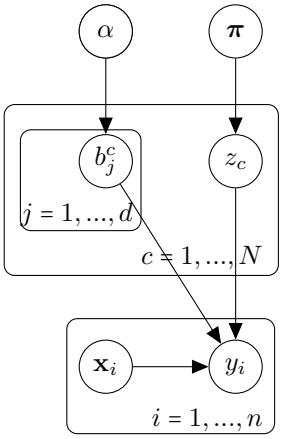

Figure 4.3 – (B) Second Bayesian interpretation of Dropout, where $\mathbf{x}_i \in \mathbb{R}^d$, N is the number of Dropout repetitions. $\boldsymbol{\pi}$ is the prior distribution to $z_c, \forall c \in \{1, 2, ..., N\}$, and α is the parameter of the Bernoulli distribution (dropout probability for keeping a node).

4.4.1 Model equations for interpretation (A)

$b_j^c \overset{\text{i.i.d.}}{\sim} \text{Ber}(\alpha)$
$z_c \overset{\text{i.i.d.}}{\sim} \text{Categorical}(\boldsymbol{\pi})$
$\tilde{y}_i^c \overset{\text{i.i.d.}}{\sim} \mathcal{N}(\mathbf{w}_{z_c}^T(\mathbf{b}^c \odot \mathbf{x}_i), \sigma_{z_c}^2)$, for $\mathbf{w}_{z_c}, \mathbf{b}^c \in \mathbb{R}^d$
$y_i \overset{\text{i.i.d.}}{\sim} \mathcal{N}\left(\sum_{c=1}^N \pi_{z_c} \tilde{y}_i^c, \sigma^2\right)$

In order to be fully Bayesian about this problem we can define a Dirichlet hyper-prior on the Categorical distribution, for simplicity we will just assume that $\boldsymbol{\pi} = \left(\frac{1}{N}, \frac{1}{N}, ..., \frac{1}{N}\right)$, meaning that all data transformations are equally likely to occur. *For the explicit definitions of all the distributions used please visit page 8.*

4.4.2 Model equations for interpretation (B)

We can think of α as the Dropout probability for keeping an input, we assume that this is a scalar.

$b_j^c \overset{\text{i.i.d.}}{\sim} \text{Ber}(\alpha)$
$z_c \overset{\text{i.i.d.}}{\sim} \text{Categorical}(\boldsymbol{\pi})$
$y_i \overset{\text{i.i.d.}}{\sim} \sum_{c=1}^N \pi_{z_c} \mathcal{N}(\mathbf{w}_{z_c}^T(\mathbf{b}^c \odot \mathbf{x}_i), \sigma_{z_c}^2) = \sum_{c=1}^N \pi_{z_c} \mathcal{N}(\mathbf{w}_{z_c}^T \mathbf{x}_i \text{diag}(\mathbf{b}^c), \sigma_{z_c}^2)$

Again in order to be fully Bayesian we might want to define a Dirichlet hyper-prior on the Categorical distribution.

From the two Bayesian interpretations provided we can see that Dropout is more similar to a mixture model which is the probabilistic interpretation of ensembles. This mixture model imposes a non-linear transformation of the input data $\mathbf{X}$ for each mixture component, and different weight and standard deviations for each of the model mixtures. This should make sense, as gradient descent on the transformed data would learn new weights for each dropout repetition. The predictions from all the different models are then averaged (ensemble averaging), to give the MC estimate for the predictions. We can see clearly that this is very different to Bayesian linear regression as the model averaging that takes place has a much richer hypothesis space. In response to a paper by Domingos, (Minka, 2002 [42]) explains the differences between model averaging and Bayesian model averaging, where he emphasised that the two are not comparable.

Gal (2017, [2]) on the other hand argues that for specific variational approximations, the optimisation for VI is identical to that of dropout on a NN. This might possibly be the case, however the similarities between the two are rather vague, especially since we do not get any indication of what the priors of such a NN are. Recent Dropout advancements leverage variational inference (Kingma et al., 2015 [47]; Molchanov et al., 2017 [48]; Li and Gal, 2017 [49]).

I believe that comparing dropout on a NN fitted using gradient descent with the equivalent BNN is simply erroneous as the two models are too different. Konda et al., (2015 [36]) highlighted that dropout enriches the input space and can be perceived to be similar to "data augmentation", which could explain the much richer Bayesian model representation required to imitate MC dropout. I believe that if we were to instead compare the posterior distribution of MC dropout, to the posterior of the BNN interpretations shown, we would observe similar results presuming that the method for estimating the posterior works well. Unfortunately, much of the advancement in the Machine learning community stemmed from business applications which reduced the interest in theory grounded work. Many algorithms such as dropout are known to work well in practice, yet the theory justifying their success is somewhat incomplete and offers a great potential for further exploration in the future.

Chapter 5

Application to data

We will now show the results after applying the different types of models discussed in this dissertation. We aim to provide an unbiased comparison of the different models and methods of estimation by choosing to show the results for the most tuned models.

Choosing the right parameters can often be very difficult and time consuming in practice, something that was experienced first hand during this dissertation. For the BNNs the hyper-parameters shown are after a thorough search over a vast variety of possible hyper-parameters. Similarly for the number of steps and step sizes used for the leapfrog integration method. Again, for the learning rates used for the feed-forward NNs a thorough search over a number of different possible learning rates was conducted.

Unless otherwise stated, all the models in this chapter were ran on an NVIDIA GeForce® GTX 1070 GPU and therefore any algorithm times are relative to that. In practice one should expect between 10-50 times increase in the computation time if these models are ran on a CPU, but these differences in speed depend on the model itself, with CNNs benefiting the most from GPU computation.

5.1 Datasets

We are currently living in an era defined by extensive and rich data, with classification being at the heart of many fields. A revolution in the way machines operate is therefore to be expected. For instance, facial recognition and object detection will be a crucial element to self driving cars and robots. Therefore our choice to look at the classical MNIST digits dataset was quite natural. When conducting experiments, we used the MNIST dataset that comes with the "Tensorflow" python package. This has 55000 training points, of which there are 55000 (28x28) pixel images and 55000 training labels, taking values from 0 to 9. With this dataset we are also given 10000 test points, again the images are (28x28) pixels and the labels take values from 0 to 9. We will call this dataset (MNIST large). For all the models with the exception of the CNNs we reduce the image size to (14x14) so that the models can

be ran more easily.

Most of our recorded experiments were conducted using the (MNIST small) datasets, which is a dataset constructed by taking the first 2000 training and first 2000 testing examples from the MNIST large dataset. The small MNIST dataset or the short script used to create the dataset can be found in [56]. When fitting the models, the image size was reduced to (14x14) due to the relatively large models' size, which would otherwise cease to make any storing of the samples generated using the HMC process possible. Even storing as little as 100 samples sometimes resulted in files that were over 120 megabytes in size!

Subsequently, a slight variation of the Mauna Loa CO_2 concentrations dataset was used for regression (Gal, 2015 [50]), which was used for several uncertainty models by (Gal and Ghahramani, 2016 [45]). We changed the dataset from the original by combining the test and training datasets and taking a random 2/3 split of the data for training (848 examples) and 1/3 split of the data for testing (424 examples). The dataset and the script used to create the data can be found in [56]. Note that this dataset has 1-dimensional inputs and 1-dimensional outputs.

The code for a 1 hidden layer BNN was also adapted for the solar regression dataset (Gal, 2016 [45]), however to avoid repetition we will not present the results obtained in this dissertation. Presuming the reader is interested further, please refer to the code found in [56].

5.2 Feed-forward NNs (Frequentist models)

These are classical NN models fitted using a supervised learning procedure through some loss minimisation. For the experiments 4 different types of optimisers were used, Stochastic Gradient Descent (SGD), Momentum, Nesterov's accelerated gradient (NAG), and Adam. After conducting several experiments using Momentum and NAG, it was decided to drop these from the analysis as they did not perform as well as Adam. SGD being the conventional gradient descent method was also chosen. Even though SGD lacked the ability to achieve as accurate results as Adam, due to the fact that it got trapped in local minima we will observe that the learning phase of this algorithm is very smooth.

5.2.1 NN with 1 hidden layer

In this section we will explore 1 hidden layer models fitted using back propagation. We will see the models being applied to both a classification and a regression case. Throughout this section a selection of graphics will be used, however for the classification case, our focus

will be directed at the learning curves (test and training errors during the training).

Only the notable and interesting cases will be presented in this study. All the MNIST models in this section were trained stochastically, i.e. using mini-batch learning to compute the back propagation gradients for batches of 200 observations, and cycling through the whole dataset n number of times (epochs). All the CO_2 regression models were trained stochastically with batches of size 212.

MNIST digits dataset (classification)

When we briefly introduced our interest in BNNs we emphasised their ability to perform well even in the scenario where we lack the sufficient amount of data, whereas ordinary NNs could over-fit. The situation of a small amount of data is not rare, and might become even more common as we move into the future, where we will seek machines that are be able to identify and learn in real time without much data. We therefore entirely focus on the Small MNIST dataset for these models.

Hidden units	Epochs	Accuracy (Adam)	Accuracy (SGD)	Time (seconds)
500	1000	0.831	0.831	30
500	3000	0.853	0.826	86
500	5000	0.856	0.821	145
800	1000	0.830	0.832	30
800	3000	0.857	0.824	88
800	5000	0.853	0.823	151
1000	1000	0.834	0.827	30
1000	3000	0.860	0.826	91
1000	5000	0.848	0.821	151
2000	1000	0.826	0.831	33
2000	3000	0.854	0.823	100
2000	5000	0.862	0.825	168
3000	1000	0.824	0.831	32
3000	3000	0.853	0.826	86
3000	5000	0.856	0.821	145

Figure 5.1 – *The table contains the best selected results from training 1 hidden layer NNs using back-propagation. The accuracy corresponds to the final accuracy on the test set. All models used a learning rate of 0.005 as this was found to be overall the best learning rate for the 1 hidden layer models. The time refers to the time for training the models only and excludes the model evaluation and other computations, it is also the average between the time taken for the SGD and Adam optimisers for the specific model setup. All the models use Softplus non-linearities for the hidden layer.*

Training NNs using back propagation is remarkably fast! Even for the networks with 3000 hidden units and using 5000 epochs for the training the models were trained in less than 3 minutes. This puts them at a significant advantage in comparison to the Bayesian approaches which are typically considerably slower than this. However, please note the presented findings were after we fitted numerous models and selected the "best" learning rates. As seen in figure 5.1 the test accuracy of these models is also sufficiently high, although for less tuned models the accuracy can be significantly lower than this.

As we had expected, using Adam as the optimiser results in significantly more accurate models than using SGD. We also see that SGD suffers more from over-fitting to a specific local minimum trajectory, which in many cases resulted in a decreasing test accuracy with more training (epochs). From the plots in figure 5.2 a much more obvious observation can be made, for the 1000 epochs the models are clearly under trained as the test accuracy seems to have a positive gradient at the end of the 1000 epochs, whereas for the 5000 epochs the models appear to have taken a trajectory worsening the test accuracy with time (over-fitting to a specific incorrect local minimum). Slightly after the 1000 epochs the test accuracy starts decreasing, whilst the training accuracy continues to increase, this is a classical example of over-fitting and also one of the primary reasons that we are exploring Bayesian alternatives.

We observed that the in terms of training accuracy Adam converges extremely quickly, although the test accuracy increases at a slower rate. The figures also emphasise that Adam seems to consistently find better local minimum approximations than SGD. SGD struggles to distinguish the "correct" direction of the gradient and can instead get trapped in an incorrect trajectory. This can sometimes lead to over-fitting to the training data, whilst test accuracy decreases (see figure 5.2).

Without trial and error we cannot know in hindsight what the "correct" amount of training could be for these models, which poses many difficulties in practice, and in particular if we seek some sort of automation in these models. We can always use the test accuracy during training as a proxy to the right amount of training, automating a stop in the model if the test accuracy decreases, but this could only be easily applied to SGD, where we have a much smoother learning curve. Adam has much more unpredictable results, as the "hill-climbing" properties of the optimiser mean that there are sections of the learning curves where we observe highly negative gradients in the accuracy, followed by a sharp spike, automating a smart stop in these models can be intrinsically difficult.

Mauna Loa dataset (Regression)

Regression is much easier to interpret than classification, as we can actually plot the trained model and see what is in fact happening when we change the model. Using back propaga-

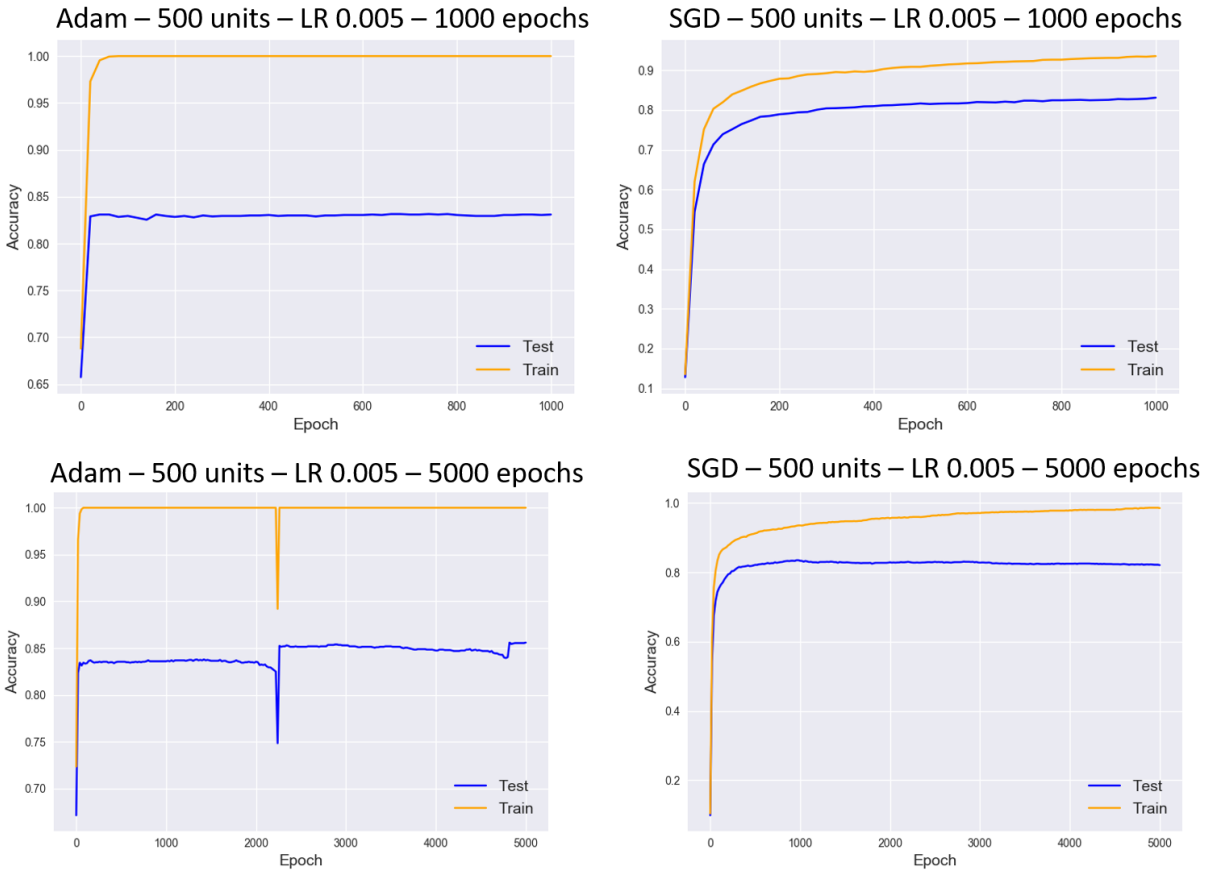

Figure 5.2 – *The learning curves during training of models fitted using back propagation and the optimiser indicated in the title of the plot. All the models composed of 1 hidden layer of 500 units. The number of epochs is indicated in the title of the plot, along with the learning rate (LR) that was used. All the models use a Softplus rectifier for the hidden layer.*

tion on this dataset posed the same problems as before, primarily that the parameters used required extensive tuning before they could actually work well.

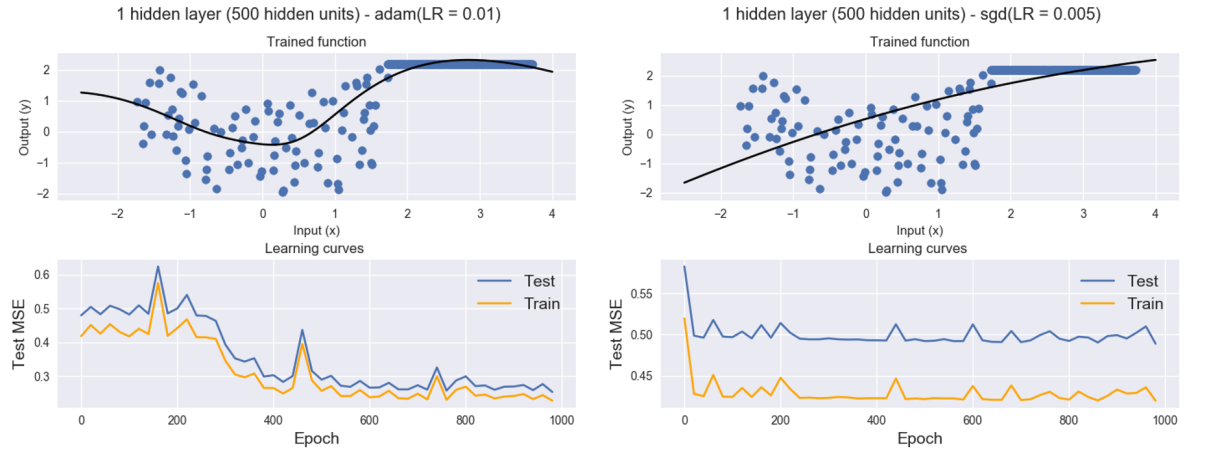

Figure 5.3 – *The learning curves during training of models fitted using back propagation and the optimiser indicated in the title of the plot (bottom plots). All the models composed of 1 hidden layer of 500 units. LR refers to the learning rate that was used for the optimiser. The plots on the top are the learned functions from the corresponding model and the scatter points are the test data.*

From figure 5.3 it is apparent that the model using the Adam optimiser was not trained for long enough, as the learning curve seems to be continuing to decrease. The learned function, also seems to be missing the structure of the data around the extremes. SGD seems to be trapped in a local minimum and thus performs poorly, fitting almost a straight line function to the data. This observation seems to be confirmed by looking at figure 5.4, where even for 5000 epochs the result is the same. For 5000 epochs we observe that the learned function using Adam is considerably better and seems to capture the structure of the data reasonably well.

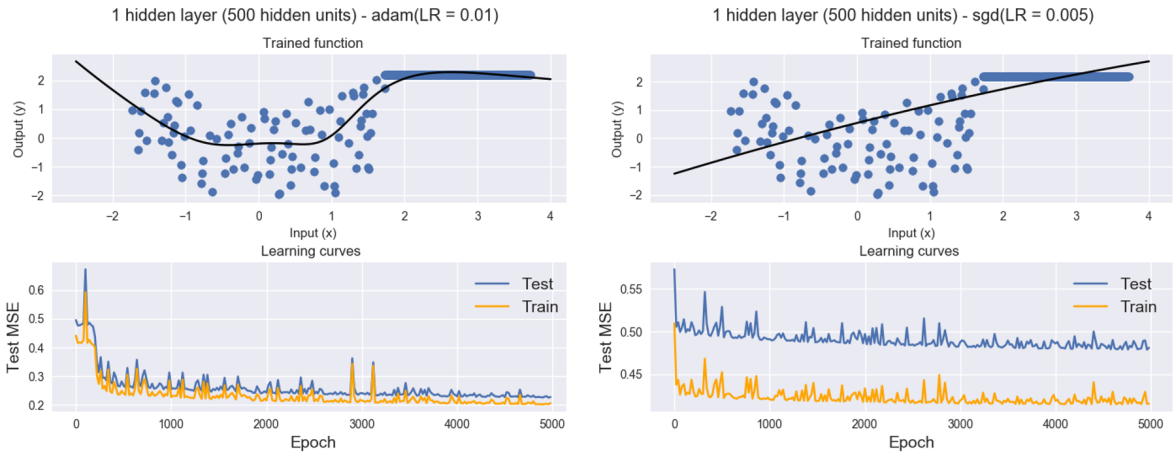

Figure 5.4 – *The learning curves during training of models fitted using back propagation and the learned functions (same as figure 5.3, but trained for longer).*

Naturally, increasing the size of the models to 5000 hidden units was expected to require more training, which is something that is emphasised by figure 5.6. Clearly SGD has not been trained for long enough. We expect that this also holds for the model with the Adam optimiser, but this is not evident from the learning curves, which seem somewhat flat with several spikes. Not being able to identify whether the algorithm has converged causes difficulties in practice. This problem is a recurring theme for models fitted using back propagation, whereas BNNs using MCMC such as HMC benefit from gathering more samples. That means that we can ultimately continue running our models indefinitely without the risk of over-fitting, although this is very costly in practice.

It is important to note that for the models with 5000 hidden units we were required to change the learning rates used. The need to find an optimal learning rate again can occur frequently in practice, which can make the use of such methods more impractical. Finally, to ensure that the models were trained for long enough we also trained the models for 50000 epochs (see figure 5.7). Here we can see that the model with the Adam optimiser shows a slight improvement. However, near the extremes the model deviates from the “true” structure of the data.

In order to avoid exploding gradients when using SGD, a much smaller learning rate was

Hidden units	Epochs	Adam (MSE)	SGD (MSE)	η_{adam}	η_{sgd}	Time (s)
500	1000	0.2529	0.4889	0.01	0.005	7
500	3000	0.2363	0.4816	0.01	0.005	19
500	5000	0.2282	0.481	0.01	0.005	32
1000	1000	0.2711	0.4302	0.01	0.005	7
1000	3000	0.2978	0.3436	0.01	0.005	19
1000	5000	0.2516	0.3448	0.01	0.005	29
3000	1000	0.4074	0.5945	0.01	0.001	7
3000	3000	0.2771	0.5732	0.01	0.001	22
3000	5000	0.2466	0.5432	0.01	0.001	35
5000	1000	0.4547	0.4945	0.005	0.001	8
5000	5000	0.3082	0.3652	0.005	0.001	37
5000	50000	0.2237	0.3022	0.005	0.001	365

Figure 5.5 – *The table contains the best selected results from training 1 hidden layer NNs using back-propagation. The MSE corresponds to the final mean squared error on the test set. The time refers to the time for training the models only and excludes the model evaluation and other computations, it is also the average between the time taken for the SGD and Adam optimisers for the specific model setup.* η_{adam} *and* η_{sgd} *are the learning rates used for the Adam and SGD optimisers respectively. All the models use Softplus non-linearities for the hidden layer.*

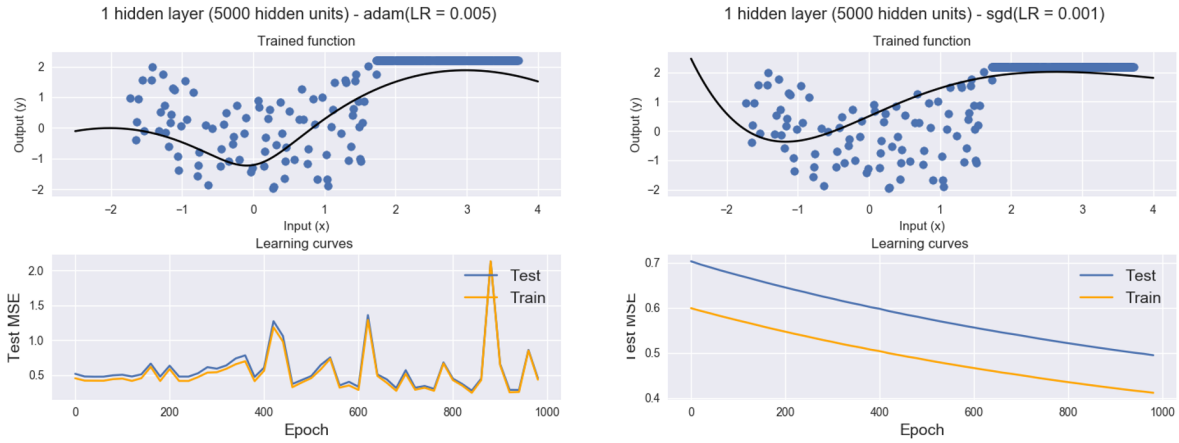

Figure 5.6 – *The learning curves during training of models fitted using back propagation and the learned functions, both models have 1 hidden layer with 5000 units.*

used (0.001), this meant that even for 50000 epochs the models were significantly under-trained. Slow convergence is an underlying problem of the SGD optimiser, and it is reflected in these figures.

5.2.2 NN with 2 hidden layers

In the interest of exploring larger models we look at models with 2 hidden layers. For the reduced MNIST images (14x14), a model with 500 hidden units has over 350000 parameters

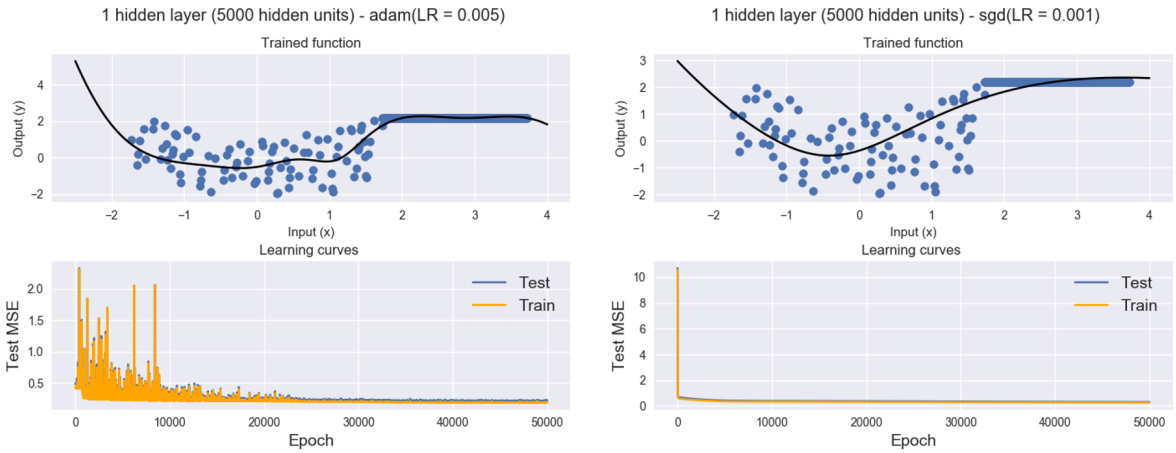

Figure 5.7 – *The learning curves during training of models fitted using back propagation and the learned functions, both models have 1 hidden layer with 5000 units and were trained for 50000 epochs.*

that need to be trained and the danger of over-fitting is now much more severe. All the MNIST and CO_2 regression models in this section were trained stochastically using batches of size 200, and 212 respectively.

MNIST digits dataset (classification)

Returning back to the classification setting, we can see from figure 5.8 that SGD suffers the most from following an incorrect trajectory. As we increase the amount of training time the test accuracy falls. The models using the Adam optimiser perform surprisingly well and in most cases do not seem to suffer from adverse effects of excessive training of the NNs. In the appendix § C.2.1 we can again see the same pattern of SGD finding a worse local minimum and Adam being much less smooth, which brings us to the question of how and which optimiser should be used. Once we know the optimiser, what learning rate should be used? These are all interesting questions, inherent in the field of Machine learning, yet they have no right answer. Different optimisers might be better at specific tasks, making model tweaking profoundly difficult.

Mauna Loa dataset (Regression)

In these experiments, using SGD as an optimiser produced really poor results as the optimiser achieved extremely slow convergence (see figure 5.9) in comparison to Adam. Once again SGD failed to capture the true underlying structure of the data, which is no surprise given that it was not completely trained.

Hidden units	Epochs	Adam	SGD	Time (seconds)
250	1000	0.833	0.8195	30
250	3000	0.852	0.813	90
250	10000	0.842	0.798	308
300	1000	0.8345	0.836	30
300	3000	0.85	0.8115	92
300	10000	0.843	0.8085	306
500	1000	0.843	0.8285	32
500	3000	0.8475	0.8165	97
500	10000	0.865	0.81	318

Figure 5.8 – *The table contains the best selected results from training 2 hidden layer NNs using back-propagation. The accuracy corresponds to the final accuracy on the test set. Like previously, all models used a learning rate of 0.005. The time refers to the time for training the models only and excludes the model evaluation and other computations, it is also the average between the time taken for the SGD and Adam optimisers for the specific model setup. All the models use Softplus non-linearities for the hidden layer.*

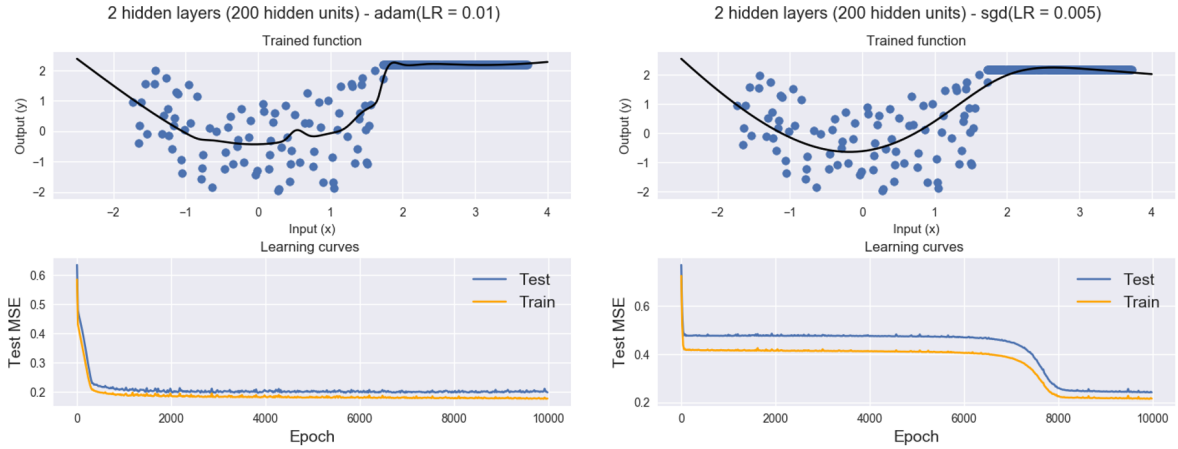

Figure 5.9 – *The learning curves during training of models fitted using back propagation and their learned functions, both models consist of 2 hidden layers each with 200 units.*

5.2.3 CNN with 2 convolutional layers and a hidden layer

For this final section we explore stochastic batch training using batches of size 100, in order to avoid computing infinite gradients. The smaller batch size helps to make the training process much smoother with less numerical instability. CNNs are image type models and thus have only been applied to the MNIST dataset. As with all the previous sections we will use the small MNIST dataset, however the images are left unreduced (28x28).

We begin to see that as the models become increasingly deeper the over-fitting problem becomes more severe. Furthermore, the choice of learning rate becomes crucial to the resulting model's ability to classify correctly. As is clear from figures 5.13 and 5.11, using a learning rate of 0.01 for Adam converges very quickly to the optimum (in less than 1000

Hidden units	Epochs	Adam (MSE)	SGD (MSE)	Time
200	1000	0.2058	0.484	7
200	5000	0.202	0.4608	30
200	10000	0.1989	0.2423	61
300	1000	0.2032	0.4654	6
300	5000	0.2006	0.434	32
300	10000	0.1975	0.2448	59
500	1000	0.2529	0.4889	7
500	5000	0.2282	0.481	32
500	10000	0.2262	0.4036	65

Figure 5.10 – *The table contains the best selected results from training 2 hidden layer NNs using back-propagation. The MSE corresponds to mean squared error on the test set. For the Adam optimiser we use the learning rate of 0.01 and for SGD a learning rate of 0.005. The time refers to the time for training the models only and excludes the model evaluation and other computations, it is also the average between the time taken for the SGD and Adam optimisers for the specific model setup. All the models use Softplus non-linearities for the hidden layer.*

epochs) and then begins to radically over-fit. Lowering the learning rate to 0.005 still results in the networks to over-fit if trained for too long, but the decrease in the test accuracy is less extreme (see figures 5.14 and 5.12).

Epochs	Accuracy (Adam)	Accuracy (SGD)	Time (s)
1000	0.9145	0.91	71
3000	0.9315	0.9015	212
5000	0.9085	0.9035	348
10000	0.1045	0.905	690

Figure 5.11 – *The table contains the results from training CNNs with 2 x (convolutional & maxpool) layers followed by a fully connected layer and a hidden layer with 250 units (see figure 1.3 in § 1.2). The accuracy corresponds to the accuracy on the test set. A learning rate of 0.01 was used both for Adam and SGD. The time was computed in the same way as previously. All the models use Softplus non-linearities for the hidden layer.*

Epochs	Accuracy (Adam)	Accuracy (SGD)	Time (s)
1000	0.9255	0.905	70
3000	0.9285	0.911	212
5000	0.9355	0.9085	352
10000	0.9275	0.9135	696

Figure 5.12 – *Identical to fig. 5.11, apart from a learning rate of 0.005 used both for Adam and SGD.*

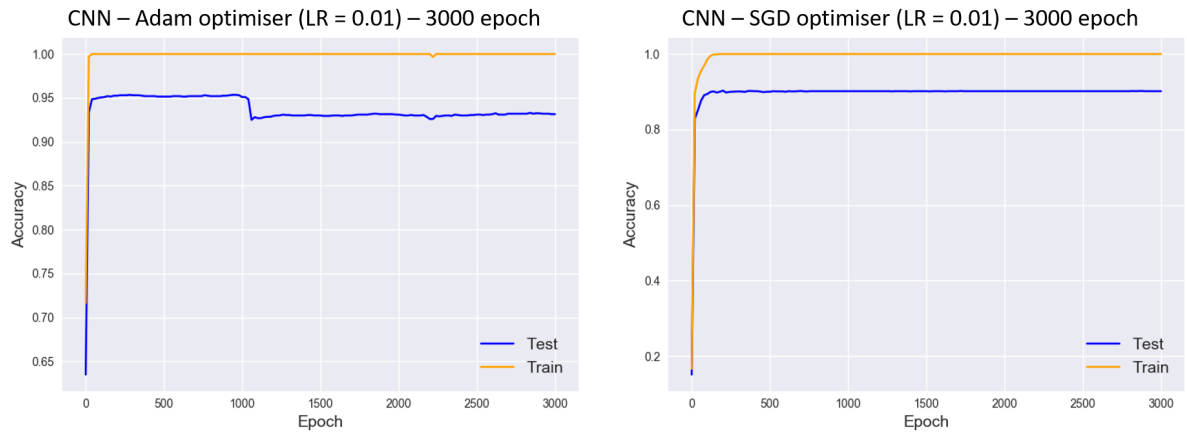

Figure 5.13 – *The learning curves during training of models fitted using back propagation and their learned functions, both CNNs consisted of 2 (convolutional & maxpool), a fully connected layer and a hidden layer with 250 units (see figure 1.3 in § 1.2). The models were trained for 3000 epochs and the hidden layer used Softplus non-linearities.*

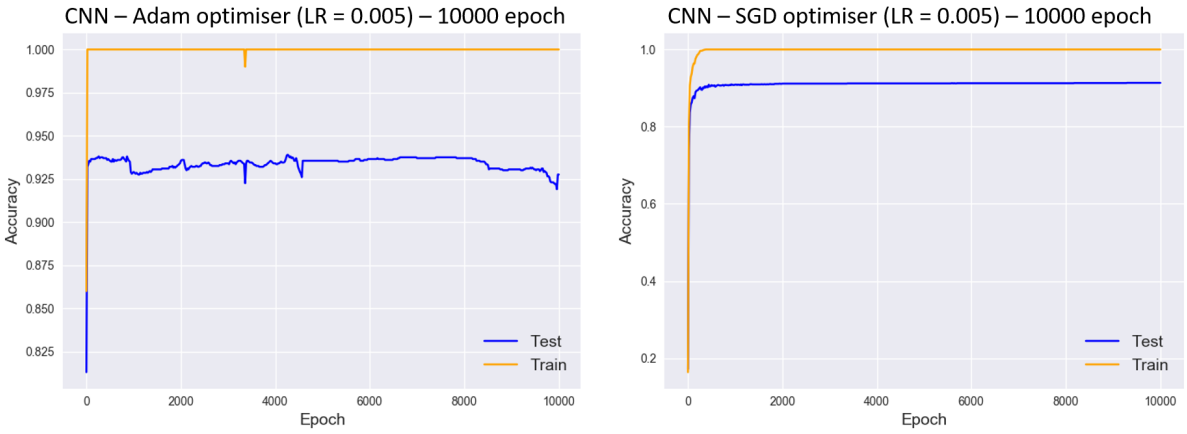

Figure 5.14 – *The learning curves during training of models fitted using back propagation and their learned functions, using identical models to figure 5.13, but trained for 10000 epochs and using a learning rate of 0.005 for both the Adam and SGD optimisers.*

5.3 Bayesian models

This section explores various types of Bayesian Neural Networks (BNNs) primarily fitted using HMC. We will explore the characteristics of the different models and discuss their advantages and drawbacks. For a principled comparison, we will use the same datasets and model structure that was explored in § 5.2. Due to computational difficulties we will not see all the models that were explored by feed-forward NNs being applied to the Bayesian framework.

When storing the "decorrelated" samples generated using MCMC we store 1 sample every n_{eff} samples defined as,

$$n_{\text{eff}} = \frac{N}{\text{ESS}}$$

where N is the total number of samples generated from the MCMC procedure, and ESS is the effective sample size (Ripley, 1987 [60]) defined as follows,

$$\mathrm{ESS} = \frac{N}{1 + 2\sum_{k=1}^{M} \rho_k}$$

where ρ_k are the auto-correlations of the particular sample for lag k, and M is the largest integer s.t. $\rho_k > 0, \forall k \in \{1, 2, ..., M\}$. Because we have more than 1 variable which we sample we take a sub-sample of J variables $(w_1, w_2, ..., w_J)$ and compute the average sum of all their positive auto-correlations,

$$n_{\mathrm{eff}} = 1 + \frac{2}{J} \sum_{j=1}^{J} \sum_{k=1}^{M} \rho_k(w_j)$$

This is defined as such for practicality purposes as the MCMC process is quite inefficient due to poor mixing. It might have been more sensible to define n_{eff} to be,

$$n_{\mathrm{eff}} = 1 + 2 \max_{j \in \{1,...,J\}} \left(\sum_{k=1}^{M} \rho_k(w_j) \right)$$

but this would mean that we would discard more samples that were very costly to generate in the first place. As a consequence of our choice of n_{eff} we will obtain slightly more correlated samples.

5.3.1 BNN with 1 hidden layer

Using Student T distribution priors proved to have poor results, and for degrees of freedom lower than 1.5 the HMC algorithm was almost impossible to tune with an acceptance rate of 0 even at the lowest learning rates (0.00005). This in turn implied that we could not generate any samples from the posterior distribution as all the samples were rejected. For 1.5 degrees of freedom we were able to achieve sensible acceptance rates and train the models by sampling from the approximate posterior distribution.

Interestingly, using Stochastic Gradient HMC (Chen et al., 2014 [57]) worked sufficiently well for the 1.0 degrees of freedom of the Student T distribution priors. We believe that our models might have been too large, and the T distribution priors resulted in a posterior distribution that was difficult to sample from (due to the heavy tails) and thus having a zero acceptance rate. Using batches of the data to compute the gradients, enabled us to sample more efficiently from the posterior distribution. Alternatively, it could have been more sensible to only use the T prior distributions on less wide networks, or use them only on the final layer and using Gaussian distributions for the other layers.

The tuning of the hyper-parameters (step size and number of leapfrog iterations) of HMC was difficult and costly. However, the hyper-parameters that did not work were identified quickly as the acceptance rates were very close to 0. A more efficient way to implement

HMC would have been to use the No-U-Turn Sampler (Hoffman and Gelman, 2014 [61]), which removes the need to tune the learning rate and the step sizes for HMC, but the implementations of HMC in "pymc3" (Salvatier et al., 2016 [62]), were considerably slower than the alternative "Edward" (Tran et al., 2017 [63]) which uses "Tensorflow" (Abadi et al., 2015 [64], 2016 [65]).

We will now take a look at a selection of models. However, due to a resource limitation we could not explore every case, and in particular we could not explore as large models as in the Frequentist section. The computation times reported refer only to the collection of samples (training) and do not explicitly reflect the actual computation time of the script itself. The script has additional features and involves regular model evaluation (every 50 samples) during the training, the collection of statistics, and storing the samples and statistics during training to the disk to avoid losing work from interrupted models (this step is extremely inefficient). These additional features are predominantly ran on the CPU rather than the GPU which is substantially slower. The actual running time of the script itself is about 2-4 times more than the time required to collect the samples (depending on the amount of CPU usage for other processes, and the size of the model). The larger the model is, the larger the sample files stored to the disk are, with some files being as large as 6GB!

Classification (1 hidden layer)

Hidden units	Prior Distribution	No. of samples	ESS	MC accuracy $\pm 1\sigma_{\mathrm{mc}}$	Point est. accuracy	Time (mins)
500	$\mathrm{Laplace}(0, \frac{10^2}{d_j})$	45000	196	0.862 ± 0.009	0.553	75.7
500	$\mathcal{N}(0, \frac{15^2}{d_j})$	45000	199	0.857 ± 0.009	0.104	75.3
500	$\mathrm{T}(\nu = 1.5, s = \frac{10^2}{d_j})$	45000	213	0.821 ± 0.009	0.8045	75.4
500	$\mathrm{T}(\nu = 1, s = \frac{10^2}{d_j})$	45000*	148	0.811 ± 0.002	0.81	9
500	$\mathrm{T}(\nu = 1, s = \frac{10^2}{d_j})$	75000*	255	0.834 ± 0.003	0.834	14.7
800	$\mathrm{Laplace}(0, \frac{15^2}{d_j})$	45000	218	0.861 ± 0.0122	0.148	100.7
800	$\mathcal{N}(0, \frac{13^2}{d_j})$	45000	218	0.855 ± 0.033	0.09	100.2
2000	$\mathrm{Laplace}(0, \frac{15^2}{d_j})$	45000	203	0.855 ± 0.033	0.29	171.8
2000	$\mathcal{N}(0, \frac{20^2}{d_j})$	45000	147	0.852 ± 0.009	0.12	171.9

Figure 5.15 – *BNNs with 1 hidden layer on the small MNIST dataset. For * the samples were generated using SGHMC, for all other models the samples were generated using HMC. s in the Student T distribution refers to the scale variable, the mean is taken to be 0. d_j refers to the number of inputs from the previous layer (same as the rescaling used in chapter 2). ESS refers to the effective sample size, and the number of samples refers to all the samples from the MCMC scheme including the burnin period. The point estimate accuracy refers to the accuracy from the model fitted using the average parameters from all samples.*

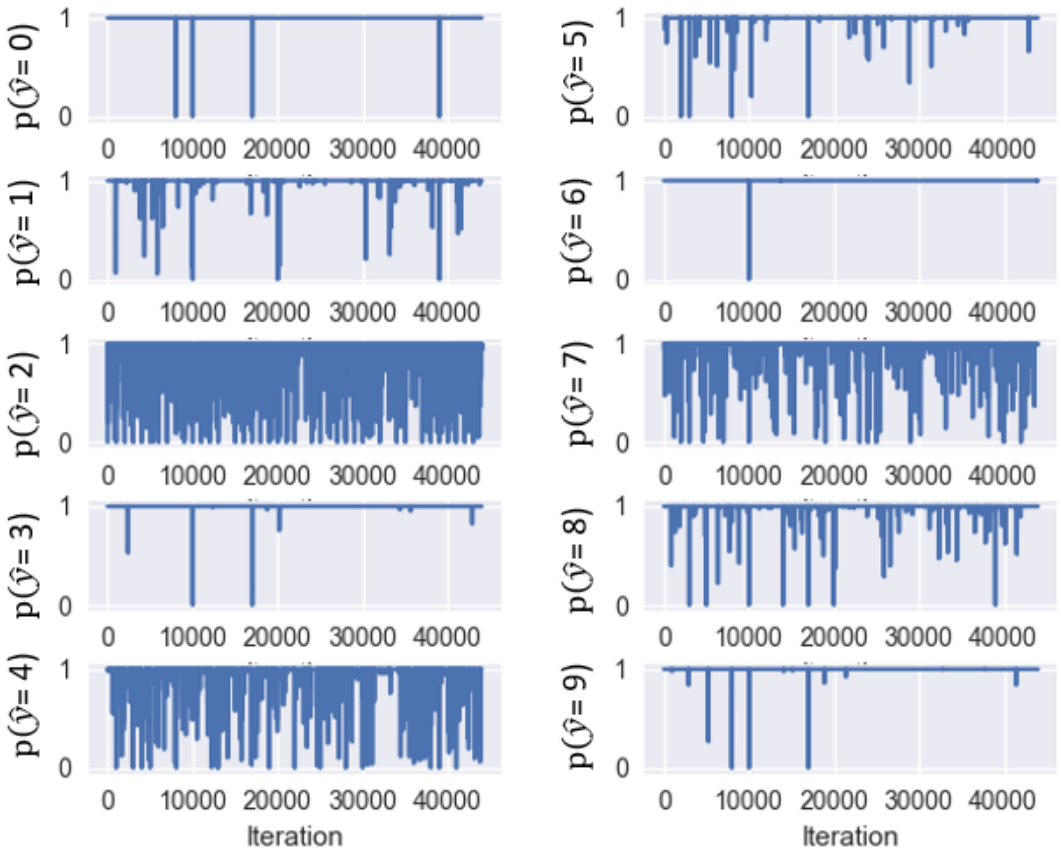

Figure 5.16 – *The trace of the probabilities of each of the 10 digits in the first instance of the true labels for these digits in the dataset, using the BNN with 1 hidden layer (2000 units) with* $\mathcal{N}(0, \frac{20^2}{d_j})$ *prior over all the parameters.*

In terms of accuracy of prediction, these models are quite similar to the feed-forward NNs, but they are much less influenced by the initialisation of the algorithm, which was commonly observed for the models minimising a loss function. However, the computation time for the BNNs using HMC can take up to 100 times longer. For these small models where our predictive distribution is highly confident (low variance) a Bayesian approach might have not been necessary. However, in more complicated models knowing the variance of the predictor can be vital in preventing dangerous decisions in practice.

From figure 5.15 we can see that the Laplace distribution priors seem to perform better than the other prior distributions used. The difference between the Normal and Laplace priors is however insignificant, which could indicate that both models are under-fitting. In general the predictive distribution when using a Laplace prior was observed to have slightly less variance. A key observation from figure 5.16 is that the model is perhaps "unidentifiable", as the output probabilities do not all seem to have converged for all the samples taken, with digits 2, 4 and 7 suffering the most. This imposes practical difficulties (Silva and Gramacy, 2010 [51]). Our MCMC scheme might be required to produce many more samples in order to obtain some usable uncorrelated samples, which can be interpreted as "slow

mixing" and this in turn can result in a greatly inefficient sampling procedure.

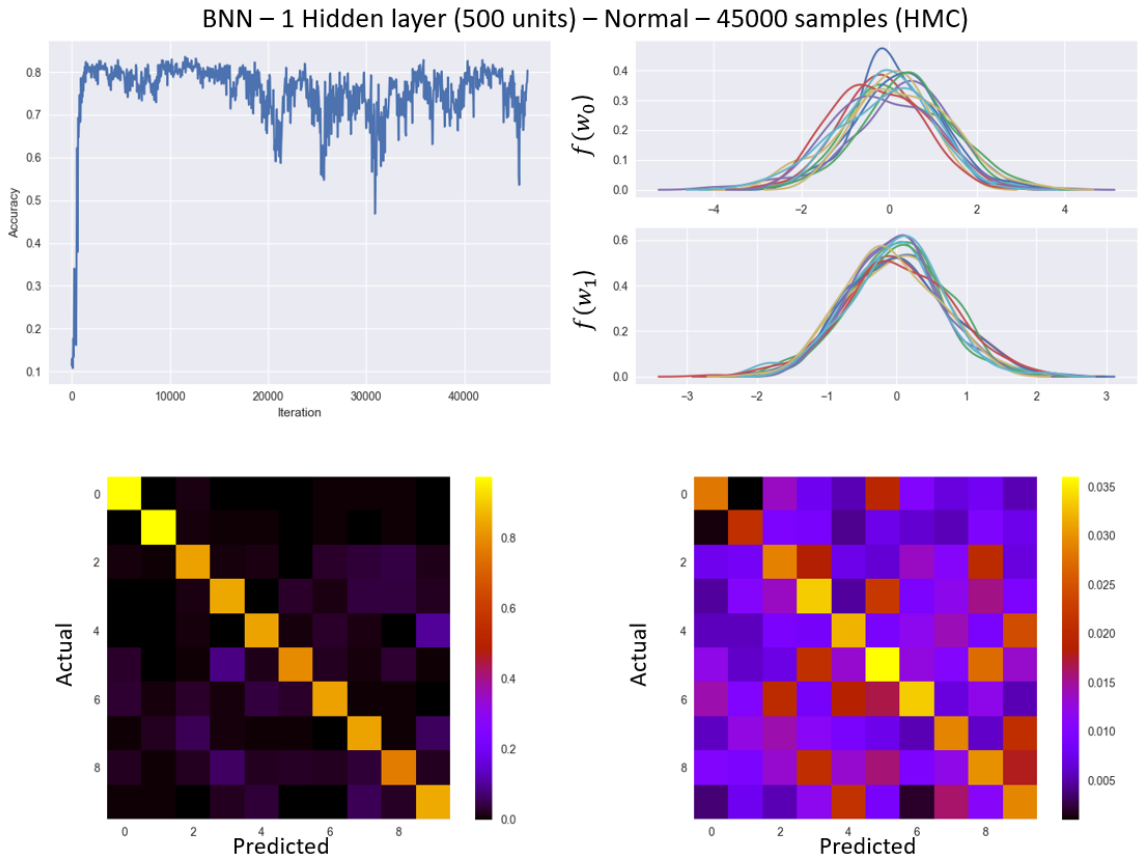

Figure 5.17 – *Diagnostic and predictive plots for the 1 hidden layer NN with 500 hidden units and* $\mathcal{N}(0, \frac{15^2}{d_j})$ *priors over all the parameters, using HMC to sample from the posterior. (Top left) A plot of the accuracy during sampling on mini-batches of samples. (Top right) Marginal distribution plots for a sample of weights. (Bottom left) Confusion matrix of the MC estimated model. (Bottom right) Confusion matrix standard deviations of the MC estimate.*

The highly correlated samples also arise due to the fact in the models with 1 hidden layer the step size for the leapfrog method was roughly around 0.001 and a trajectory of length 70 was used, which meant that the HMC dynamics proposed new samples quite close to the previous sample, and so on. However, such a low step size was necessary to obtain an acceptance rate that was non-zero.

For the model with 500 hidden units and $\mathcal{N}(0, \frac{15^2}{d_j})$ priors (figure 5.17), it is clear that most of the weights are centred around zero, with some slightly to the left or right. Bearing in mind that in the first layer we have 98000 weights, from which we have drawn only a very small sample (12 weights), this type of structure was to be expected in such a sparse dataset.

From the confusion matrix we can see that our model predicts digits 0 and 1 correctly the most and the most errors seem to be incurred for digits 5 and 8. The most variable prediction probability is observed for the digit 5. Digits 5 and 3 are most commonly confused with each other. $p(\hat{y} = 1|y = 0)$ and $p(\hat{y} = 0|y = 1)$ are both almost identical to zero, and the extremely

low variance of these estimated probabilities indicates that our model is highly confident in these estimates. This is a positive outcome demonstrating that our models are producing sensible results. The fact that 3 and 5 are confused is not surprising, especially given the fact that we pre-process the images shrinking them from (28x28) to (14x14), making it highly probable that some of the real 3s are being rendered to look like 5s and vice versa.

Hidden units	Prior Distribution	No. of iterations	MC accuracy $\pm 1\sigma_{\text{mc}}$ (500 samples)	Time (mins)
500	$\text{Laplace}(0, \frac{10^2}{d_j})$	5000	0.827 ± 0.020	1.1
500	$\mathcal{N}(0, \frac{20^2}{d_j})$	5000	0.865 ± 0.006	1
1000	$\mathcal{N}(0, \frac{20^2}{d_j})$	10000	0.844 ± 0.014	2.2
1000	$\text{Laplace}(0, \frac{15^2}{d_j})$	10000	0.852 ± 0.011	2.3
2000	$\text{Laplace}(0, \frac{15^2}{d_j})$	10000	0.867 ± 0.010	2.6
2000	$\mathcal{N}(0, \frac{20^2}{d_j})$	10000	0.856 ± 0.013	2.3
3000	$\text{Laplace}(0, \frac{15^2}{d_j})$	10000	0.869 ± 0.010	2.8
3000	$\mathcal{N}(0, \frac{20^2}{d_j})$	10000	0.866 ± 0.013	2.6

Figure 5.18 – *BNNs with 1 hidden layer on the small MNIST dataset. The samples were drawn using a Variational Mean Field approximation. s and d_j are defined as before. The MC accuracy refers to accuracy using MC integration for 500 samples from the posterior distribution.*

The reader should not be alarmed that the point estimate accuracies for most of the models are much lower than the MC prediction accuracy. In fact this emphasises that our sampled probability marginal distributions for the weights are quite rich and do not contain significantly correlated samples. In figure 5.19 we illustrate this difference on a simple example where we have 2 Gaussians both centred at zero but with different variance. The point estimate $\hat{\theta}$ would be the mean, which cannot distinguish between the two distributions. In the NN sense, for the point estimate we would take the average weights and forward feed them through the network, which as expected performs worse than the MC prediction.

Interestingly, using the SGHMC algorithm to generate samples leads to the second layer weights (weights from the hidden layer to output) to converge quickly (see figure 5.20). Similarly most of the first layer weights also seem to converge with the exception of one of the weights. This might suggest that the MCMC procedure might have not been run for long enough. The marginal distributions of the weights vary significantly from each other, and seem to be quite skewed. Note that some of the weights are centred around very large values (-60) or (-20), which might suggest why using the T distribution with $\nu = 1$ did not work with standard HMC, as the models are very likely to suffer from exploding gradients, or vanishing gradients due to the accumulation of either highly positive inputs or highly negative inputs to the non-linear, or softmax layers and thus propose new samples that are always rejected. Like before, the predictive plots seem quite sensible, although the variance of the

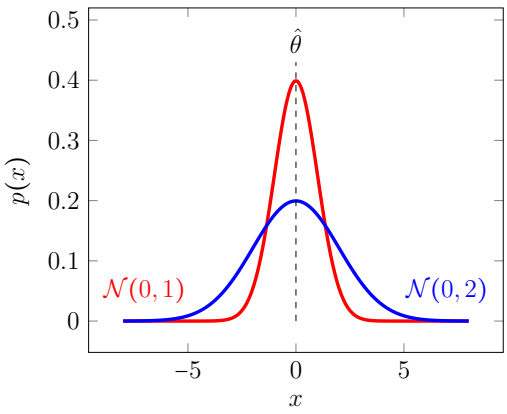

Figure 5.19 – *Probability density functions of two different Gaussian distributions with a point estimate of their mean labelled as* $\hat{\theta}$.

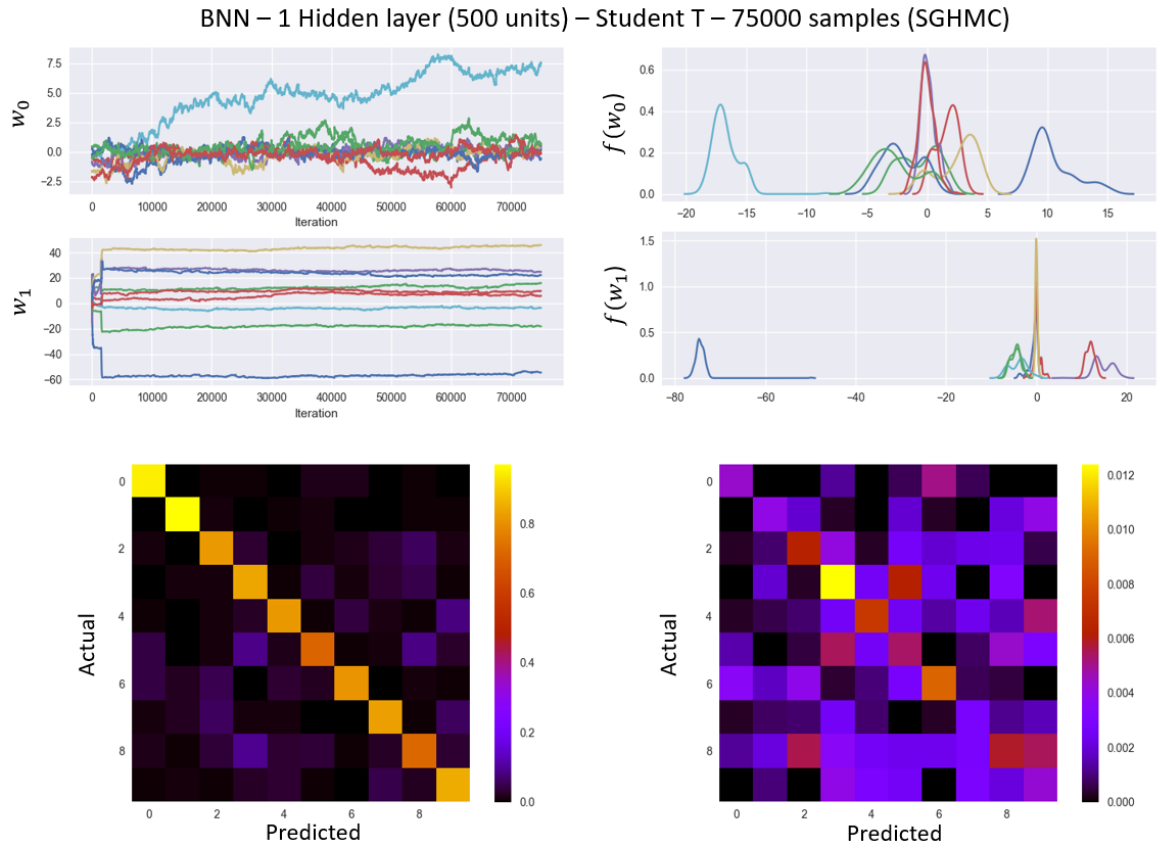

Figure 5.20 – *Diagnostic and predictive plots for the 1 hidden layer NN with 500 hidden units and* $T(\nu = 1, s = \frac{10^2}{d_j})$ *priors over all the parameters, using SGHMC to sample from the posterior. (Top left) The trace plots of samples of the weights from the two layers. The top right, and bottom two plots are the same to the ones seen before.*

predictions is much higher than that for the models with the Normal or Laplace distribution priors.

From the trace plot of the model with 800 hidden units and Laplace priors over the parameters (figure 5.21), we observe that whilst the weights in the second layer of the are sufficiently randomly scattered, suggesting that they may have converged; the weights in

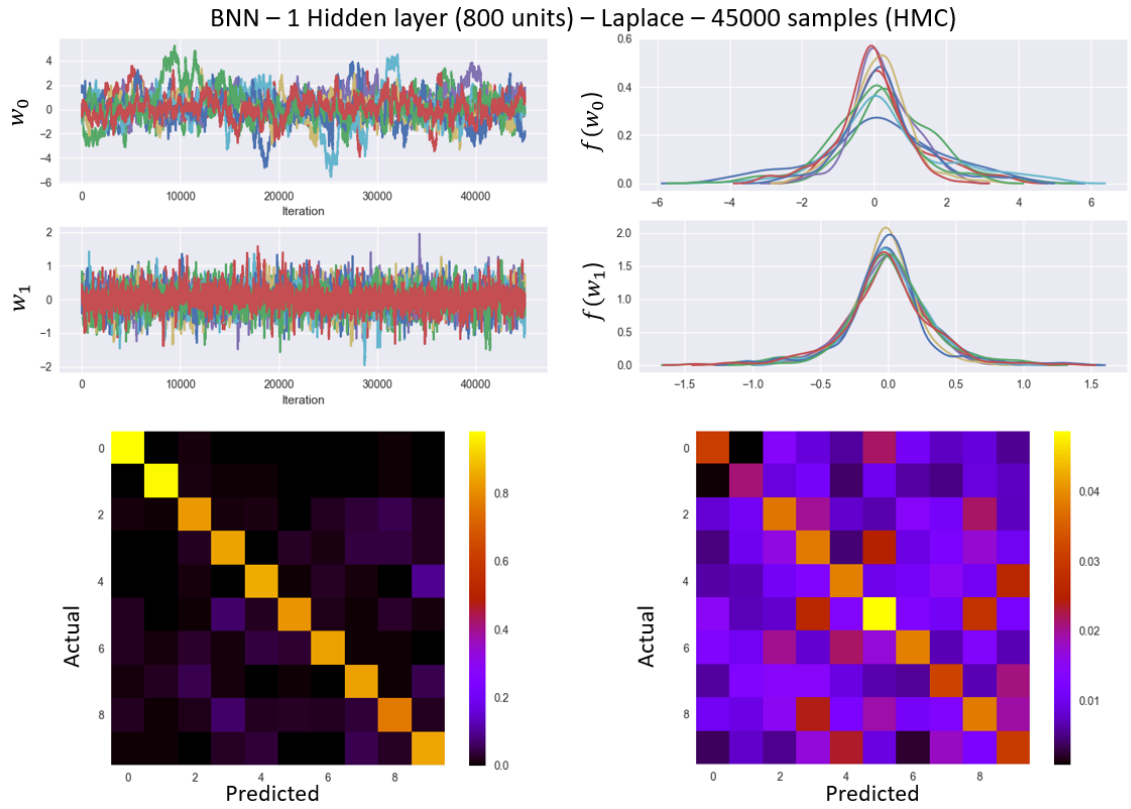

Figure 5.21 – *Diagnostic and predictive plots for the 1 hidden layer NN with 800 hidden units and* $\text{Laplace}(0, \frac{15^2}{d_j})$ *priors over all the parameters, using HMC to sample from the posterior. (Top left) The trace plots of samples of the weights from the two layers. The top right, and bottom two plots are the same to the ones seen before.*

the first layer ($\mathbf{W}_0$) seem like they still have not completely converged and perhaps taking more samples might solve this problem. Moreover, this might be indicative that the Laplace priors generate less correlated samples than the Normal priors seen in figure 5.22. We also note that the second layer of weights has marginal distributions of the weights that are quite similar in structure, whereas in the first layer the weights appear to have a different scale parameter to each other, but are again centred at zero. This could mean that, either the prior distribution has a strong impact on the posterior, or that the weights are in fact centred around zero. I believe that it is a combination of these two factors which gives rise to this structure, as we are only taking a very small sample of weights, most of which we expected to be centred at zero since the dataset is very sparse.

Throughout the MNIST 1 hidden layer models we can see that the variance of the predictive distribution of models with Laplace or Student T distribution priors is generally lower than that for the models with Normal distribution priors. This could suggest that our models using the sparsity promoting priors are more confident about their predictions.

The most surprising result in this section was that the models using the variational mean field approximations performed remarkably well. They also converge significantly faster than

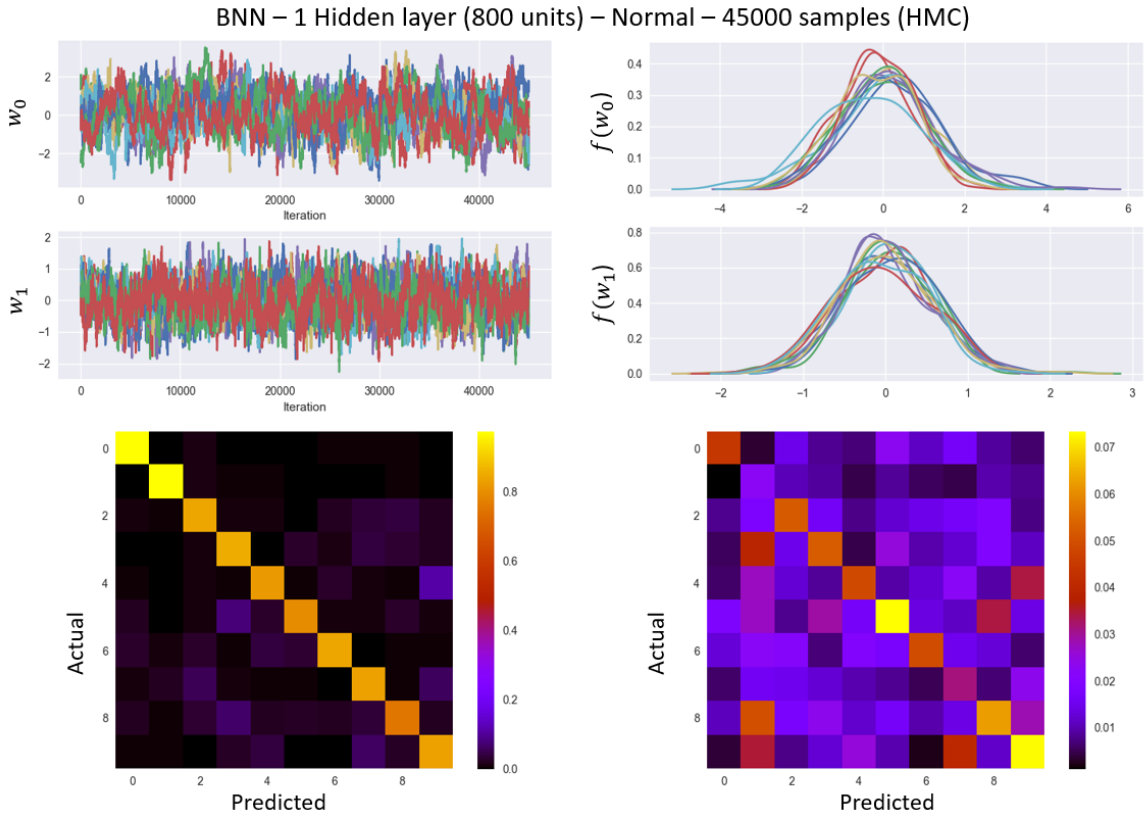

Figure 5.22 – *Diagnostic and predictive plots for the 1 hidden layer NN with 800 hidden units and* $\mathcal{N}(0, \frac{1^2}{d_j})$ *priors over all the parameters, using HMC to sample from the posterior. (Top left) The trace plots of samples of the weights from the two layers. The top right, and bottom two plots are the same to the ones seen before.*

the MCMC schemes, observing running times up to 70 times faster than for the HMC sampling scheme! However, this result was only restricted to the 1 hidden layer models as the variational mean field approximations did not work at all for the 2 hidden layer and CNN models. The factored approximation might be sensible for the smaller models, but for the more complex models it fails (see figure 5.24).

From figure 5.23 we can see that the predictive distribution using the variational approximation is quite similar in structure to the predictive distributions seen when using HMC, although there appears to be a slightly higher variance. Unlike before, we see that most of the second layer of weights exhibit a much more spiky structure and have lower variance than the first layer of weights both for the Normal and the Laplace priors. There are however some exceptions to this, where some of the weights have considerably higher variance than the other weights in the second layer.

To check whether the results obtained on the entire MNIST dataset (55000 training points, 10000 test points) were consistent with the small dataset we also run HMC on the large MNIST dataset. This is the only part of the dissertation where we apply the Bayesian methodology to the entire MNIST dataset due to the long time required to run these models.

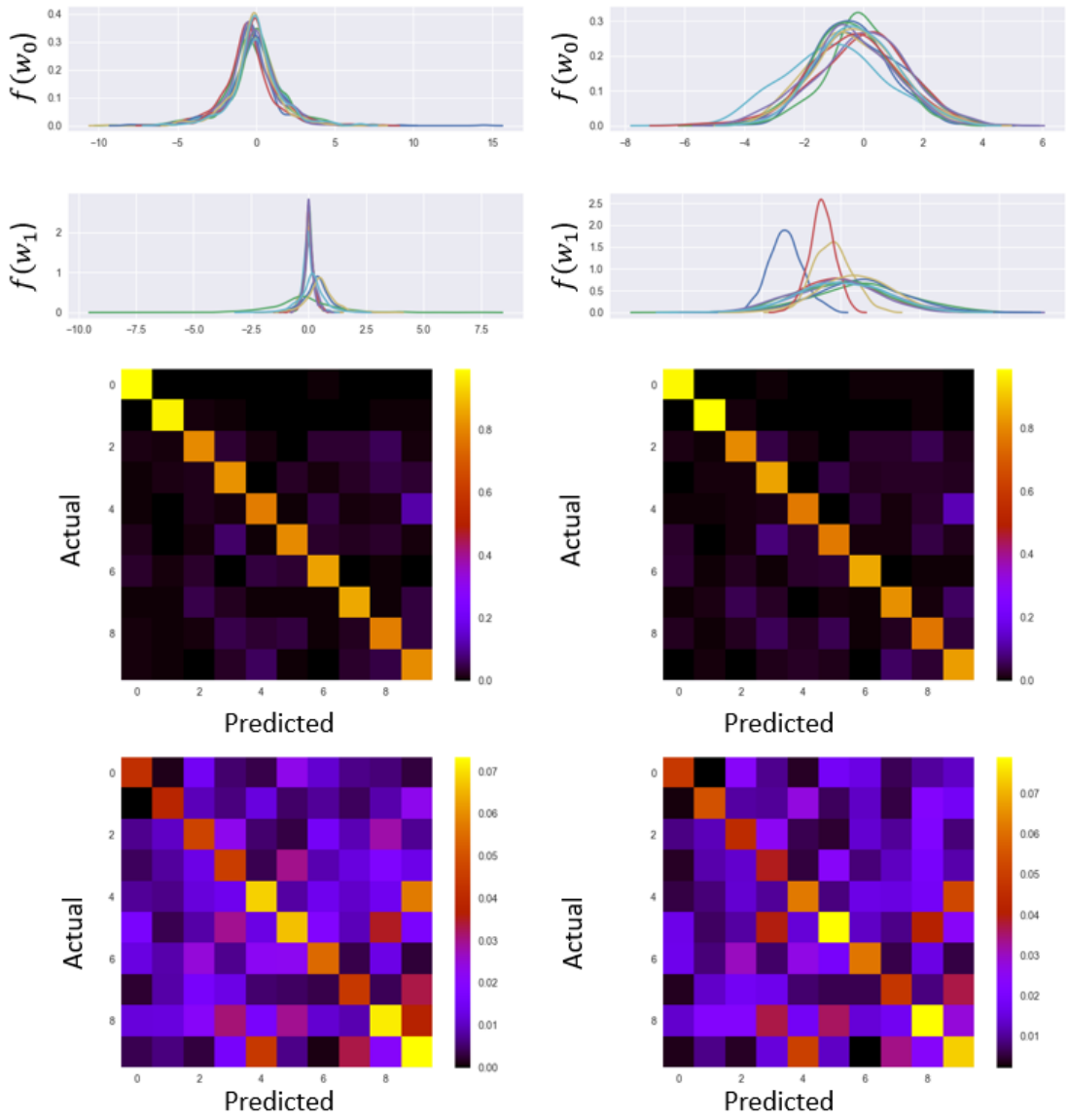

Figure 5.23 – *On the left are the distribution and predictive plots for the 1 hidden layer NN with 3000 hidden units and* $\mathrm{Laplace}(0, \frac{15^2}{d_j})$ *prior distributions over the weights, using the variational mean field approximation. On the right are the distribution and predictive plots for the 1 hidden layer NN with 3000 hidden units and* $\mathcal{N}(0, \frac{20^2}{d_j})$ *prior distributions over all the parameters.*

However, due to memory limitations we were required to feed the training data in batches of 10000, as the images could not be handled otherwise on the GPU memory. This is likely to have slightly worsened the performance of HMC as it requires gradient computations over the entire dataset. However, using batches means that we have acquired noisy estimates of the required gradients. Please note that batch HMC here is not the SGHMC described in (Chen et al., 2014 [57]), which uses the second-order Langevin dynamics (Welling and Teh, 2011 [46]).

We observe that for the large MNIST dataset the HMC sampling scheme for both the Laplace and Normal distribution priors produce predictive distributions that are very similar to each other, even though the marginal distributions of the samples are quite different (see

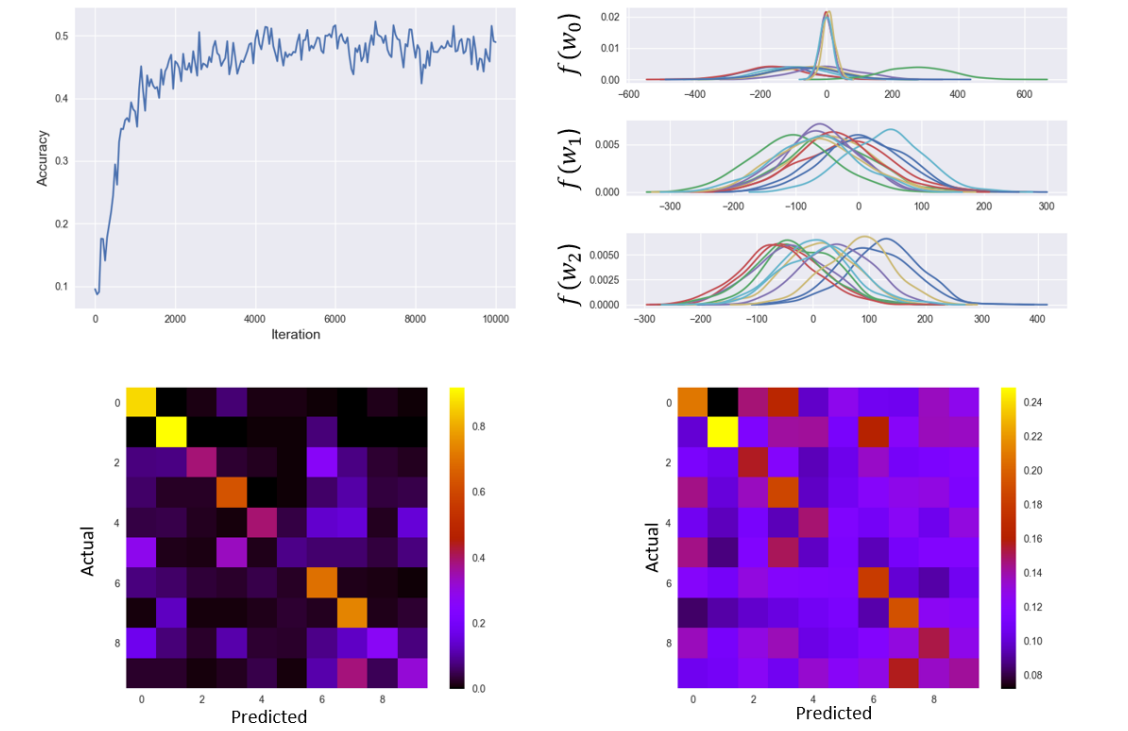

Figure 5.24 – *Diagnostic plots of a 2 hidden layer BNN with 100 hidden units in each layer, using VMF approximation on the small MNIST dataset. All the parameters had* $\mathcal{N}(0, \frac{7^2}{d_j})$ *priors. (Top left) the learning curve on the test set during training. The other plots have been previously defined. The final model accuracy* $\pm$ *1 standard deviation using the MC estimate on 500 was 0.535* $\pm$ *0.036.*

Hidden units	Prior Distribution	No. of samples	ESS	MC accuracy $\pm 1\sigma_{\text{mc}}$	Point est. accuracy	Time (mins)
800	$\text{Laplace}(0, \frac{13^2}{d_j})$	50400	162	0.967 $\pm$ 0.005	0.415	368.9
800	$\mathcal{N}(0, \frac{20^2}{d_j})$	50400	172	0.968 $\pm$ 0.003	0.725	368.2

Figure 5.25 – *Summary table for the 1 hidden layer BNN using (batch) HMC on the large MNIST dataset.*

figures 5.26 and 5.27) . For the second layer of weights, the model with Normal priors seem to be mostly centred around 0 with the exception of some weights that have means slightly different from 0, whereas for the Laplace priors, all the sampled weights are centred at zero. Again the first layer of weights does not seem to converge.

The predictive distributions for both models seem very sensible, as the predictive distribution struggles to distinguish the digit 4 from 9, and the digit 3 from 5, which is logical given the way the images are being shrank. The fact that the point estimate accuracy for the model with the Normal priors (0.725) is higher than that for the model with the Laplace priors (0.415) is explained by the Laplace distribution containing more information in its tails than the Normal distribution (fatter tails). This implies that the point estimate would lie further from the truth in the case of the Laplace priors.

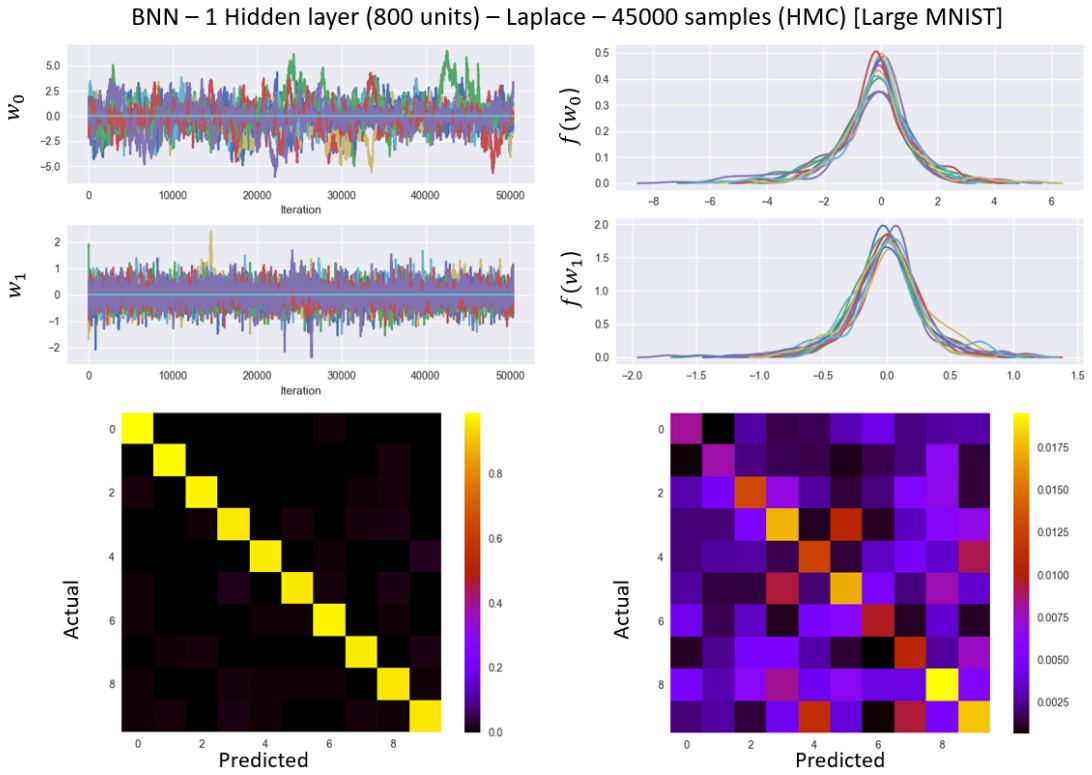

Figure 5.26 – *Diagnostic and predictive plots for the 1 hidden layer NN with 800 hidden units and* $Laplace(0, \frac{13^2}{d_j})$ *priors over all the parameters applied to the large MNIST dataset.*

Looking at the running times of the two models, we demonstrate that using HMC scales very well with the sample size, and in fact the larger sample size benefits more from running on the GPU. The ability to scale well to larger datasets is crucial for the practicality of the algorithm. Although, the running times for HMC are incomparable to those for feed-forward NNs. VI shows much promise on the less complex models, by achieving great performance at running times that are comparable to feed-forward NNs.

Regression (1 hidden layer)

Using the same regression dataset as previously, BNNs with 1 hidden layer will be explored. The samples are all collected using HMC only, as VI and SGHMC did not perform as well. In [56] we also provide an example script applying a more state-of-the-art variational inference algorithm known as Automatic Differentation Variational Inference (ADVI) (Kucukelbir et al., 2015 [66]), however we do not evaluate this against other approximate inference algorithms, in the paper ADVI is compared to MCMC algorithms.

Noting back to the Gaussian likelihood § 2.1, all the models used for the regression dataset assume a Gaussian likelihood with mean being the output of the NN and variance $\sigma^2 = 0.07^2$, for simplicity this quantity was fixed at a value that seemed to perform best. A

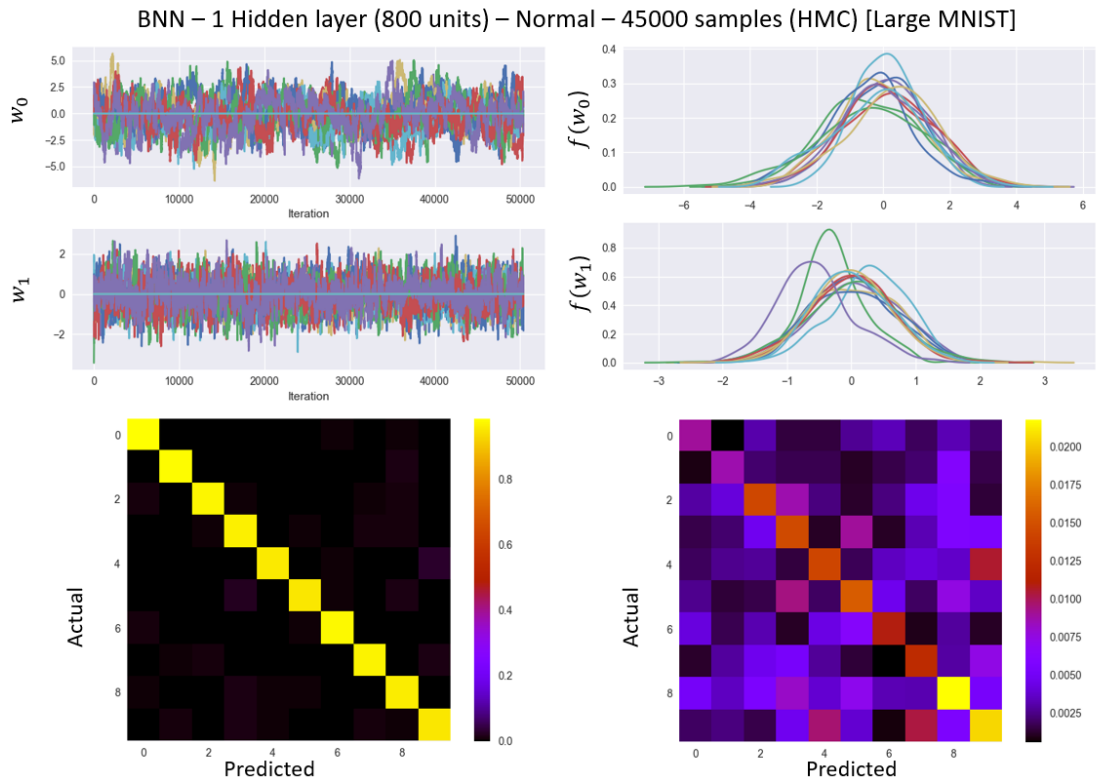

Figure 5.27 – *Diagnostic and predictive plots for the 1 hidden layer NN with 800 hidden units and* $\mathcal{N}(0, \frac{20^2}{d_j})$ *priors over all the parameters applied to the large MNIST dataset. prior distributions over all the parameters.*

Hidden units	Prior Distribution	No. of samples	ESS	MC MSE $\pm 1\sigma_{\text{mc}}$	Point est. MSE	Time (mins)
1000	$\text{Laplace}(0, \frac{20^2}{d_j})$	150000	192	0.208 ± 0.001	0.336	122.5
1000	$\mathcal{N}(0, \frac{15^2}{d_j})$	150000	177	0.207 ± 0.004	0.254	121.7
2000	$\text{Laplace}(0, \frac{20^2}{d_j})$	150000	156	0.208 ± 0.002	13.169	146.4
2000	$\mathcal{N}(0, \frac{18^2}{d_j})$	150000	192	0.208 ± 0.005	0.678	146.9
3000	$\text{Laplace}(0, \frac{30^2}{d_j})$	150000	258	0.209 ± 0.016	0.868	178
5000	$\text{Laplace}(0, \frac{20^2}{d_j})$	150000	184	0.208 ± 0.002	0.456	317.8

Figure 5.28 – *Summary table for BNNs with 1 hidden layer on the* CO_2 *regression dataset, using HMC to generate the samples from the posterior.*

more sophisticated model would essentially try to be Bayesian about this term and allow the sampling scheme to find the right value for the output noise.

Choo (2000, [67]) explored the use of other hyper-parameter sampling using Gibbs sampling, but also hyper-parameter sampling using HMC, his results however were unsuccessful. The inherent inability of HMC to move between multiple modal regions was his explanation for this phenomenon, which made movement from the region with high hyper-parameter values to the region with low hyper-parameter values exceedingly difficult. As a conse-

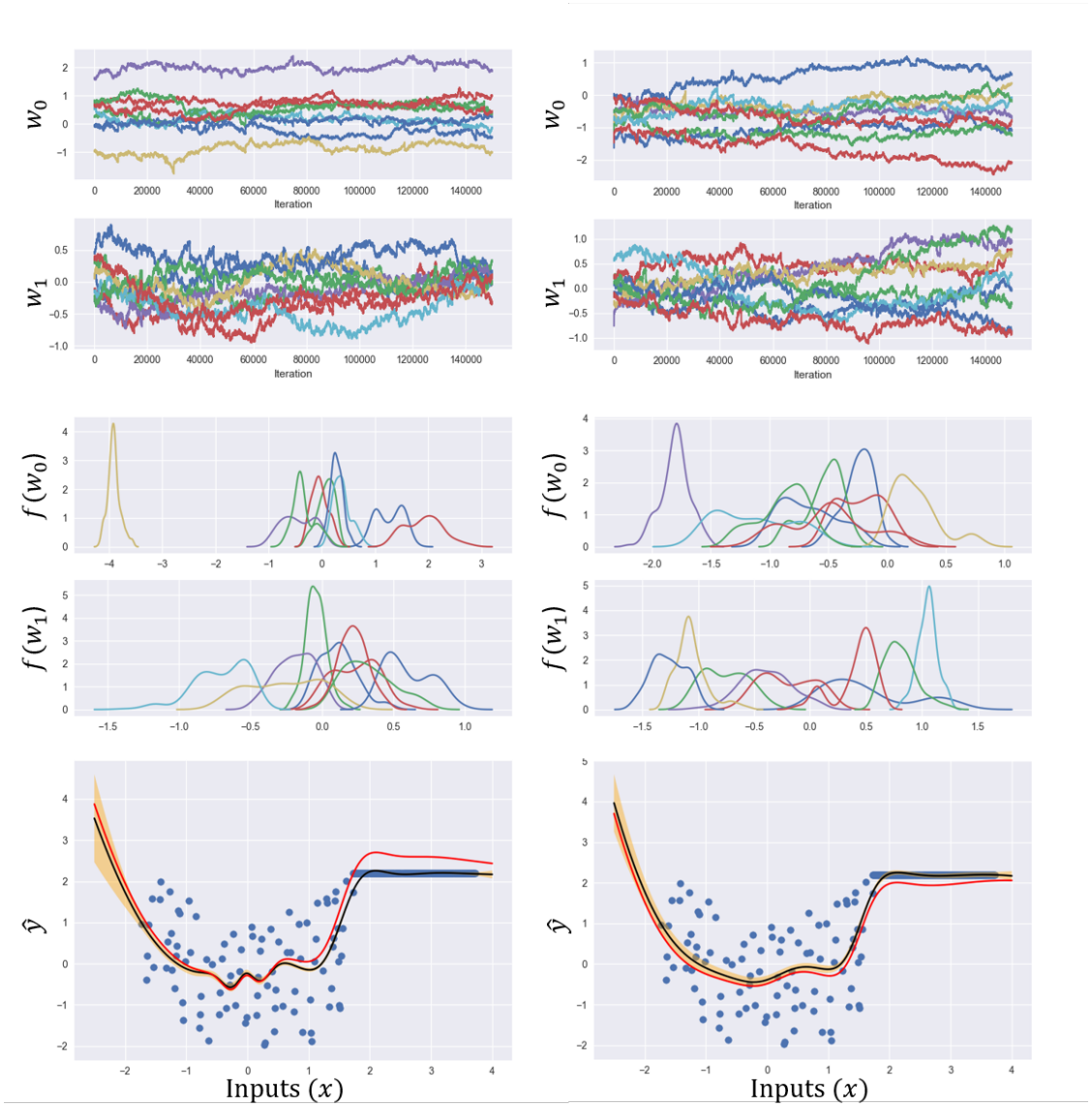

Figure 5.29 – *Diagnostic and predictive plots for BNNs with 1 hidden layer of 1000 units. (Left)* $Laplace(0, \frac{20^2}{d_j})$ *on the (Right)* $\mathcal{N}(0, \frac{15^2}{d_j})$ *prior. The bottom plots show the MC approximation of the predictive distribution (black), the point estimate (red), the test point scatter, and the orange region is the 99% credibility interval.*

quence of these difficulties we will not explore fully Bayesian approaches in this dissertation.

This dataset posed difficulties with finding a step size that was high enough to move at all in the space where HMC searched, but low enough for the algorithm to accept some of the proposals. This meant that we were forced to use a step size of 5×10^{-5}, and to compensate for the extremely small step size we used 60 leap frog steps. This meant that the proposals were very highly correlated. Looking at figures 5.28 and 5.29 we can see how few samples were actually used (typically 1 sample per 1000) meaning that this sampling scheme was very inefficient.

For the models with 1000 hidden units with a Laplace prior distribution, we can see that

the weights of the first layer possibly burnt in by iteration 60000, whereas the second layer weights $\mathbf{W}_1$ do not show any indication of convergence. It is possible that the chain was not ran for long enough, but in order to draw this conclusion we would typically require a trace of the output probabilities instead of the weights.

The marginal distributions for the weights seem to show indications of bi-modality in some cases, which makes it exceedingly difficult for HMC to find a good solution. Nonetheless, the models should not be deemed as a failure, as their performance is still relatively good, thus capturing the general trends in the data. The tight credibility regions are believed to be an artefact of the correlation of the samples and the low output variance. The Laplace priors seem to be better at detecting the signals from the data, and we see that the predictive distribution has a wavelike curvature in the centre, which is something that we could typically expect to see for this particular dataset. Once again, we observe that the point estimate for the model with Laplace priors deviates the most from the predictive distribution, this deviance is most noticeable at the extremities of the data.

This signal detection pattern is repeated for the networks with 2000 hidden units (figure 5.30), where the Laplace priors offer a more wavelike trend for the predictive distribution. The deviance of the point estimates is a lot more severe in this case, especially for the model with the Normal distribution priors. From the marginal distribution plots can again observe the possibility of bi-modality, which ultimately limits the performance of the HMC algorithm.

Contrastingly to the model with the Normal priors, the marginal distributions tend to be mostly clamped together in one region with some marginal distributions lying significantly further from the group (see figures 5.29 and 5.30). This is believed to emphasise the sparsity effect that the Laplace prior is having. The marginal distributions for the model with the Normal priors tend to be more spread out over the space. This difference in the structure of the marginals is believed to give rise to the wavelike (signal detection) predictive distribution, whereas the Normal priors tend to smooth the predictive distribution.

5.3.2 BNN with 2 hidden layers

We have seen that BNNs work well for single hidden layer networks, but we will now focus on larger models with 2 hidden layers. This is an important milestone that useful BNNs need to overcome in order to ensure that they can be useful in practice.

We will explore models of similar scope to the ones fitted using back propagation. However, due to the algorithm time complexity we are limited to a smaller number of models. We will not see many models using variational approximations and/or with Student T distribution priors. These were initially tested, and were found to fail in practice. The reader is encour-

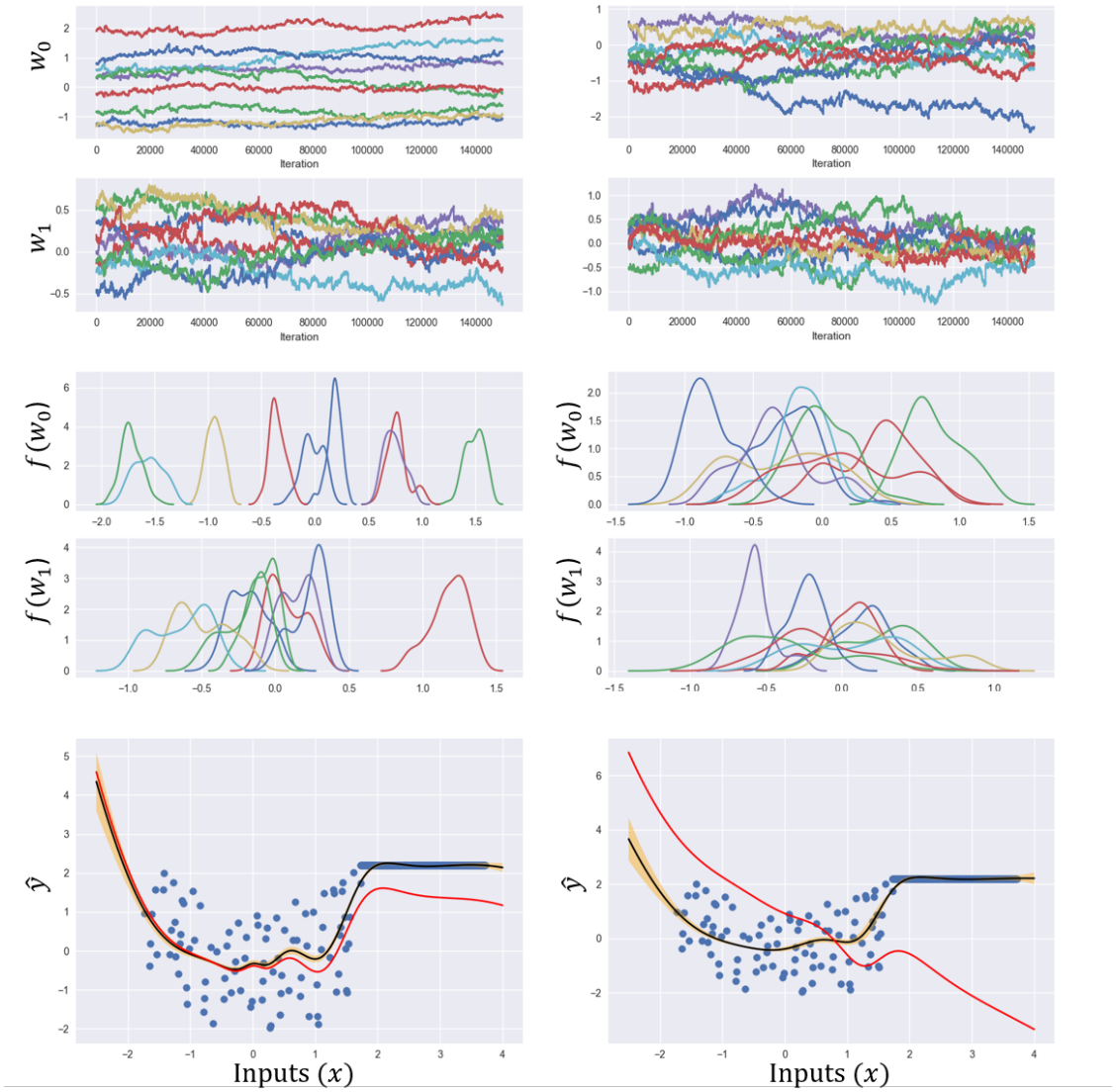

Figure 5.30 – *Diagnostic and predictive plots for BNNs with 1 hidden layer of 2000 units. (Right)* $\mathcal{N}(0, \frac{18^2}{d_j})$ *prior on the (Left)* $\mathrm{Laplace}(0, \frac{20^2}{d_j})$*. The bottom plots show the MC approximation of the predictive distribution (black), the point estimate (red), the test point scatter, and the orange region is the 99% credibility interval.*

aged to explore such models, using the code provided in [56], as it is possible that the "right" hyper-parameters for the HMC algorithm were overlooked.

Classification (2 hidden layers)

All models explored in this section will use the small MNIST dataset that was used for the previous models. The graphical output presented will refer only to the key findings. The two hidden layer MNIST models are in general more accurate than the single hidden layer BNNs at the cost of a higher variance.

In the models explored, the marginal distributions for the weights of the models with

Hidden units	Prior Distribution	No. of samples	ESS	MC accuracy $\pm 1\sigma_{\text{mc}}$	Point est. accuracy	Time (mins)
250	$\text{Laplace}(0, \frac{10^2}{d_j})$	45000	156	0.856 ± 0.010	0.782	62.4
250	$\mathcal{N}(0, \frac{15^2}{d_j})$	45000	154	0.868 ± 0.022	0.622	64.5
300	$\text{Laplace}(0, \frac{10^2}{d_j})$	36000	173	0.871 ± 0.010	0.791	55.1
300	$\mathcal{N}(0, \frac{15^2}{d_j})$	36000	153	0.863 ± 0.027	0.717	55
500	$\text{Laplace}(0, \frac{15^2}{d_j})$	36000	196	0.875 ± 0.015	0.549	72.4
500	$\mathcal{N}(0, \frac{20^2}{d_j})$	36000	195	0.865 ± 0.014	0.537	72.5

Figure 5.31 – *Summary table for BNNs with 2 hidden layers on the small MNIST dataset, using HMC to generate the samples.*

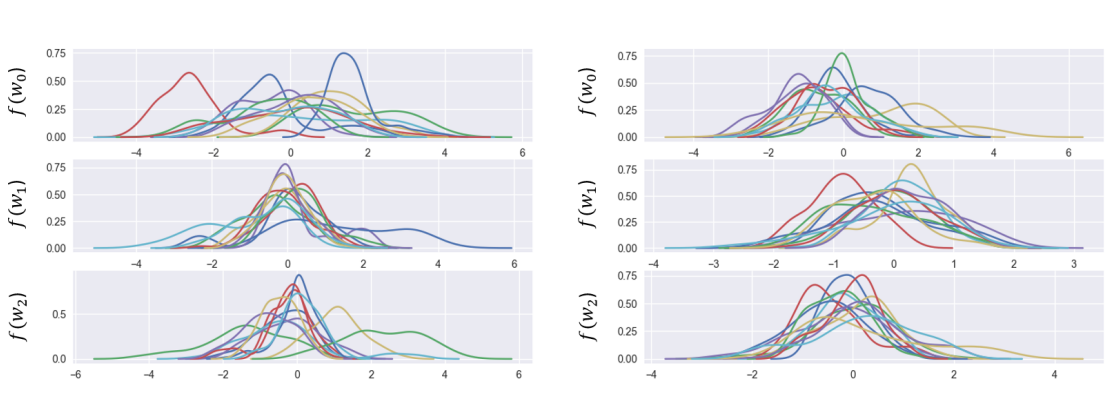

Figure 5.32 – *The marginal distribution plots for the 2 hidden layer BNN with 300 units in each layer for the small MNIST dataset. (Left) The model with* $\text{Laplace}(0, \frac{10^2}{d_j})$ *priors over the parameters. (Right) The model with* $\mathcal{N}(0, \frac{15^2}{d_j})$ *priors over the parameters.*

Laplace priors tend to be more clumped together. The active weights are consequently further from the rest of the weights. With the exception of the 2 hidden layer (300 hidden unit) model (figure 5.32), the marginals for most of the less active weights are clumped together quite closely around 0. One could expect this from a Laplace prior which has fatter tails and thus an observation far from zero is more probable than if we had a Normal prior. This ensures that when the Laplace posterior mean is adjusted it is because of a genuine (strong) signal from the data, the Laplace prior might mean that certain weaker signals do not result in an adjustment in the posterior. On the other hand, the models with Normal priors are more likely to respond to a weaker signal from the data and thus tend to be less clumped.

The predictive plots for both the 500 hidden unit models (figures 5.33 and 5.34) seem sensible, with the models making the most mistakes on the digit 8, and least mistakes on digits 0 and 1. As previously, the digit 4 is mistaken with the digit 9 and vice versa. And the digit 3 is mistaken with the digit 5 the most and vice versa. Intuitively, these mistakes should make sense to us. However, as the networks become wider the models become increasingly likely to be unidentifiable (see figures 5.35 and 5.36). Paradoxically, the deeper

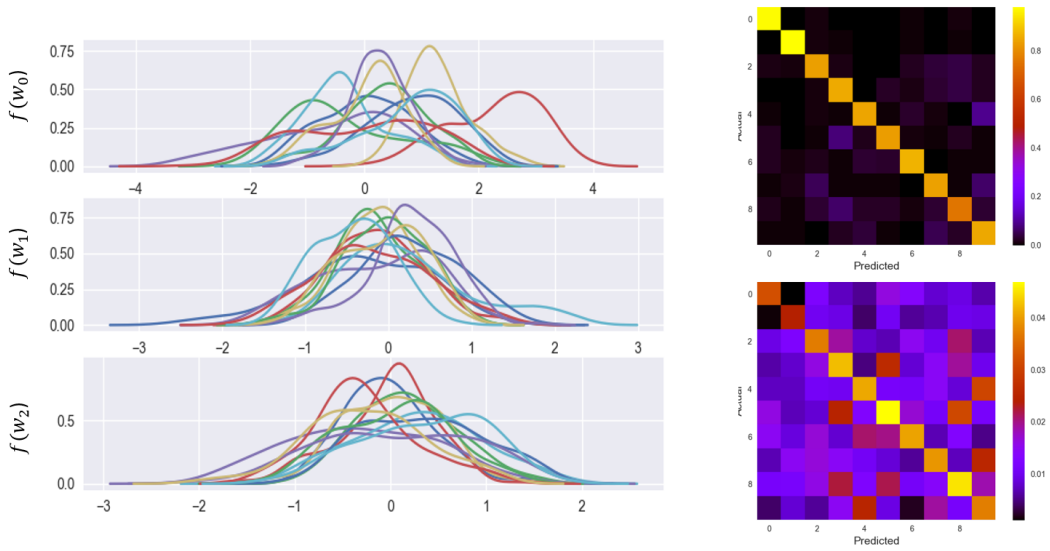

Figure 5.33 – *The marginal distributions and predictive plots for the 2 hidden layer BNN with* $\mathcal{N}(0, \frac{20^2}{d_j})$ *and 500 units in each hidden layer.*

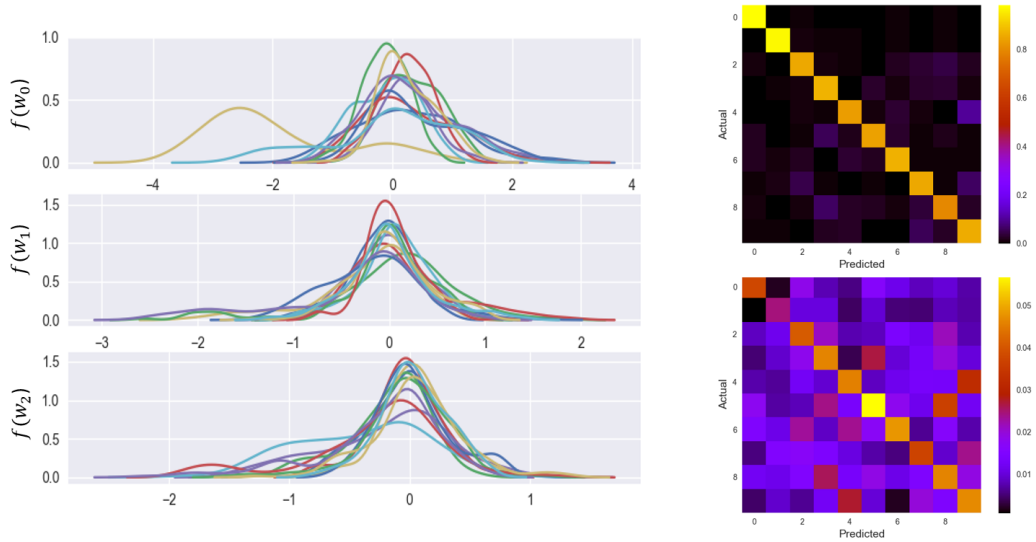

Figure 5.34 – *The marginal distributions and predictive plots for the 2 hidden layer BNN with* $Laplace(0, \frac{15^2}{d_j})$ *and 500 units in each hidden layer.*

and less wide models have more stable output probabilities than the models with many hidden units and only one hidden layer (figure 5.16).

Regression (2 hidden layers)

Adding a second hidden layer to the BNNs introduced another layer of complications in setting the right step sizes. The step size in these models had to be reduced further to 5.5×10^{-6}, which was unbearably small. Our exploration of the state space was relatively limited due to this change. Even increasing the number of leapfrog steps to 70 still did not help substantially. Sadly there was no other alternative which would avoid the rejection of all the

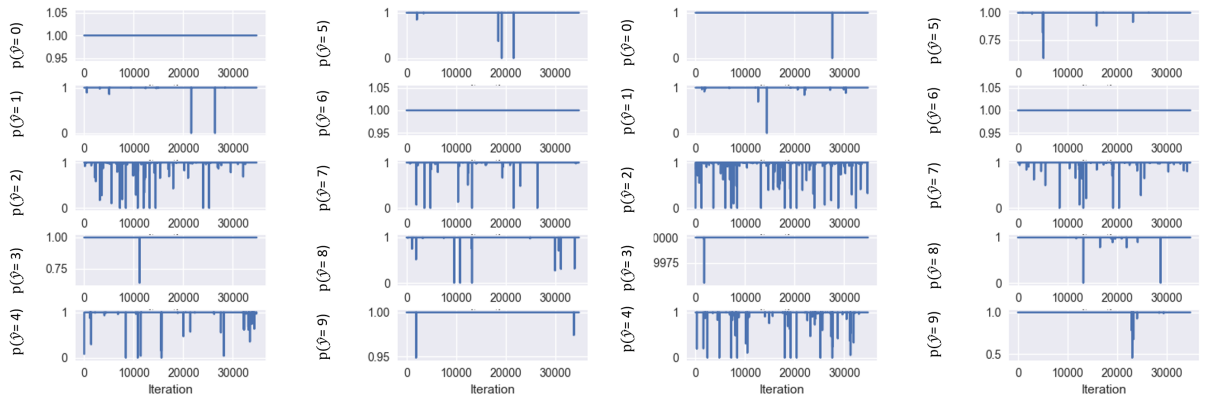

Figure 5.35 – *The trace of the output probabilities for the same datapoint used in previous example. Using the BNNs with 2 hidden layers of 300 units each. On the (left) the model with Laplace priors is used, on the (right) the model uses Normal priors.*

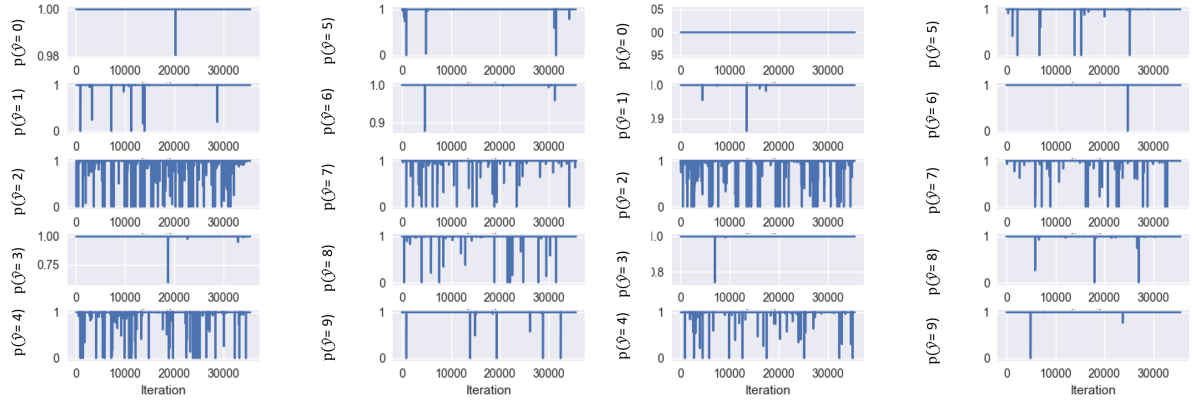

Figure 5.36 – *The trace of the output probabilities for the same datapoint used in previous example. Using the BNNs with 2 hidden layers of 500 units each. On the (left) the model with Laplace priors is used, on the (right) the model uses Normal priors.*

proposals. Naturally decreasing the step size means that the samples are more correlated (see figure 5.39).

The trace plots for the first layer of weights seen in figures 5.38 and 5.39 are almost completely flat meaning that the posterior distribution is almost identical to the prior distribution. Thus the prior distribution could be having a strong influence on this layer of weights. However, as we are naturally limited to the numbers of weights that can be observed, we are not in a position to conclude that this is the case for all the other weights in the first layer.

It is important to stress that there was a substantial indication that for some of the larger models, such as the ones with the 500 hidden units, we did not collect enough samples to have a good approximation of the posterior distribution. This becomes evident by looking at figure 5.40. Plotting the test MSE during the sampling stage exposes the inability for the approximations of the weights to converge quickly enough, making the use of HMC less

practical in this situation.

Hidden units	Prior Distribution	No. of samples	ESS	MC MSE $\pm 1\sigma_{\text{mc}}$	Point est. MSE	Time (mins)
50	$\mathrm{T}(\nu = 3, s = \frac{5^2}{d_j})$	120000	197	0.209 ± 0.010	0.227	101
100	$\mathrm{T}(\nu = 3, s = \frac{5^2}{d_j})$	120000	192	0.211 ± 0.0005	0.211	130.1
100	$\mathrm{Laplace}(0, \frac{5^2}{d_j})$	120000	163	0.205 ± 0.001	0.207	128.6
300	$\mathrm{Laplace}(0, \frac{5^2}{d_j})$	80000	94	0.204 ± 0.001	0.219	83.8
300	$\mathcal{N}(0, \frac{7^2}{d_j})$	120000	147	0.206 ± 0.001	0.27	158.4
500	$\mathrm{Laplace}(0, \frac{7^2}{d_j})$	135000	201	0.202 ± 0.001	0.631	211.6
500	$\mathcal{N}(0, \frac{10^2}{d_j})$	140000	217	0.206 ± 0.003	0.211	211.3

Figure 5.37 – *Summary table for BNNs with 2 hidden layers, using the CO_2 regression dataset.*

The continuously declining MSE during the training is both a good and a bad sign; a good sign in the sense that the samples we are accepting seem to give more accurate predictions, and bad in the sense that the sampling scheme has not converged and technically we should not be using the samples before convergence in building the predictive distribution. However, due to a lack of computational resources to ran these models for many days in order to generate enough completely uncorrelated and converged samples from the posterior, we settle for a sub-optimal scheme.

Sub-optimality was mainly prevalent for the 2 hidden layer models with (500 or more hidden units), firstly because we were forced to use extremely small step sizes, and secondly because the space that needed to be explored was significantly larger. Comparing the number of parameters that need to be estimated as in a BNN with 1 hidden layer with 5000 hidden units (a bit over 10000 parameters) in comparison with a BNN with 2 hidden layers of 500 units each (over 250000 parameters) we see that the difference in the state space is immense.

In an attempt to explore whether we could use the Student T distribution we explored 2 hidden layer models with scaled Student T distributions of 3 degrees of freedom. This achieved an increase in the stability of such distributions, although the models performed better with fewer hidden units. Although the Student T distribution priors performed worse than either the Laplace or Normal distribution priors, there seemed to be a strong indication that the Student T distributions have a lot of potential for picking up signals in the data (see figure 5.41). The marginal distributions are very peaky for the 100 hidden unit model, which could mean that the model has a high confidence in the estimates for the weights, or that the posterior samples are highly correlated.

When comparing the models with Student T distribution priors to the model of 100 hid-

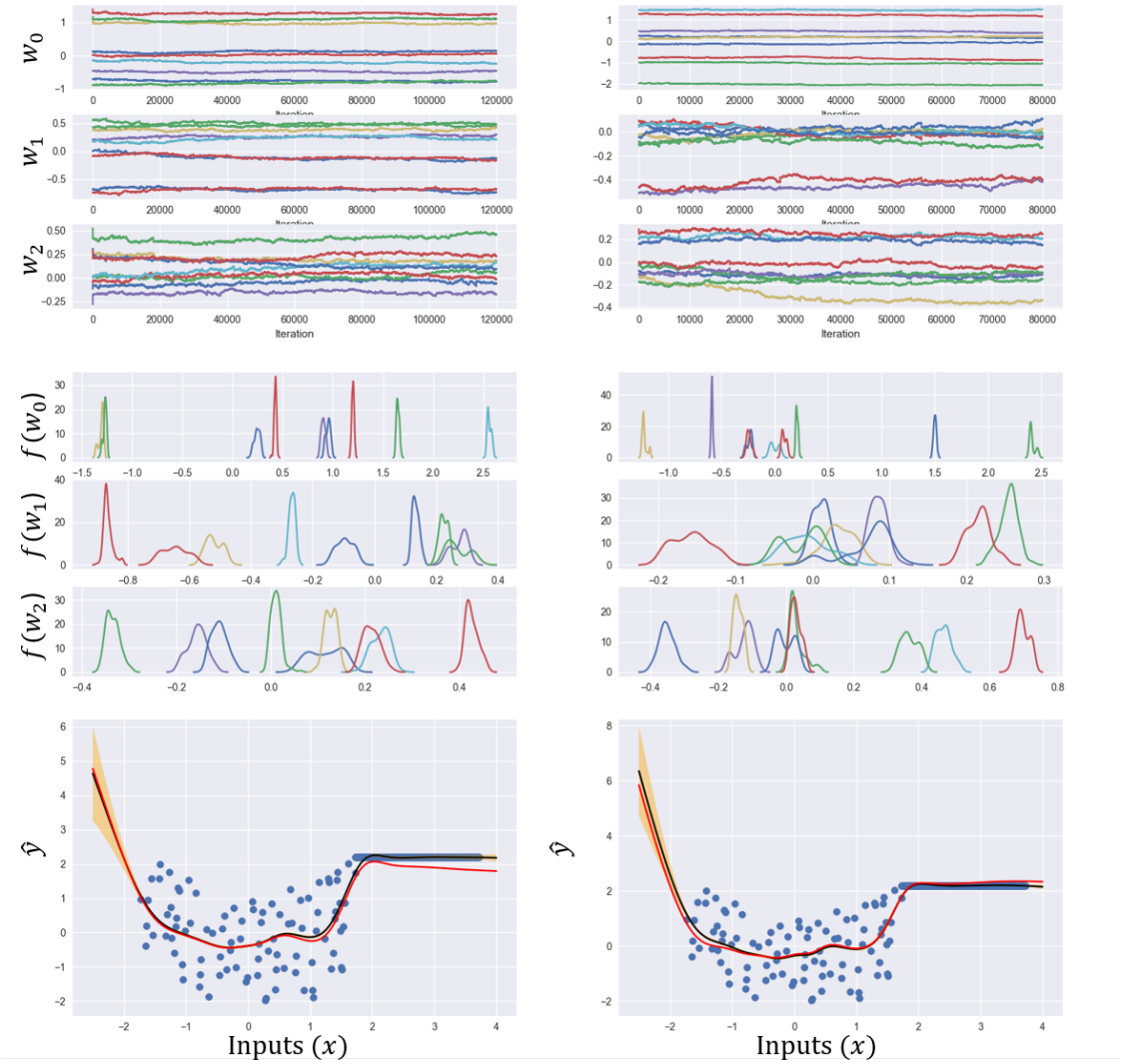

Figure 5.38 – *Diagnostic and predictive plots for BNNs with 2 hidden layers of 300 units. (Left)* $\mathcal{N}(0, \frac{7^2}{d_j})$ *priors are used for all the parameters, on the (Right)* $\mathrm{Laplace}(0, \frac{5^2}{d_j})$ *priors are used for all the parameters.*

den units and a Laplace prior distribution, it becomes apparent that they are significantly outperformed by the Laplace prior distribution model. This provides a smoother and more consistent fit to the data (see figure 5.42). We believe that this provides a strong indication that there is considerable room for calibration of the Student T distribution models, such as exploring a higher number of degrees of freedom and other scale parameters.

5.3.3 BCNN with 2 convolutional layers and a hidden layer

From figures 5.44, 5.45, and 5.46 there is strong evidence that the BCNN models cannot be identified using the HMC sampling scheme. For the specific probability traces assessed digits 2, 4 and 7 seem to have difficulties converging. The probabilities seem to also take extreme values, either 0 or 1 in most cases, this observation and the marginal distributions

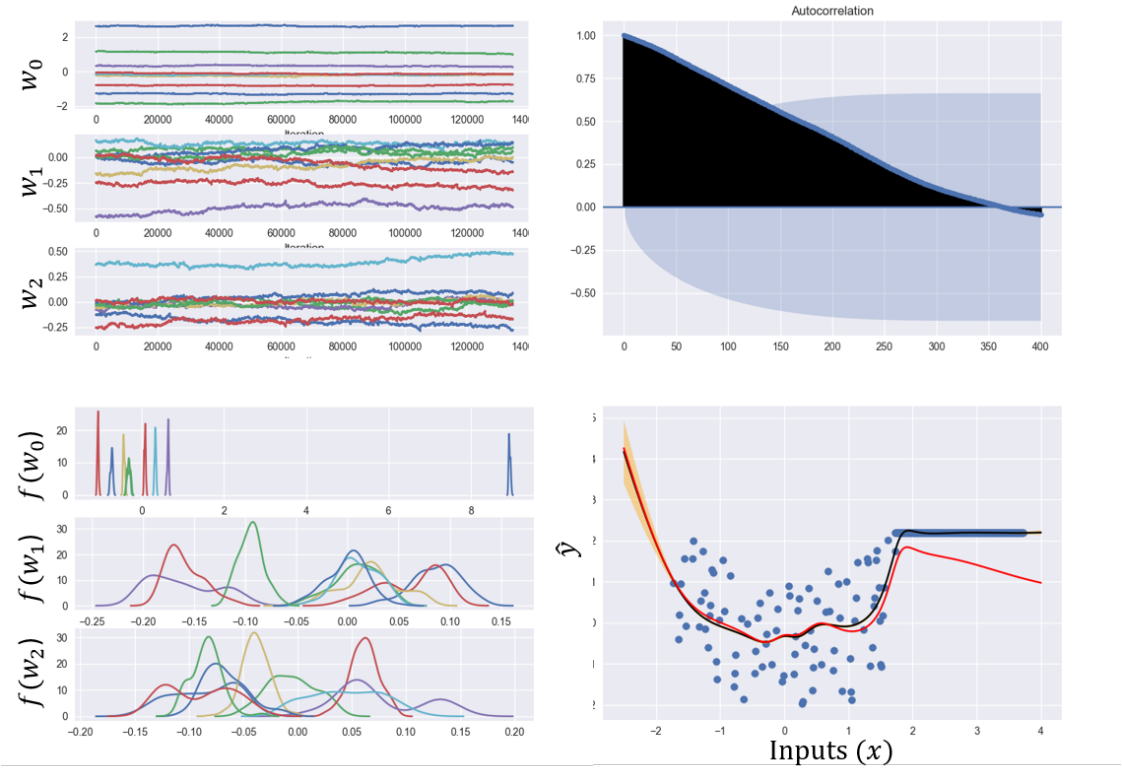

Figure 5.39 – *Diagnostic and predictive plots for the BNN with 2 hidden layers of 500 units and* $Laplace(0, \frac{7^2}{d_j})$ *priors over the weights. The (top right) plot is a plot of the auto-correlations of a sample of weights, the x-axis is the lag and y-axis the autocorrelation value.*

could suggest the possibility of multi-modal marginal distributions. This means that HMC is at its limitations, as it cannot handle multi-modality very well, instead the samples will be concentrated around one of the modes. To handle multi-modality one would perhaps require to incorporate this into the model structure, defining mixture of Gaussian or mixture of Laplace priors. However this would add another layer of complexity to the model which ultimately also needs to be approximated. For this approximation, a possible suggestion is to use a less expensive method of approximation such as VI.

What could be most interesting, is to see a combination of HMC with VI. For example using HMC to sample from the posterior distribution, followed by VI to estimate the hyper-parameters. This inexpensive addition to the iterative scheme might allow the use of more sophisticated models, such as the use of a mixture model prior that could ultimately enable the HMC algorithm to converge even in the case of multi-modality.

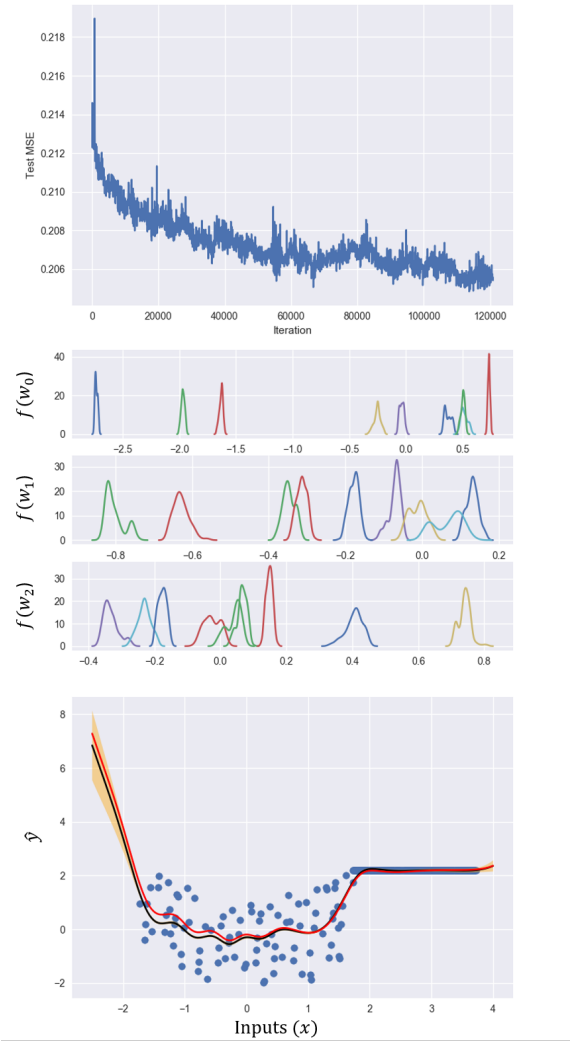

Figure 5.40 – *Diagnostic and predictive plots for the 2 hidden layer BNN with* $\mathcal{N}(0, \frac{10^2}{d_j})$ *priors over the parameters and 500 hidden units in each hidden layer. (Top) Plot of the test mean squared error during the sampling stage of the algorithm, evaluated using MC integration on random samples from the posterior which does not incorporate a burnin or spacing out the samples. Note that the first 20000 iterations are removed from this plot to allow clear visibility. The latter two plots have been explained previously.*

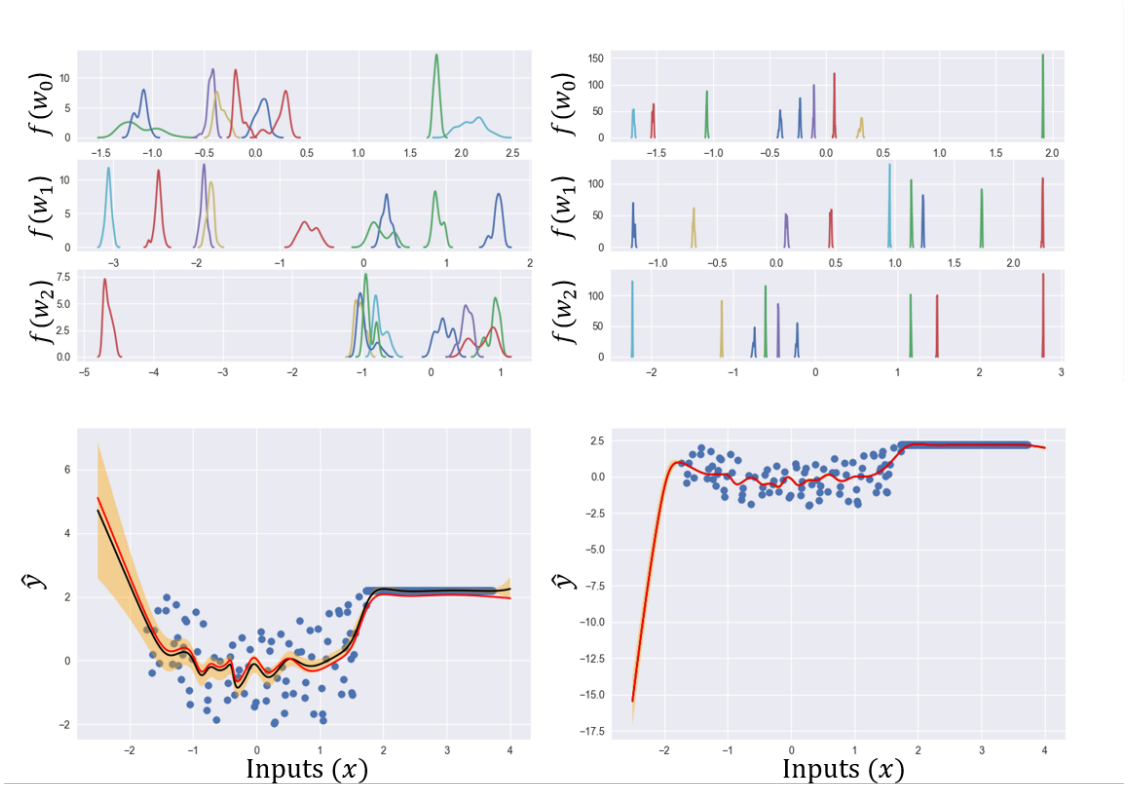

Figure 5.41 – *Marginal distribution and predictive plots for 2 hidden layer BNNs with* $\mathrm{T}(\nu = 3, s = \frac{5^2}{d_j})$ *priors over all the parameters. (Left) model with 50 hidden units in each hidden layer. (Right) model with 100 hidden units in each hidden layer.*

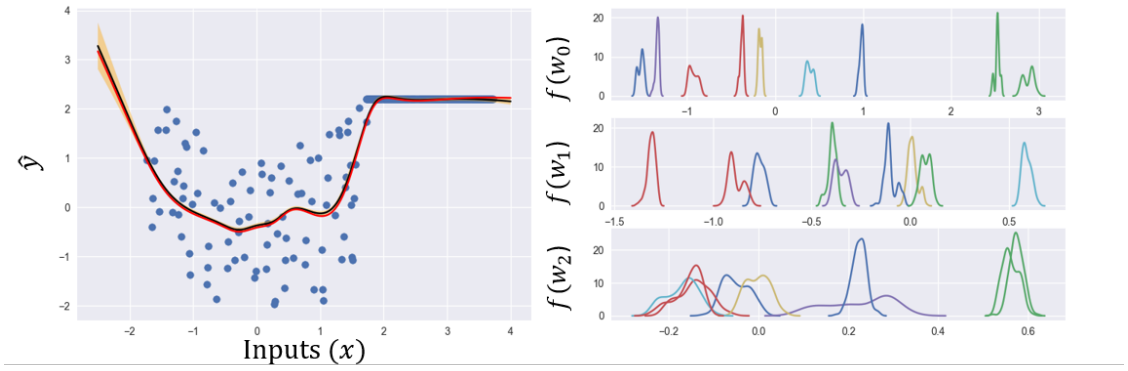

Figure 5.42 – *Marginal distribution and predictive plots for 2 hidden layer BNNs with* $\mathrm{Laplace}(0, \frac{5^2}{d_j})$ *priors over all the parameters and 100 hidden units in each hidden layer.*

Hidden units	Prior Distribution	No. of samples	ESS	MC accuracy $\pm 1\sigma_{\text{mc}}$	Point est. accuracy	Time (mins)
250	$\text{Laplace}(0, \frac{10^2}{d_j})$	37500	77	0.927 ± 0.016	0.897	314.1*
250	$\mathcal{N}(0, \frac{15^2}{d_j})$	37500	73	0.939 ± 0.013	0.908	314*

Figure 5.43 – *Summary table for BCNN consisting of 2 (convolutional & maxpool) layers, a fully connected layer and a hidden layer with 250 units (see figure 1.3 in § 1.2). The samples were generated using the HMC sampling scheme on the small MNIST dataset with unreduced images (28x28). * Both the models were ran on an NVIDIA K80 GPU, which was roughly 1.92 times slower than the NVIDIA GeForce® GTX 1070 used for all other experiments. The figures have all been adjusted downwards by a factor of 1.92.*

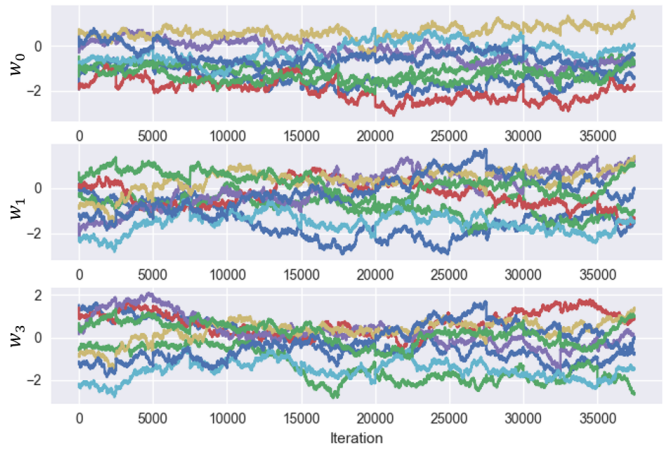

Figure 5.44 – *Trace plot for a sample of weights of the BCNN with $\mathcal{N}(0, \frac{15^2}{d_j})$ prior distributions. Weights are sampled from the first ($\mathbf{W}_0$), second ($\mathbf{W}_1$) and final layers ($\mathbf{W}_3$).*

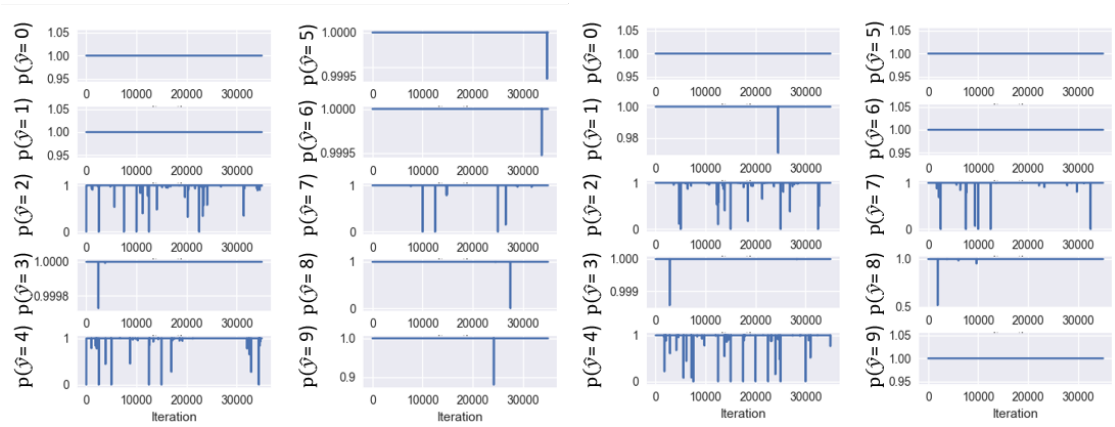

Figure 5.45 – *The traces of the output probabilities of each of the 10 digits in the first instance of the true labels for these digits in the dataset, using a BCNN with $\mathcal{N}(0, \frac{15^2}{d_j})$ (on the left), and $\text{Laplace}(0, \frac{10^2}{d_j})$ (on the right) prior distributions.*

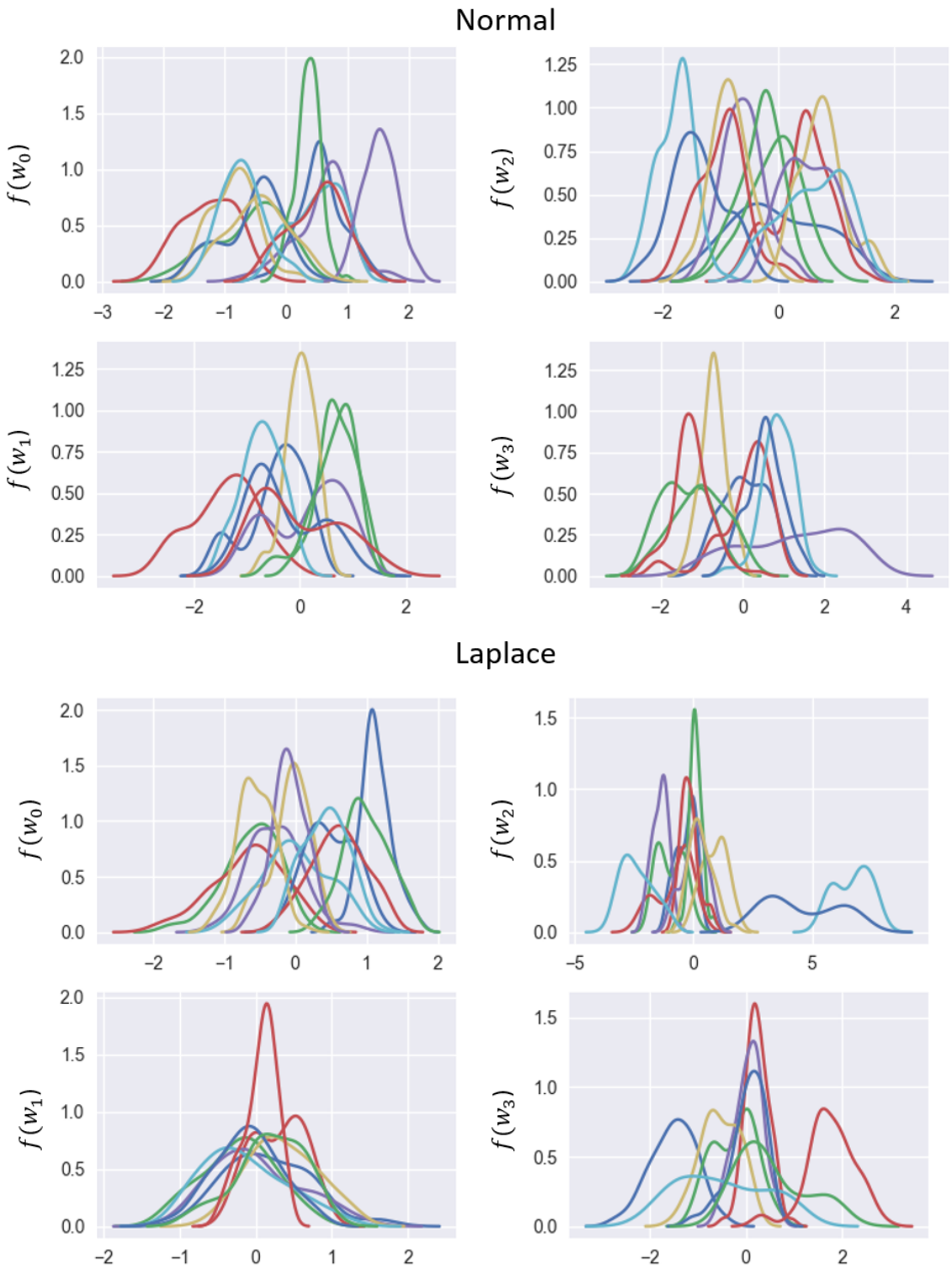

Figure 5.46 – *The two plots on the top correspond to the marginal distributions for the BCNN with* $\mathcal{N}(0, \frac{15^2}{d_j})$ *(on the top), and* $\mathrm{Laplace}(0, \frac{10^2}{d_j})$ *(on the bottom) prior distributions.*

Chapter 6

Conclusion

We saw that Dropout has a lot of interpretations, however the most convincing interpretation to myself is that Dropout enriches the input space (Konda et al., 2015). When expressing MC Dropout of linear regression into the two possible Bayesian interpretations provided in this report, the richness of Dropout came to light. We saw Dropout being expressed as a kind of mixture model, which is considerably more expressive than ordinary Bayesian linear regression.

In the final chapter we witnessed the difficulty associated with identifying convergence in the feed-forward NNs, which makes automation more difficult. Bayesian models using HMC or SGHMC to sample from the posterior do not suffer from this phenomenon. The more samples that are collected, the more accurate the prediction will be, assuming that the samples came from the invariant distribution and are sufficiently uncorrelated. However, this is incredibly computationally costly.

Identifying the correct learning rate in the feed-forward NNs can be very difficult and it requires a thorough search over the alternatives. The trained models suffered as a result of a poor choice of learning rate. We saw that the Adam optimiser consistently found better local solutions than SGD, which appeared to reach inferior local solutions.

As we discovered through the applications, tuning the hyper-parameters for HMC and SGHMC was extremely difficult. Before identifying good leapfrog step sizes and number of steps, a thorough and extremely costly search was conducted. The tuning of the hyper-parameters was also dependent on the choice of prior distribution, with Student T priors being more sensitive to the choice of hyper-parameters.

In terms of predictive accuracy, the BNNs arrived at very similar results to the feed-forward NNs for the smaller models. However, the larger more complex feed-forward CNN models were outperformed by the BCNN models, which did not seem to suffer from over-fitting. BNNs though still greatly suffer from extremely slow computation, which limited the

size of the models that we were able to consider, making algorithms like HMC and SGHMC less practical.

Furthermore, the difficulty of storing Bayesian NNs is a significant problem. Storing the samples that need to be used for prediction requires large amounts of memory. This means that it is extremely difficult to store the trained models to smart phones that have limited memory capacity. Feed-forward NNs do not suffer from this problem, which makes them more compelling for Smart phone App developers.

We also found that using a factored (VMF) approximating distribution for Variational Inference worked surprisingly well for the models with 1 hidden layer. This performed better than the feed-forward NNs in terms of prediction, for only a slight increase in computational cost. However, the VMF approximation obtained extremely poor results for the more complex (2 hidden layer) models, making their use quite impractical.

The choice of prior distribution was dependent on the problem, with Laplace priors slightly outperforming the models with the Normal priors in the MNIST setting with 2 hidden layers. For the models with 1 hidden layer this difference was indistinguishable. For the BCNNs, the model with Normal priors was significantly better than the model with Laplace priors. This reiterated our point that prior distributions should be chosen according to the task at hand.

In general, the models using Student T priors, performed worse than the other models. This was believed to be a consequence of the heavy tails of the T distributions. However, the regression models showed some weak evidence of signal identification properties that the Student T priors might possess.

It was not uncommon for the models using MCMC to struggle to obtain stable results in the observed (probability) traces, which could have emphasised the unidentifiability of certain models. Ultimately, this made the use of HMC more impractical in such scenarios. However, we found that decreasing the width of the hidden layers and increasing their depth lead to more stable results in terms of the output probability traces. Moreover, there seemed to be some evidence of multi-modality in the posterior distributions, meaning that it may have been inherently difficult to sample from such distributions using HMC.

Finally, the leapfrog step sizes used in the models explored we extremely small, in order to allow the HMC algorithm to produce proposals with non-zero acceptance probability. This resulted in highly correlated samples, making the HMC sampling scheme extraordinarily inefficient.

6.1 Further work

To counteract the limitations of having to tune the HMC hyper-parameters the use of the No-U-Turn Sampler (Hoffman and Gelman, 2014) is suggested for future work. For models using T distribution priors it would be interesting to leverage skip connections (Duvenaud et al., 2014, He et al., 2015), which might be able to help avoid large singular values in the Jacobian distribution and improve the stability of inference when using such heavy tailed distributions.

It would be interesting to see an exploration of less wide but deeper networks, as it is believed that they might give rise to more stable and accurate samples from the posterior distribution when using an MCMC sampling scheme such as HMC.

There is a lot of scope for improvement in the choice of prior distributions, for example using a more diverse selection of T distribution priors, such as ones with higher degrees of freedom. Alternatively, using a mixture of prior distributions, such as using Student T distribution priors only on the final layer of parameters and using Normal distribution priors for the earlier set of parameters. Another very interesting prior distribution that could be used in future work is a Spike and Slab prior distribution over the parameters.

Models that can handle multi-modality in the posterior distribution, by perhaps defining mixture of Gaussian or mixture of Laplace prior distributions over the parameters could be very promising. It would be interesting to see the use of HMC to sample from the posterior distribution, followed by VI technique to estimate the hyper-parameters for the mixture prior distributions.

We hope that in future we will see an application of the Bayesian interpretations of Dropout developed in this dissertation and to see how they compare with MC Dropout. More importantly, it would be interesting to compare the posterior distribution obtained for the Bayesian interpretation of Dropout to that of ordinary MC Dropout.

Appendices

Appendix A

Algorithms

A.1 Introduction

A.1.1 SGD

Algorithm 5: SGD algorithm

Data: $\mathcal{D}$
Input : η
initialise Θ;
for $i \leq n_{epoch}$ **do**
    shuffledata ; // Shuffle the dataset to randomise the batches
    for $j \leq n_{batches}$ **do**
        $\mathcal{D}^{(j)} \subseteq \mathcal{D} \setminus (\mathcal{D}^{(1)}, ..., \mathcal{D}^{(j-1)})$; // Select a batch
        forall $\theta \in \Theta$ **do**
            $\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \cdot \nabla_\theta L(\boldsymbol{\theta}; \mathcal{D}^{(j)})$; // Update parameter vector/matrix
        end
    end
end

Figure A.1 – n_{epoch} is the total number of cycles through the whole dataset. $n_{batches}$ corresponds to the total number of batches per epoch, which is typically the whole number greater than the $N/batchsize$, where N is the number of datapoints. η is the learning rate, $L(\theta; \mathcal{D}^{(j)})$ is the loss function for the j^{th} batch of the data, and Θ is the set of all parameters that need to be learned. For every epoch each of the batches are disjoint sets of fixed size from the set of all datapoints $\mathcal{D}$.

A.1.2 Momentum

Algorithm 6: Momentum algorithm

Data: $\mathcal{D}$

Input : η, γ

initialise Θ;

$\mathbf{v} = 0$;

for $i \leq n_{epoch}$ **do**

    shuffledata ; // Shuffle the dataset to randomise the batches

    for $j \leq n_{batches}$ **do**

        $\mathcal{D}^{(j)} \subseteq \mathcal{D} \setminus (\mathcal{D}^{(1)}, ..., \mathcal{D}^{(j-1)})$; // Select a batch

        forall $\boldsymbol{\theta} \in \Theta$ **do**

            $\mathbf{v} = \gamma \cdot \mathbf{v} + \eta \cdot \nabla_{\theta} L(\boldsymbol{\theta}; \mathcal{D}^{(j)})$;

            $\boldsymbol{\theta} = \boldsymbol{\theta} - \mathbf{v}$; // Update parameters

        end

    end

end

Figure A.2 – $n_{epoch}, n_{batches}, N, \eta$, and $L(\theta; \mathcal{D}^{(j)})$ are defined exactly as before. $\mathbf{v}$ is the momentum vector and γ is the friction coefficient.

A.1.3 Nesterov's Accelerated Gradient (NAG)

Algorithm 7: NAG algorithm

Data: $\mathcal{D}$

Input : $\eta, \gamma < 1$

initialise Θ;

$\mathbf{v} = 0$;

for $i \leq n_{epoch}$ **do**

  shuffledata ; `// Shuffle the dataset to randomise the batches`

  for $j \leq n_{batches}$ **do**

    $\mathcal{D}^{(j)} \subseteq \mathcal{D} \setminus (\mathcal{D}^{(1)}, ..., \mathcal{D}^{(j-1)})$; `// Select a batch`

    forall $\boldsymbol{\theta} \in \Theta$ **do**

      $\mathbf{v} = \gamma \cdot \mathbf{v} + \eta \cdot \nabla_{\theta} L(\boldsymbol{\theta} - \gamma \cdot \mathbf{v}; \mathcal{D}^{(j)})$;

      $\boldsymbol{\theta} = \boldsymbol{\theta} - \mathbf{v}$; `// Update parameters`

    end

  end

end

Figure A.3 – $n_{epoch}, n_{batches}, N, \eta$, and $L(\ \cdot\ ; \mathcal{D}^{(j)})$ are defined exactly as before. $\mathbf{v}$ is the momentum vector and γ is the friction coefficient.

A.1.4 Adam

Algorithm 8: Adam algorithm

Data: $\mathcal{D}$

Input : η, ϵ

Input : $\beta_1, \beta_2 \in [0, 1)$

initialise Θ;

$\mathbf{m} = \mathbf{0}$; // Initialising the first moment vector

$\mathbf{v} = \mathbf{0}$; // Initialising the second moment vector

$t = 0$; // Initialising the time-step

for $i \leq n_{epoch}$ **do**

  shuffledata ; // Shuffle the dataset to randomise the batches

  for $j \leq n_{batches}$ **do**

    $\mathcal{D}^{(j)} \subseteq \mathcal{D} \setminus (\mathcal{D}^{(1)}, ..., \mathcal{D}^{(j-1)})$; // Select a batch

    $t = t + 1$;

    forall $\theta \in \Theta$ **do**

      $\mathbf{g} = \nabla_\theta L(\boldsymbol{\theta}; \mathcal{D}^{(j)})$; // compute the gradients w.r.t. $\boldsymbol{\theta}$

      $\mathbf{m} = \beta_1 \cdot \mathbf{m} + (1 - \beta_1) \cdot \mathbf{g}$; // update biased first moment estimate

      $\mathbf{v} = \beta_2 \cdot \mathbf{v} + (1 - \beta_2) \cdot \mathbf{g}^2$; // update biased second moment estimate

      $\hat{\mathbf{m}} = \mathbf{m}/(1 - \beta_1^t)$; // bias corrected first moment estimate

      $\hat{\mathbf{v}} = \mathbf{v}/(1 - \beta_2^t)$; // bias corrected second moment estimate

      $\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \cdot \hat{\mathbf{m}}/(\sqrt{\hat{\mathbf{v}}} + \epsilon)$; // update the parameters

    end

  end

end

Figure A.4 – $n_{epoch}, n_{batches}, N, \eta$, and $L(\theta; \mathcal{D}^{(j)})$ are defined exactly as before. Note that operations on vectors such as the square, division and the square root are all element-wise operations, e.g. $\mathbf{g}^2 = \mathbf{g} \odot \mathbf{g}$. As suggested in (Kingma et al., 2015 [11]) good default settings are $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\eta = 0.001$, and $\epsilon = 10^{-8}$.

Appendix B

Derivations

B.1 Introduction

B.1.1 Derivative of the loss w.r.t. softmax input

We have that $\hat{y}_{im} = \frac{\exp(z_{im})}{\sum_{c=1}^{k}\exp(z_{ic})}$, hence:

$$\frac{\partial \hat{y}_{im}}{\partial z_{ij}} = \frac{\partial}{\partial z_{ij}}\left(\frac{e^{z_{im}}}{\sum_{c=1}^{k} e^{z_{ic}}}\right) = \frac{\delta_{mj}e^{z_{im}}}{\sum_{c=1}^{k} e^{z_{ic}}} - \frac{e^{z_{im}}e^{z_{ij}}}{\left(\sum_{c=1}^{k} e^{z_{ic}}\right)^2} = \frac{e^{z_{im}}}{\sum_{c=1}^{k} e^{z_{ic}}}\left(\delta_{mj} - \frac{e^{z_{ij}}}{\sum_{c=1}^{k} e^{z_{ic}}}\right)$$

$$= \hat{y}_{im}(\delta_{mj} - \hat{y}_{ij}) = \begin{cases} \hat{y}_{im}(1-\hat{y}_{im}), & \text{if } m = j \\ \hat{y}_{im}\hat{y}_{ij}, & \text{if } m \neq j \end{cases} \tag{B.1}$$

$$NB: \quad L = -\sum_{l=1}^{n}\sum_{c=1}^{k} y_{lc}\log(\hat{y}_{lc}) \Rightarrow \frac{\partial L}{\partial z_{ij}} = -\sum_{l=1}^{n}\delta_{li}\sum_{c=1}^{k}\frac{\partial}{\partial z_{ij}} y_{ic}\log(\hat{y}_{ic})$$

$$= -\sum_{c=1}^{k}\frac{\partial}{\partial z_{ij}} y_{ic}\log(\hat{y}_{ic}) \overset{\text{Chain rule}}{=} -\sum_{c=1}^{k}\frac{y_{ic}}{\hat{y}_{ic}}\frac{\partial y_{ic}}{\partial z_{ij}} = -\frac{y_{ij}}{\hat{y}_{ij}}\frac{\partial y_{ij}}{\partial z_{ij}} - \sum_{c\neq j}\frac{y_{ic}}{\hat{y}_{ic}}\frac{\partial y_{ic}}{\partial z_{ij}}$$

$$\overset{\text{from } (B.1)}{=} -y_{ij} + y_{ij}\hat{y}_{ij} + \sum_{c\neq j} y_{ic}\hat{y}_{ij} = \hat{y}_{ij}\underbrace{\left(\sum_{c=1}^{k} y_{ic}\right)}_{=1} - y_{ij} = \hat{y}_{ij} - y_{ij} \tag{B.2}$$

$$\Rightarrow \frac{dL}{d\mathbf{Z}} = \hat{\mathbf{Y}} - \mathbf{Y}$$

B.1.2 Derivative of the loss w.r.t. the hidden layer

$$NB: \qquad \mathbf{Z} = \mathbf{HW} + \mathbf{b} \Rightarrow \mathbf{z_i} = \mathbf{h_i^T W} + \mathbf{b} \Rightarrow z_{ik} = \sum_j h_{ij}w_{jk} + b_k$$

$$\frac{\partial z_{ik}}{\partial h_{im}} = \sum_j \frac{\partial}{\partial h_{im}}\left(h_{ij}w_{jk} + b_k\right) = \sum_j \delta_{jm}w_{jk} = w_{mk} \tag{B.3}$$

$$\frac{\partial L}{\partial h_{ik}} \overset{\text{Chain rule}}{=} \sum_j \frac{\partial L}{\partial z_{ij}} \frac{\partial z_{ij}}{\partial h_{ik}} \overset{(B.2)\&(B.3)}{=} \sum_j (\hat{y}_{ij} - y_{ij}) w_{kj}$$

$$\Rightarrow \frac{\partial L}{\partial \mathbf{H}} = \mathbf{W^T}(\hat{\mathbf{Y}} - \mathbf{Y}) \tag{B.4}$$

B.1.3 Derivative of the loss w.r.t. the bias of the output layer

From before $\mathbf{Z} = \mathbf{HW} + \mathbf{b} \Rightarrow z_{ik} = \sum_j h_{ij} w_{jk} + b_k$

$$\frac{\partial z_{ik}}{\partial b_m} = \sum_j \frac{\partial}{\partial b_m} \left(h_{ij} w_{jk} + b_k \right) = \delta_{km} \tag{B.5}$$

$$\frac{\partial L}{\partial b_m} \overset{\text{Chain rule}}{=} \sum_j \frac{\partial L}{\partial z_{ij}} \frac{\partial z_{ij}}{\partial b_m} \overset{(B.2)\&(B.5)}{=} \sum_j \delta_{jm} (\hat{y}_{ij} - y_{ij}) = \hat{y}_{im} - y_{im}$$

$$\Rightarrow \frac{\partial L}{\partial \mathbf{b}} = \hat{\mathbf{Y}} - \mathbf{Y} \tag{B.6}$$

B.1.4 Derivative of the loss w.r.t. the weight of the output layer

$\mathbf{Z} = \mathbf{HW} + \mathbf{b}$, using standard matrix calculus:

$$\frac{\partial \mathbf{Z}}{\partial \mathbf{W}} = \mathbf{H^T} \tag{B.7}$$

$$\frac{\partial L}{\partial \mathbf{W}} \overset{\text{Chain rule}}{=} \frac{\partial \mathbf{Z}}{\partial \mathbf{W}} \frac{\partial L}{\partial \mathbf{Z}} \overset{(B.2)\&(B.7)}{=} \mathbf{H^T}(\hat{\mathbf{Y}} - \mathbf{Y})$$

B.1.5 Derivative of the loss w.r.t. the ReLU module

Suppose the input to the ReLU module is $\mathbf{M}$, and we have that $\mathbf{H} = \text{relu}(\mathbf{M})$

$$\frac{\partial h_{ij}}{\partial m_{ik}} = \delta_{jk} \mathbb{1}_{\{m_{ik} > 0\}} \tag{B.8}$$

$$\frac{\partial L}{\partial m_{ik}} \overset{\text{Chain rule}}{=} \sum_j \frac{\partial L}{\partial h_{ij}} \frac{\partial h_{ij}}{\partial m_{ik}} \overset{(B.8)}{=} \sum_j \frac{\partial L}{\partial h_{ij}} \delta_{jk} \mathbb{1}_{\{m_{ik} > 0\}}$$

$$= \frac{\partial L}{\partial h_{ik}} \mathbb{1}_{\{m_{ik} > 0\}} \overset{(B.4)}{=} \sum_j (\hat{y}_{ij} - y_{ij}) w_{kj} \mathbb{1}_{\{m_{ik} > 0\}}$$

$$\Rightarrow \frac{\partial L}{\partial \mathbf{M}} = \mathbf{W^T}(\hat{\mathbf{Y}} - \mathbf{Y}) \odot \mathbb{1}_{\{\mathbf{M} > \mathbf{0}\}} \tag{B.9}$$

Where $\mathbb{1}_{\{\mathbf{M} > \mathbf{0}\}}$ is a matrix with zeros where the entries of $\mathbf{M}$ are equal to or less than zero and ones where they are positive.

$$\left[\mathbb{1}_{\{\mathbf{M}>\mathbf{0}\}}\right]_{ij} = \begin{cases} 1, & \text{if } m_{ij} > 0 \\ 0, & \text{if } m_{ij} \leq 0 \end{cases}$$

Alternatively if we are using the **softplus** non-linearity we have that $\mathbf{H} = \log(\mathbf{1} + \mathbf{e}^{\mathbf{M}})$, where $\mathbf{1}$ corresponds to a matrix with all entries equal to one. In this case $\frac{\partial \mathbf{H}}{\partial \mathbf{M}} = \frac{\mathbf{1}}{\mathbf{1}+\mathbf{e}^{-\mathbf{M}}}$, which corresponds to the elementwise division.

$$\Rightarrow \frac{\partial L}{\partial \mathbf{M}} = \mathbf{W}^{\mathbf{T}}(\hat{\mathbf{Y}} - \mathbf{Y}) \odot \frac{\mathbf{1}}{\mathbf{1} + \mathbf{e}^{-\mathbf{M}}} \tag{B.10}$$

We note that since the cross-entropy is the negative log-likelihood for the Softmax likelihood, we can re-use most of these derivatives for the implementation of HMC and we therefore choose not to go into detailed derivations of the derivatives for the energy functions in this dissertation.

Appendix C

Plots

Any supplementary graphics and plots not in the main body of the dissertation can be found in this section. This section is primarily intended to provide a more comprehensive understanding of the models discussed in the dissertation.

C.1 Chapter 2

This section of the Appendix contains supplementary plots from Chapter 2.

C.1.1 1 hidden layer neural networks

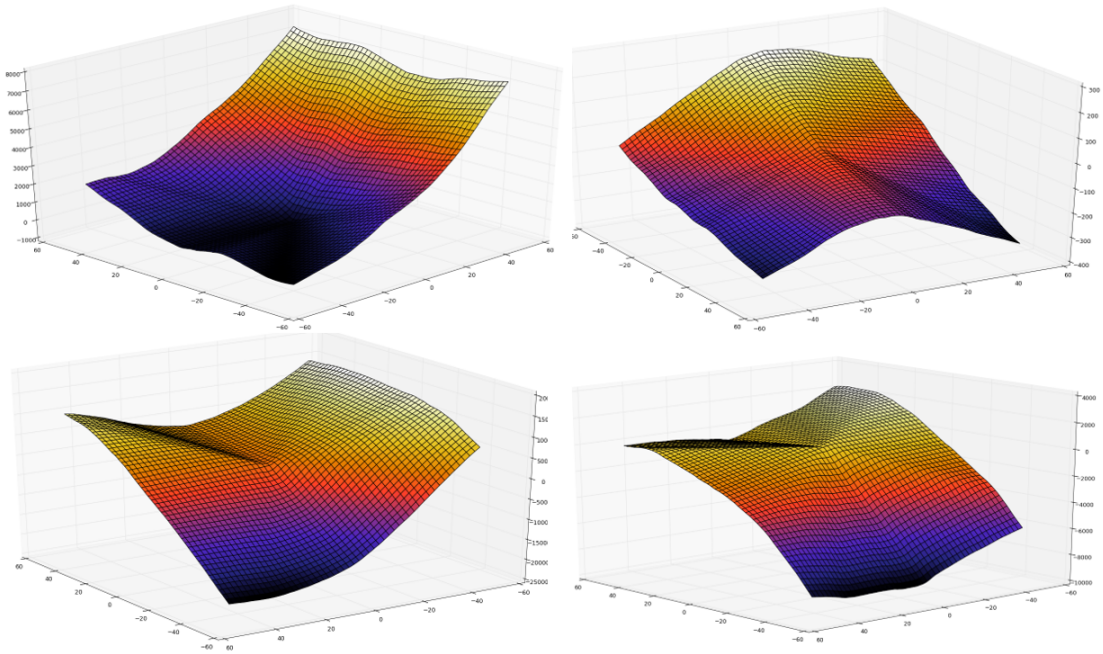

Figure C.1 – *Drawn functions from 1 hidden layer BNNs with 10000 hidden units. The drawn functions are evaluated by taking the average of 100 samples. The top left plot has a* $\mathcal{N}(\mathbf{0}, 10^2/d_j\mathbf{I}_{d_j})$ *prior distribution over the weights, whilst the bottom left plot has a* $\mathcal{N}(\mathbf{0}, 20^2/d_j\mathbf{I}_{d_j})$ *prior distribution over the weights. The top right plot has a* $\mathrm{Laplace}(\mathbf{0}, 5^2/d_j\mathbf{I}_{d_j})$ *prior over the weights, whilst the bottom right plot has a* $\mathrm{Laplace}(\mathbf{0}, 10^2/d_j\mathbf{I}_{d_j})$ *prior over the weights. Where* d_j *is the number of inputs from the previous layer (the number of columns of the units being multiplied by the weight matrix. All the networks use a ReLU activation functions for the hidden layer, and the input region is -60 to +60.*

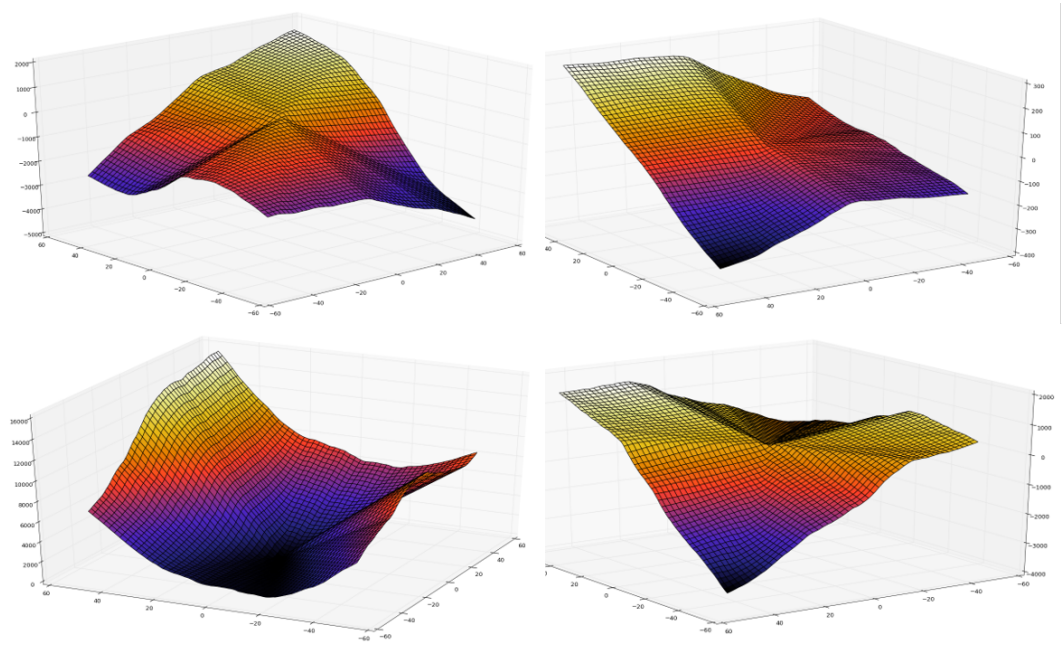

Figure C.2 – *Drawn functions from 1 hidden layer BNNs with 10000 hidden units. The drawn functions are evaluated by taking the average of 1000 samples. The same prior distributions as in figure C.1 are used, with the difference that the networks use a Softplus activation functions for the hidden layer.*

C.1.2 2 hidden layer neural networks

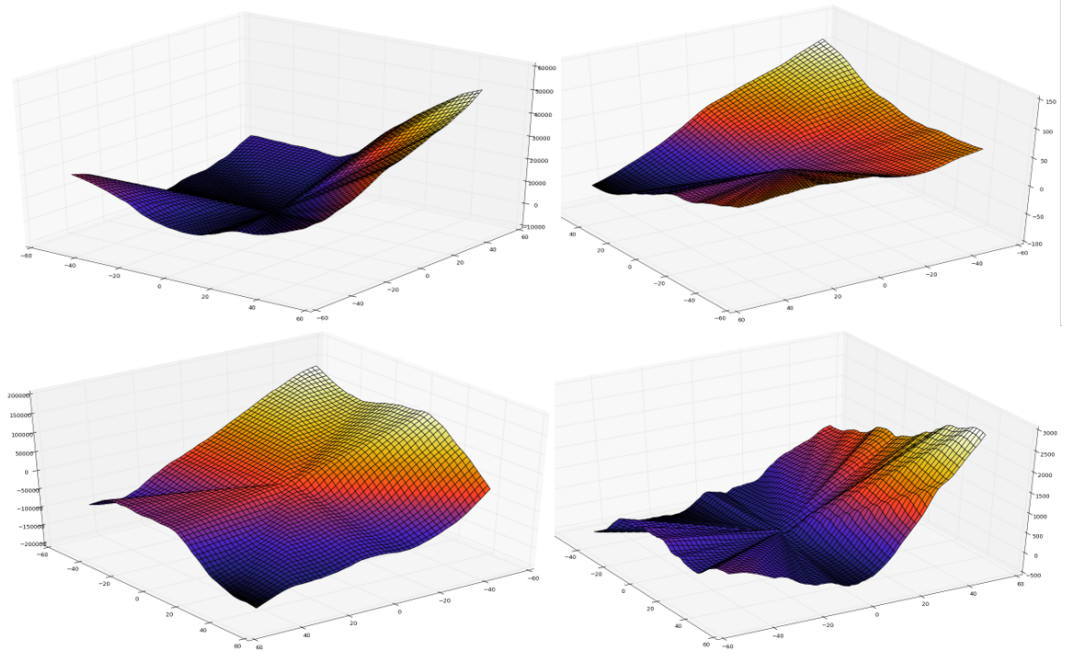

Figure C.3 – *Drawn functions from 2 hidden layer BNNs with 10000 hidden units for each of the hidden layers (same for all drawn functions). The drawn functions are evaluated by taking the average of 100 samples. The top left plot has a* $\mathcal{N}(\mathbf{0}, 10^2/d_j\mathbf{I}_{d_j})$ *prior distribution over the weights, whilst the bottom left plot has a* $\mathcal{N}(\mathbf{0}, 20^2/d_j\mathbf{I}_{d_j})$ *prior distribution over the weights. The top right plot has a* $\mathrm{Laplace}(\mathbf{0}, 10^2/d_j\mathbf{I}_{d_j})$ *prior over the weights, whilst the bottom right plot has a* $\mathrm{Laplace}(\mathbf{0}, 20^2/d_j\mathbf{I}_{d_j})$ *prior over the weights. Where* d_j *is the number of inputs from the previous layer (the number of columns of the units being multiplied by the weight matrix. All the networks use Softplus activation functions for both the hidden layers, and the input region is -60 to +60.*

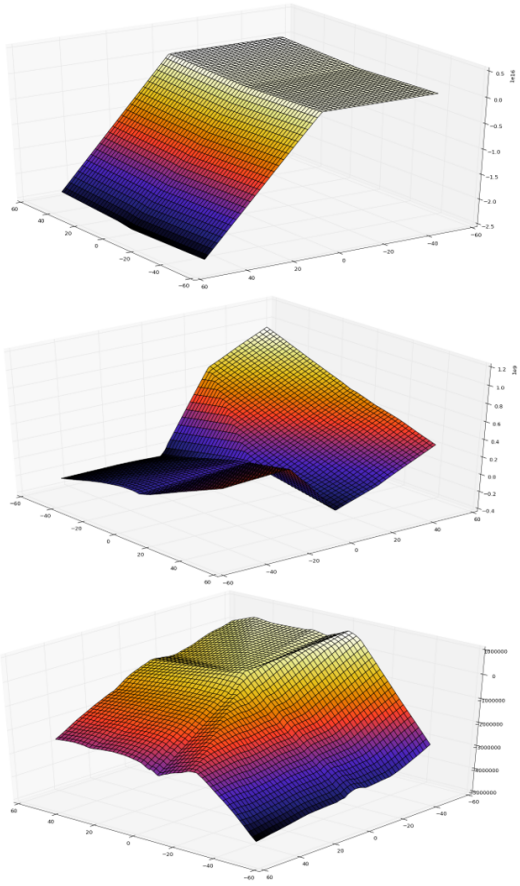

Figure C.4 – *Drawn functions from 2 hidden layer BNNs with 1000 hidden units for each of the hidden layers. The drawn functions are evaluated by taking the average of 1000 samples. The prior distributions for the weigths are standard Student T distributions with 0.5, 1, and 1.5 degrees of freedom for the top, centre, and bottom plots respectively. All the networks use Softplus activation functions for the hidden layer, and have an input region from -60 to +60.*

C.2 Chapter 5

C.2.1 2 hidden layers NN

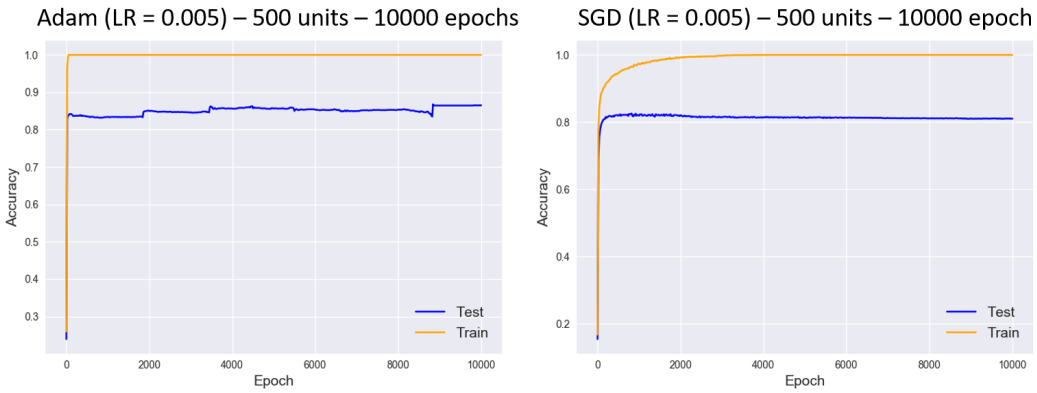

Figure C.5 – *The learning curves during training of models fitted using back propagation on the MNIST small dataset. All the models composed of 2 hidden layers each with 500 hidden units. The models were both trained for 10000 epochs and used Softplus non-linearities for both their hidden layers.*

Appendix D

Additional results

D.1 Chapter 5

D.1.1 1 hidden layer NN

Epochs	Hidden units	Optimiser	Learning Rate	MSE	Time (seconds)
1000	500	Adam	0.01	0.277	8
1000	500	SGD	0.01	0.485	7
1000	1000	Adam	0.01	0.482	8
1000	1000	SGD	0.005	0.321	7
3000	1000	Adam	0.01	0.242	22
3000	1000	SGD	0.005	0.315	20
5000	1000	Adam	0.01	0.238	37
5000	1000	SGD	0.005	0.338	34
5000	2000	Adam	0.005	0.245	40
3000	3000	Adam	0.005	0.357	25
3000	3000	SGD	0.001	0.572	22
5000	3000	Adam	0.005	0.377	41
5000	3000	SGD	0.001	0.536	37

Figure D.1 – Selected results for models with 1 hidden layer using Softplus non-linearities. Several different learning rates and optimisers that have not been presented in the main results section.

D.1.2 2 hidden layers NN

Number of epochs	Hidden units	Optimiser	Learning Rate	MSE	Time (seconds)
2000	100	Adam	0.005	0.208	12
5000	100	Adam	0.005	0.204	30
2000	200	Adam	0.005	0.209	14
3000	200	Adam	0.005	0.213	24
5000	200	Adam	0.005	0.201	32
2000	300	Adam	0.005	0.205	12
3000	300	Adam	0.005	0.202	22
5000	300	Adam	0.005	0.199	32
3000	400	Adam	0.001	0.208	24
5000	400	Adam	0.001	0.206	41
3000	500	Adam	0.01	0.207	23
5000	500	Adam	0.01	0.203	35
3000	400	SGD	0.005	0.285	21
5000	400	SGD	0.005	0.250	36
3000	500	SGD	0.005	0.246	21
5000	500	SGD	0.005	0.223	35

Figure D.2 – Selected (best) results for models with 2 hidden layers with an equal number of hidden units each using Softplus non-linearities. Several different learning rates and optimisers were used, only the best performing models are shown in this table. MSE refers to the final mean squared error on the test set. The time figure refers only to the training time for the models and excludes any other computations such as the time taken to evaluate the models.

Appendix E

Software, Code and References

E.1 Software and Code

This section will reference all the software and packages that were used. All the results provided in this dissertation were created using Python.

All the code for this dissertation can be found on the Github page https://github.com/cstavrou/Priors-and-Algorithms-for-Bayesian-Neural-Networks/ [56]. The page has different folders for each chapter of the dissertation, where all the relevant code and results can be found. For instructions on how to run the code, please read the "README" files available for each of the different scripts. All scripts can be run from the Terminal and many of the scripts take input arguments, which are described in both the code itself and on the Github page.

Python libraries

matplotlib
numpy
sys
timeit
random
cv2
os
sklearn
pandas
seaborn
tensorflow (1.2)
edward
statsmodels
pymc3
theano

GPU support software

CUDA® Toolkit 8.0
cuDNN v5.1

Bibliography

[1] Neal, R. M., '*Bayesian learning for neural networks'*, Thesis, University of Toronto, 1995.

[2] Gal, Y., '*Uncertainty in deep learning*', Thesis, University of Cambridge, 2017.

[3] McCulloch, W.S. and W. Pitts, '*A logical calculus of the ideas immanent in nervous activity'*, Bulletin of Mathematical Biophysics, vol. 5, 1943, pp. 115-133.

[4] Hebb, D. O., *The Organization of Behaviour: A Neuropsychological Theory*, New York, Wiley, 1949.

[5] Rosenblatt, F., '*The Perceptron - a perceiving and recognizing automaton'*, Report No. 85-460-1, Cornell Aeronautical Laboratory, 1957.

[6] Rosenblatt, F., *Principles of Neurodynamics*, Washington, Spartan Books, 1961.

[7] Rumelhart, D. E., G. E. Hinton, and R. J. Williams, '*Learning internal representations by error-propagation'*, Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Vol. 1, 1986, pp. 318-362.

[8] Linnainmaa, S., 'The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors', Master's Thesis, University of Helsinki, 1970.

[9] Widrow, B., '*An adaptive "Adaline" neuron using chemical "memistors"*', Technical Report No. 1553-2, 1960.

[10] Nesterov, Y., '*A method of solving a convex programming problem with convergence rate O(1/k^2)*', Soviet Mathematics Doklady, Vol. 27, 1983, pp. 372–376.

[11] Kingma, D. P., Ba, J. L. '*Adam: a Method for Stochastic Optimization'*, 3^{rd} International Conference on Learning Representations. San Diego, 2015.

[12] Fukushima, K. '*Cognitron: A Self-organizing Multilayered Neural Network'*, Biological Cybernetics, Vol. 20, 1975, pp. 121-136.

[13] Dugas, C. et al., '*Incorporating Second-Order Functional Knowledge for Better Option Pricing'*, Advances in Neural Information Processing Systems, Vol. 13 (NIPS 2000), 2001, pp. 1-7.

[14] Neal, R. M., '*Probabilistic inference using Markov chain Monte Carlo methods*, Technical Report CRG-TR-93-1, University of Toronto, 1993.

[15] Gilks, W.R., S. Richardson, and D. Spiegelhalter, *Markov Chain Monte Carlo in Practice*, London, UK, Chapman & Hall, 1996.

[16] Bishop, C. M., Pattern Recognition and Machine Learning, Secaucus, NJ, USA, Springer-Verlag New York, Inc., 2006.

[17] Metropolis, N. et al., '*Equation of state calculations by fast computing machines'*, The Journal of Chemical Physics, Vol. 21, No. 6, 1953, pp. 1087-1092.

[18] Hastings, W. K., '*Monte Carlo sampling methods using Markov chains and their applications'*, Biometrika, Vol. 57, No. 1, 1970, pp. 97-109.

[19] Geman, S., and D. Geman, '*Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images*', IEEE Transactions on pattern analysis and machine intelligence, Vol. PAMI-6, No. 6, 1984, pp. 721-741.

[20] Duane, S., A. D. Kennedy, B. J. Pendleton, and D. Roweth, '*Hybrid Monte Carlo*', Physics Letters B, Vol. 195, No. 2, 1987, pp. 216-222.

[21] Jylänki, P., A. Nummenmaa, A. Vehtari, '*Expectation Propagation for neural networks with sparsity-promoting priors*', Journal of Machine Learning Research, Vol. 15, 2014, pp. 1849-1901.

[22] Kalaitzis, A., R. Silva, '*Flexible sampling of discrete data correlations without the marginal distributions*', Advances in Neural Information Processing Systems 26 (NIPS'13), 2013.

[23] Hinton, G. E., D. van Camp, '*Keeping the neural networks simple by minimizing the description length of the weights*', Proceedings '93 COLT, 1993, pp. 5-13.

[24] Peterson, C., J. R. Anderson, '*A mean field theory learning algorithm for neural networks*', Complex Systems, Vol. 1, 1987, pp. 995- 1019.

[25] Jordan M. I., Z. Ghahramani, T. S. Jaakkola, L. K. Saul, '*An introduction to variational methods for graphical models*', Machine Learning, Vol. 37, 1999, pp. 183–233.

[26] Kullback, S., R. A. Leibler, '*On information and sufficiency*', Annals of Mathematical Statistics, Vol. 22, No. 1, 1951, pp. 79–86.

[27] Mitchell, T. J., J. J. Beauchamp, '*Bayesian variable selection in linear regression*', Journal of the American Statistical Association, Vol. 83, No. 404, 1988, pp. 1023-1032.

[28] Ishwaran, H., J. S. Rao, '*Spike and slab variable selection: Frequentist and Bayesian strategies*', The Annals of Statistics, Vol. 33, No. 2, 2005, pp. 730–773.

[29] He, K. et al., '*Deep residual learning for image recognition*', arXiv:1512.03385v1, 2015.

[30] Duvenaud, D., O. Rippel, R. P. Adams, and Z. Ghahramani, '*Avoiding pathologies in very deep networks*', Proceedings of machine learning research, Vol. 33, 2014, pp. 202-210.

[31] Hubel, D. H., T. N. Wiesel, '*Receptive fields, binocular interaction, and functional architecture in the cat's visual cortex*', Journal of Physiology, Vol. 160, 1962, pp. 106-154.

[32] Fukushima, K., S. Miyake, '*Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position*', Pattern Recognition, Vol. 15, 1982, pp. 455-469.

[33] Lecun, Y., L. Bottou, Y. Bengio, and P. Haffner, '*Gradient-based learning applied to document recognition*', Proceedings of the IEEE, Vol. 86, No. 11, 1998, pp. 2278-2324.

[34] Hinton, G. E. et al., '*Improving neural networks by preventing co-adaptation of feature detectors*', arXiv:1207.0580v1, 2012.

[35] Srivastava, N. et al., '*Dropout: A simple way to prevent neural networks from overfitting*', Journal of Machine Learning Research, Vol. 15, 2014, pp. 1929-1958.

[36] Konda, K., X. Bouthillier, R. Memisevic, and P. Vincent, '*Dropout as data augmentation*', arXiv:1506.08700v1, 2015.

[37] Simpson, D., H. Rue, A. Riebler, T. G. Martins, S. H. Sørbye, '*Penalising model component complexity: A principled, practical approach to constructing priors*', Statistical Science, Vol. 32, No. 1, 2017, pp. 1-28.

[38] Hernández-Lobato, J. M., R. P. Adams, '*Probabilistic Backpropagation for scaleable learning of Bayesian neural networks*', In ICML, 2015.

[39] Hernández-Lobato, J. M., D. Hernández-Lobato, A. Suárez, '*Expectation Propagation in Linear Regression Models with Spike-and-slab Priors*', Machine Learning, Vol. 99, No. 3, 2015, pp. 437-487.

[40] Blundell, C., J. Cornebise, K. Kavukcuoglu, D. Wierstra, '*Weight uncertainty in neural networks*', Proceedings of the 32nd International Conference on Machine Learning, JMLR: W&CP, Vol. 37, 2015.

[41] Myshkov, P., S. Julier, '*Posterior distribution analysis for Bayesian inference in neural networks*', Workshop on Bayesian Deep Learning, NIPS 2016.

[42] Minka, T. P., '*Bayesian model averaging is not model combination*', 2002, https://tminka.github.io/papers/minka-bma-isnt-mc.pdf, (accessed 17 September 2017).

[43] Baldi, P., P. Sadowski, '*Understanding Dropout*', Advances in Neural Information Processing Systems 26 (NIPS), 2013.

[44] Baldi, P., P. Sadowski, '*The dropout learning algorithm*', Artificial Intelligence, Vol. 210, 2014, pp. 78-122.

[45] Gal, Y., Z. Ghahramani, '*Dropout as a Bayesian approximation: Representing model uncertainty in deep learning*', arXiv:1506.02142v6, 2016.

[46] Welling, M., Y. W. Teh, '*Bayesian learning via stochastic gradient Langevin dynamics*', Proceedings of the 28th International Conference on Machine Learning, 2011.

[47] Kingma, D. P., T. Salimans, M. Welling, '*Variational dropout and the local reparameterization trick*', Advances in Neural Information Processing Systems 28 (NIPS 2015), 2015.

[48] Molchanov, D., A. Ashukha, D. Vetrov, '*Variational Dropout sparsifies deep neural networks*', Proceedings of the 34th International Conference on Machine Learning, Vol. 70, 2017, pp. 2498-2507.

[49] Li, Y., Y. Gal, '*Dropout inference in Bayesian neural networks with alpha-divergences*', Proceedings of the 34th International Conference on Machine Learning, Vol. 70, 2017, pp. 2052-2061.

[50] Gal, Y., '*Dropout As A Bayesian Approximation: Code*', https://github.com/yaringal/DropoutUncertaintyCaffeModels, 2015.

[51] Silva, R., R. Gramacy, '*Gaussian process structural equation models with latent variables*', Proceedings of the 26th Conference on Uncertainty on Artificial Intelligence, UAI 2010.

[52] Ding, W. G., '*Draw convnet*', https://github.com/gwding/draw_convnet, 2016.

[53] Gerven, M. V., B. Cseke, R. Oostenveld, T. Heskes, '*Bayesian source localization with the Multivariate Laplace prior*', Advances in Neural Information Processing Systems 22 (NIPS 2009), 2009.

[54] Ingraham, J. B., D. S. Marks, '*Variational Inference for Sparse and Undirected Models*', Proceedings of the 34th International Conference on Machine Learning, PMLR 70, 2017, pp. 1607-1616.

[55] MacKay, D. J., '*Bayesian interpolation*', Neural Computation, Vol. 4, 1992, pp. 415-447.

[56] Stavrou, C., '*Priors and Bayesian Neural Networks*, https://github.com/cstavrou/Priors-and-Algorithms-for-Bayesian-Neural-Networks, 2017.

[57] Chen, T., E. B. Fox, C. Guestrin, '*Stochastic gradient Hamiltonian Monte Carlo*', arXiv:1402.4102v2, 2014.

[58] Geyer, C. et al., Handbook of Markov Chain Monte Carlo, Boca Raton, FL, USA, Chapman & Hall/CRC, 2011.

[59] Blei, D. M., A. Kucukelbir, J. D. McAuliffe, '*Variational Inference: A Review for Statisticians*', *arXiv preprint* arXiv:1601.00670v5, 2017.

[60] Ripley, B. D., '*Stochastic Simulation*', Chichester, UK, John Wiley & Sons, Inc., 1987.

[61] Hoffman, M. D., A. Gelman, '*The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo*', Journal of Machine Learning Research, Vol. 15, 2014, pp. 1593-1623.

[62] Salvatier, J., T. V. Wiecki, C. Fonnesbeck, '*Probabilistic programming in Python using PyMC3*', PeerJ Computer Science 2:e55, $\mathrm{https://doi.org/10.7717/peerj\text{-}cs.55}$, 2016.

[63] Tran, D. et al., '*Edward: A library for probabilistic modeling, inference, and criticism*', arXiv:1610.09787v3, 2017.

[64] Abadi, M. et al., '*TensorFlow: Large-scale machine learning on heterogeneous systems*', 2015, Software available from: $\mathrm{http://tensorflow.org/}$.

[65] Abadi, M. et al., '*Tensorflow: A system for large-scale machine learning*', Proceedings of the 12^{th} USENIX Symposium on OSDI'16, 2016, pp. 265-283.

[66] Kucukelbir, A. R. Ranganath, A. Gelman, D. M. Blei, '*Automatic Variational Inference in Stan*', NIPS'15 Proceedings of the 28th International Conference on Neural Information Processing Systems, 2015, pp. 568-576.

[67] Choo, K., '*Learning hyper-parameters for neural network models using Hamiltonian dynamics*', Thesis, University of Toronto, 2000.