Aalto University
School of Science
Master's Programme in Computer, Communication and Information Sciences

Trung Trinh

Scalable Bayesian neural networks

Master's Thesis
Espoo, May 18, 2021

Supervisor: Professor Samuel Kaski
Advisor: Markus Heinonen, PhD

Aalto University
School of Science
Master's Programme in Computer, Communication and Information Sciences

ABSTRACT OF MASTER'S THESIS

Author: Trung Trinh

Title:
Scalable Bayesian neural networks

Date: May 18, 2021 **Pages:** 69

Major: Machine Learning, Data Science and Artificial Intelligence **Code:** SCI3044

Supervisor: Professor Samuel Kaski

Advisor: Markus Heinonen, PhD

The ability to output accurate predictive uncertainty estimates is vital to a reliable classifier. Standard neural networks (NNs), while being powerful machine learning models that can learn complex patterns from large datasets, do not possess such ability. Therefore, one cannot reliably detect when an NN makes a wrong prediction. This shortcoming prevents applying NNs in safety-critical domains such as healthcare and autonomous vehicles.

Bayesian neural networks (BNNs) have emerged as one of the promising solutions combining the learning capacity of NNs with probabilistic representations of uncertainty. By treating its weights as random variables, a BNN produces over its outputs a distribution from which uncertainty can be quantified. As a result, a BNN can provide better predictive performance while being more robust against out-of-distribution (OOD) samples than a respective deterministic NN.

Unfortunately, training large BNNs is challenging due to the inherent complexity of these models. Therefore, BNNs trained by standard Bayesian inference methods typically produce lower classification accuracy than their deterministic counterparts, thus hindering their practical applications despite their potential.

This thesis introduces implicit Bayesian neural networks (iBNNs), which are scalable BNN models that can be applied to large architectures. This model considers weights as deterministic parameters and augments the input nodes of each layer with latent variables as an alternative method to induce predictive uncertainty. To train an iBNN, we only need to infer the posterior distribution of these low-dimensional auxiliary variables while learning a point estimate of the weights. Through comprehensive experiments, we show that iBNNs provide competitive performance compared to other existing scalable BNN approaches, and are more robust against OOD samples despite having smaller numbers of parameters. Furthermore, with minimal overhead, we can convert a pretrained deterministic NN to a respective iBNN with better generalisation performance and predictive uncertainty. Thus, we can use iBNNs with pretrained weights of state-of-the-art deep NNs as a computationally efficient post-processing step to further improve performance of those models.

Keywords: Bayesian neural network, deep learning, neural network, uncertainty quantification

Language: English

Preface

The work in this thesis was supported by the Academy of Finland (Flagship programme: Finnish Center for Artificial Intelligence FCAI, Grants 294238, 319264, 292334, 334600) with the computational resources provided by the Aalto Science-IT project. This work also resulted in a research paper for submission to a journal, of which I am the first author. All the materials in this thesis, including those from the research paper, are originally written by me.

I would like to thank my supervisor Prof. Samuel Kaski for his kind support and invaluable advice.

I would like to thank my advisor Dr. Markus Heinonen for his constant support, insightful feedback and discussions.

Finally, I would like to thank my parents and sister for their love and support.

Espoo, May 18, 2021

Trung Trinh

Abbreviations and Acronyms

AUPR Area Under the Precision-Recall curve. 53

BNN Bayesian neural network. 8–13, 15–18, 20, 22–30, 35, 36, 40, 45–49, 55, 56, 58, 60, 68

CNN Convolutional neural network. 58

DE Deep ensemble. 25–27, 46–48, 51, 52, 58, 68

ECE Expected calibration error. 37, 39, 42, 46, 47, 51, 54

ELBO Evidence lower bound. 13–16, 25, 27, 32, 36

GP Gaussian Process. 7

HMC Hamiltonian Monte Carlo. 17, 18, 22, 40

i.i.d. Independent and identically distributed. 11, 39, 46

iBNN Implicit Bayesian neural network. 9, 29–33, 35–37, 40–60, 68, 69

KL Kullback-Leibler. 13–16, 25, 32, 33, 36, 37, 41, 42, 45, 48, 49, 56, 57, 68, 69

MCMC Markov Chain Monte Carlo. 8, 12, 16, 19, 20, 23, 28

ML Machine learning. 7, 8

NLL Negative log-likelihood. 37, 39, 42, 45–47, 49, 51, 54

NN Neural network. 7–12, 15, 16, 18, 20, 22–27, 29, 35–37, 40, 46, 48, 55–58, 60

OOD Out of distribution. 7, 9, 20, 46, 53, 55, 57–60

PCA Principal component analysis. 45, 58

SG-HMC Stochastic Gradient Hamiltonian Monte Carlo. 18, 19

SG-MCMC Stochastic Gradient Markov Chain Monte Carlo. 19, 20, 23–25

SGD Stochastic Gradient Descent. 15, 19, 25–27, 33, 34

SGLD Stochastic Gradient Langevin Dynamics. 19

SWAG Stochastic Weight Averaging Gaussian. 26, 27, 45, 47, 48, 51–54, 57, 68

VI Variational inference. 8, 12–16, 19, 20, 23–28, 47, 56

Contents

Chapter 1

Introduction

Neural networks (NNs) are powerful machine learning (ML) models that can learn complex patterns from large amounts of data [1–3]. They achieve state-of-the-art performance in various tasks and domains, such as computer vision [4], natural language translation [5], and speech recognition [6], leading to the wide adoption of NNs in many commercial applications [7].

However, one must be careful when using deep NNs in production systems. Typically, in these systems, an NN will encounter diverse and unpredictable input samples. These samples can be divided into two main groups: *in-distribution* samples which are assumed to come from the same distribution as the training samples, and *out-of-distribution* (OOD) samples which come from a different distribution than the training data [8]. An NN should ideally be more uncertain when making predictions on OOD samples than in-distribution samples because the former have less in common with the training data than the latter. However, deep NNs tend to produce overconfident predictions on both in-distribution and OOD test samples [9–12]. It is, therefore, difficult to determine when a model actually makes a meaningful and trustworthy prediction. In other words, deep NNs lack the ability to output reliable predictive uncertainty [13, 14]. This shortcoming hinders the usage of NNs in automated decision making systems in safety-critical domains, such as self-driving vehicles [15] and healthcare [16]. In these systems, the ability to detect when a model should not be trusted is of the utmost importance [17], as they use the model's predictions to make decisions affecting the health and safety of a person. Therefore, these systems need ML models that can produce accurate uncertainty estimates in order to make informed decisions, such as asking for human intervention in cases of high uncertainty, thus preventing catastrophic consequences from happening due to faulty predictions from the model.

While existing probabilistic ML models such as Gaussian Processes (GPs) [18] can output reliable predictive uncertainty, they have limited applications to high-dimensional unstructured data such as images, audio, and natural languages. This is because of the difficulty in defining meaningful kernels for these types of data, as GPs rely on these kernels to measure the similarities between testing and training samples. Even though the applications of GPs to image classification are actively explored [19–21], their performance is still far below

the current state-of-the-art NNs. Therefore, it is more promising to develop NNs that can produce reliable uncertainty estimates.

Bayesian neural networks (BNNs) [22, 23] are one of the main types of NNs that can output reliable predictive uncertainty. Because an NN is an overparameterised model, there are many weight configurations consistent with the training data. Hence, there is uncertainty about which weight value provides the most suitable hypothesis for the task. A BNN quantifies this uncertainty by treating weights as random variables and attempting to learn their posterior distribution given the training data. Theoretically, this allows a BNN to consider all possible weight values and thus all possible hypotheses representable by the model's architecture when making predictions, instead of relying on any single one of them as in the case of deterministic NNs. As a result, in a BNN, the distribution over weights induces the distribution over outputs. Thus, weight uncertainty captured by the posterior distribution propagates into uncertainty in the model's predictions [24], allowing for a more accurate representation of uncertainty than a similar deterministic model [13, 25–28].

Unfortunately, the potential of BNNs is hindered by the challenges of inferring their posterior densities, especially in the regime of large NNs and datasets where NNs are most commonly applied. Modern deep NNs can contain more than tens of millions of parameters, meaning that their posterior distributions are very high-dimensional. Moreover, weights in an NN are arranged into a layered structure with non-linear activation functions in between. The posterior distribution of an NN, therefore, has a complicated multi-modal structure [24]. The complexity of the posterior weight distribution is one of the main reasons why standard approximate Bayesian inference methods such as mean-field variational inference (VI) and Markov Chain Monte Carlo (MCMC) struggle to train BNNs that can yield comparable performance to similar deterministic models in practice [29–31]. Furthermore, maintaining a distribution over weights almost always incurs higher computational and memory costs than a single point-estimate. For instance, if a BNN defines a Gaussian distribution with its own mean and variance parameters for each weight, a common practice in mean-field VI, it will have twice the number of parameters of a similar deterministic NN. For these reasons, ML practitioners might refrain from using BNNs in their applications despite the promise of better uncertainty estimates.

1.1 Motivation

BNNs combine the powerful learning capacity of deep NNs with probabilistic representations of uncertainty, which is important for applications of NNs to safety-critical tasks. However, training a large BNN on a large dataset is a difficult task, which limits the adoption of BNNs to commercial applications despite their potential. Therefore, in this thesis, we will focus on *scalable BNN models* that maintain good predictive performance and low computational complexity in the context of large NNs and datasets, while providing reliable uncertainty estimates.

1.2 Contributions

This thesis introduces *implicit Bayesian neural networks* (iBNNs), efficient BNNs that can scale up to large NNs and datasets. We achieve efficiency and scalability by introducing low-dimensional latent variables to a deterministic NN to induce predictive uncertainty, and framing the training problem as inferring the posterior distribution of these latent variables. Through comprehensive experiments, we show that iBNNs provide competitive predictive performance compared to other existing scalable BNN methods and are more robust against OOD samples despite having smaller numbers of parameters. Furthermore, we can easily convert a pretrained deterministic NN to a respective iBNN with minimal overhead. Thus, we can apply iBNNs to pretrained state-of-the-art deep NNs, which have been growing rapidly in size due to the abundance of training data and computational resources, as a simple post-processing method to further improve those models' performance.

1.3 Outline

We first discuss the necessary background on BNNs in Chapter 2. Chapter 3 then thoroughly presents all the details about iBNNs. In Chapter 4, we perform an in-depth analysis of the behaviour of these models. In Chapter 5, we rigorously evaluate the generalisation performance and predictive uncertainty of iBNNs under different settings. Chapter 6 discusses the advantages and disadvantages of iBNNs, provides additional insights into BNNs and presents possible future research directions. Chapter 7 concludes this thesis.

Chapter 2

Bayesian neural networks

In this chapter, we will discuss Bayesian neural networks (BNNs). We first provide a formal definition of BNNs in Section 2.1. We then present the training methods for BNNs in Section 2.2. Section 2.3 discuss uncertainty quantification in BNNs. Current challenges of BNNs will be outlined in Section 2.4. Finally, we present some recent works in scalable BNNs in Section 2.5.

2.1 Formal definition

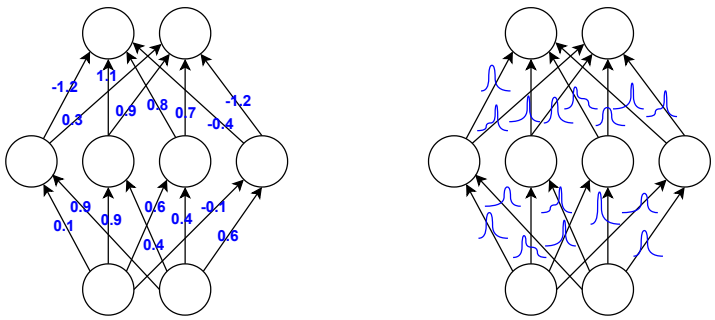

Figure 2.1: (a) In a standard NN, each weight has a fixed value. (b) In a BNN, each weight is a random variable with a distribution.

In a standard deterministic NN with a fixed architecture $\mathcal{M}$, the weight vector $\boldsymbol{\theta}$ is considered to have one true value which can be found through training on a dataset (Figure 2.1a). On the other hand, a *Bayesian neural network* (BNN) [22, 23, 32] treats its weights $\boldsymbol{\theta}$ as random variables with a prior probability distribution (Figure 2.1b). Thus, we can view a BNN as a probabilistic model with hidden variables $\boldsymbol{\theta}$. Training a BNN is equivalent to inferring the posterior distribution of the weights $\boldsymbol{\theta}$ using Bayes' theorem:

$$p(\boldsymbol{\theta}|\mathcal{D},\mathcal{M}) = \frac{p(\mathcal{D}|\mathcal{M},\boldsymbol{\theta})p(\boldsymbol{\theta}|\mathcal{M})}{p(\mathcal{D}|\mathcal{M})} = \frac{p(\mathcal{D}|\mathcal{M},\boldsymbol{\theta})p(\boldsymbol{\theta}|\mathcal{M})}{\int_{\boldsymbol{\theta}} p(\mathcal{D}|\mathcal{M},\boldsymbol{\theta})p(\boldsymbol{\theta}|\mathcal{M})d\boldsymbol{\theta}} \tag{2.1}$$

where $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N$ is the observed data and $\mathcal{M}$ denotes the architecture of the NN.

The likelihood term $p(\mathcal{D}|\mathcal{M}, \boldsymbol{\theta})$ depends on the weights $\boldsymbol{\theta}$, the architecture $\mathcal{M}$ and the learning problem. For example, in 1D homoscedastic regression problems where the variance of the output is assumed to be constant with respect to the input, the likelihood for each sample is typically a Gaussian distribution with known variance σ^2:

$$p(\mathcal{D}|\mathcal{M}, \boldsymbol{\theta}) = \prod_{i=1}^{N} \mathcal{N}(y_i|\mathbf{f}_{\boldsymbol{\theta}}^{\mathcal{M}}(\mathbf{x}_i), \sigma^2) \tag{2.2}$$

where $\mathbf{f}_{\boldsymbol{\theta}}^{M}$ denotes the function represented by an NN with the architecture $\mathcal{M}$ and weights $\boldsymbol{\theta}$. We can write the likelihood of the entire dataset as the product of the likelihoods of individual samples because these samples are assumed to be independent and identically distributed (i.i.d.). For classification problems, the likelihood for each sample is a Categorical distribution:

$$p(\mathcal{D}|\mathcal{M}, \boldsymbol{\theta}) = \prod_{i=1}^{N} \text{Categorical}(\mathbf{y}_i|\mathbf{f}_{\boldsymbol{\theta}}^{\mathcal{M}}(\mathbf{x}_i)) \tag{2.3}$$

The prior term $p(\boldsymbol{\theta}|\mathcal{M})$ defines the distribution of weights that we believe to be suitable to the task and the neural architecture $\mathcal{M}$ before observing any data. Choosing a good prior is very important, since the architecture $\mathcal{M}$ and the prior $p(\boldsymbol{\theta}|\mathcal{M})$ together define the prior distribution over functions $p(\mathbf{f})$. Specifically, the architecture $\mathcal{M}$ defines the set of possible functions (hypotheses) while the prior $p(\boldsymbol{\theta}|\mathcal{M})$ controls the prior probability of each function. A popular option for the prior is a diagonal multivariate Gaussian distribution with zero mean $p(\boldsymbol{\theta}|\mathcal{M}) = \mathcal{N}(\mathbf{0}, \sigma^2 I)$ [22, 23, 30, 33–36], which corresponds to L2 regularisation in deterministic NNs [22]. While this prior does express a reasonable belief, it does not take into account the structure $\mathcal{M}$ of the NN. Therefore, using this prior typically results in underperforming models in practice [31, 37, 38]. As a result, additional measures are applied to improve performance of a BNN with a Gaussian prior [31, 36]. We identify choosing priors as one of the main challenges of BNNs and we discuss more on this subject in Section 2.4.

The evidence $p(\mathcal{D}|\mathcal{M}) = \int p(\mathcal{D}|\mathcal{M}, \boldsymbol{\theta}) p(\boldsymbol{\theta}|\mathcal{M}) d\boldsymbol{\theta}$, or marginal likelihood, defines the expected probability assigned to the dataset $\mathcal{D}$ by a random function $\mathbf{f} \sim p(\mathbf{f})$ where $p(\mathbf{f})$ is the prior over functions induced by the neural architecture $\mathcal{M}$ and the weight prior $p(\boldsymbol{\theta}|\mathcal{M})$. In other words, the marginal likelihood $p(\mathcal{D}|\mathcal{M})$ indicates the suitability of the architecture $\mathcal{M}$ and the prior $p(\boldsymbol{\theta}|\mathcal{M})$ for the dataset $\mathcal{D}$. If we define the *support* of $p(\mathcal{D}|\mathcal{M})$ as the collection of datasets $\mathcal{D}$ where $p(\mathcal{D}|\mathcal{M}) > 0$, then the size of this collection reflects the learning capacity of the model [28]. For example, a linear model has a truncated support as it can only learn linear functions, whereas a deep NN with a non-linear activation can learn most complex patterns. The *inductive bias* of a model can be defined as how the probability mass of $p(\mathcal{D}|\mathcal{M})$ is distributed over the datasets in its support [28]. For example, a convolutional NN will assign higher marginal

likelihoods to image datasets, while a recurrent NN will favour sequential data. Intuitively, we should prefer a complex model so that we can incorporate every possible hypotheses that can explain the data to achieve good generalisation performance for a learning problem. Furthermore, we should carefully choose the network's structure and prior so that we can assign higher prior probabilities to hypotheses having reasonable inductive biases towards the data [28].

After we have obtained the posterior distribution, we can evaluate any function of interest $\mathbf{g}$ whose value depends on $\boldsymbol{\theta}$ by *marginalising over the posterior distribution*. In other words, we calculate the expected value of $\mathbf{g}$ with respect to the weight posterior:

$$\mathbb{E}_{p(\boldsymbol{\theta}|\mathcal{D},\mathcal{M})}[\mathbf{g}] = \int_{\boldsymbol{\theta}} \mathbf{g}(\boldsymbol{\theta}) p(\boldsymbol{\theta}|\mathcal{D},\mathcal{M}) d\boldsymbol{\theta} \tag{2.4}$$

For instance, to make a prediction on a new input $\mathbf{x}$, we use the *posterior predictive distribution* where the function $\mathbf{g}$ is the likelihood:

$$p(\mathbf{y}|\mathbf{x},\mathcal{D},\mathcal{M}) = \int_{\boldsymbol{\theta}} p(\mathbf{y}|\mathbf{x},\boldsymbol{\theta},\mathcal{M}) p(\boldsymbol{\theta}|\mathcal{D},\mathcal{M}) d\boldsymbol{\theta} \tag{2.5}$$

which is equivalent to taking the average of all the possible explanations $\mathbf{f}_{\boldsymbol{\theta}}^{\mathcal{M}}(\mathbf{x})$ weighted by the posterior distribution.

Inferring the posterior distribution of a BNN is a difficult problem due to the high dimensionality of this distribution and the non-linear nature of the model [23]. We therefore rely on approximate Bayesian inference methods to infer the posterior distribution $p(\boldsymbol{\theta}|\mathcal{D},\mathcal{M})$, which we will discuss in the next section.

2.2 Inference methods

In this section, we will discuss two approximate Bayesian inference methods for training BNNs. The first one is Variational Inference (VI), which attempts to find a tractable proxy distribution which closely resembles the intractable target distribution. The second method is Markov Chain Monte Carlo (MCMC) which approximates a distribution using a set of representative samples. We first discuss VI in Section 2.2.1. Section 2.2.2 will be about MCMC and we will compare these two methods in Section 2.2.3

2.2.1 Variational inference

2.2.1.1 Definition

Variational inference (VI) [39] is an efficient Bayesian inference method that transforms the problem of posterior inference into an optimisation problem. The main idea behind VI is that we can approximate an intractable target distribution by finding a simple and tractable one that most resembles the target distribution. Formally, let p denote the target distribution and $q_{\boldsymbol{\phi}}$ denote a tractable distribution parameterised by $\boldsymbol{\phi}$, both are distributions of a random variable $\mathbf{z}$. In VI terminology, $q_{\boldsymbol{\phi}}$ is called the *variational distribution* and $\boldsymbol{\phi}$ are the *variational*

parameters. To approximate p, we need to find the value of ϕ that minimises the Kullback-Leibler (KL) divergence between q_ϕ and p:

$$\phi^* = \underset{\phi}{\operatorname{argmin}} \operatorname{KL}\left[q_\phi(\mathbf{z})||p(\mathbf{z})\right] := \underset{\phi}{\operatorname{argmin}} \int_{\mathbf{z}} q_\phi(\mathbf{z}) \left(\log q_\phi(\mathbf{z}) - \log p(\mathbf{z})\right) d\mathbf{z} \quad (2.6)$$

In the case of BNNs whose true posterior is $p(\boldsymbol{\theta}|\mathcal{D}, \mathcal{M})$, if we denote $q_\phi(\boldsymbol{\theta})$ as the approximate posterior of the weights, then the KL divergence in the above equation will become:

$$\begin{aligned}
&\operatorname{KL}[q_\phi(\boldsymbol{\theta})||p(\boldsymbol{\theta}|\mathcal{D}, \mathcal{M}))] && (2.7)\\
&\quad := \int_{\boldsymbol{\theta}} q_\phi(\boldsymbol{\theta}) \left(\log q_\phi(\boldsymbol{\theta}) - \log p(\boldsymbol{\theta}|\mathcal{D}, \mathcal{M})\right) d\boldsymbol{\theta} && (2.8)\\
&\quad = \int_{\boldsymbol{\theta}} q_\phi(\boldsymbol{\theta}) \left(\log q_\phi(\boldsymbol{\theta}) - \log \tilde{p}(\boldsymbol{\theta}|\mathcal{D}, \mathcal{M}) + \log p(\mathcal{D}|\mathcal{M})\right) d\boldsymbol{\theta} && (2.9)\\
&\quad = \underbrace{\int_{\boldsymbol{\theta}} q_\phi(\boldsymbol{\theta}) \left(\log q_\phi(\boldsymbol{\theta}) - \log \tilde{p}(\boldsymbol{\theta}|\mathcal{D}, \mathcal{M})\right) d\boldsymbol{\theta}}_{-\mathcal{L}(\phi)} + \log p(\mathcal{D}|\mathcal{M}) && (2.10)
\end{aligned}$$

where $\tilde{p}(\boldsymbol{\theta}|\mathcal{D}, \mathcal{M})$ is the unnormalised posterior, which is the numerator in the right hand side of Equation (2.1):

$$\tilde{p}(\boldsymbol{\theta}|\mathcal{D}, \mathcal{M}) = p(\mathcal{D}|\boldsymbol{\theta}, \mathcal{M}) p(\boldsymbol{\theta}|\mathcal{M}) \quad (2.11)$$

Since the log evidence $\log p(\mathcal{D}|\mathcal{M})$ is a constant with respect to the weights $\boldsymbol{\theta}$, minimising the KL divergence in Equation (2.10) is equivalent to maximising $\mathcal{L}(\phi)$:

$$\begin{aligned}
\mathcal{L}(\phi) &:= \int_{\boldsymbol{\theta}} q_\phi(\boldsymbol{\theta}) \left(\log \tilde{p}(\boldsymbol{\theta}|\mathcal{D}, \mathcal{M}) - \log q_\phi(\boldsymbol{\theta})\right) d\boldsymbol{\theta} && (2.12)\\
&= \int_{\boldsymbol{\theta}} q_\phi(\boldsymbol{\theta}) \left(\log p(\mathcal{D}|\boldsymbol{\theta}, \mathcal{M}) + \log p(\boldsymbol{\theta}|\mathcal{M}) - \log q_\phi(\boldsymbol{\theta})\right) d\boldsymbol{\theta} && (2.13)\\
&= \int_{\boldsymbol{\theta}} q_\phi(\boldsymbol{\theta}) \log p(\mathcal{D}|\boldsymbol{\theta}, \mathcal{M}) d\boldsymbol{\theta} + \int_{\boldsymbol{\theta}} q_\phi(\boldsymbol{\theta}) \left(\log p(\boldsymbol{\theta}|\mathcal{M}) - \log q_\phi(\boldsymbol{\theta})\right) d\boldsymbol{\theta} && (2.14)\\
&= \mathbb{E}_{q_\phi(\boldsymbol{\theta})}\left[\log p(\mathcal{D}|\boldsymbol{\theta}, \mathcal{M})\right] - \operatorname{KL}\left[q_\phi(\boldsymbol{\theta})||p(\boldsymbol{\theta}|\mathcal{M})\right] && (2.15)
\end{aligned}$$

This function is called the evidence lower bound (ELBO), because it is the lower bound of the log evidence $p(\mathcal{D}|\mathcal{M})$ since the KL term in the left hand side of Equation (2.10) is non-negative:

$$\log p(\mathcal{D}|\mathcal{M}) = \mathcal{L}(\phi) + \operatorname{KL}\left[q_\phi(\boldsymbol{\theta})||p(\boldsymbol{\theta}|\mathcal{D}, \mathcal{M})\right] \geq \mathcal{L}(\phi) \quad (2.16)$$

We can see that as the ELBO approaches the log evidence, the forward KL divergence between the variational distribution and the target posterior approaches zero. In VI, we actually use the ELBO as the objective function instead of $\operatorname{KL}\left[q_\phi(\boldsymbol{\theta})||p(\boldsymbol{\theta}|\mathcal{D}, \mathcal{M})\right]$ to avoid calculating the intractable log evidence $\log p(\mathcal{D}|\mathcal{M})$. Equation (2.15) shows the ELBO is a sum of two terms:

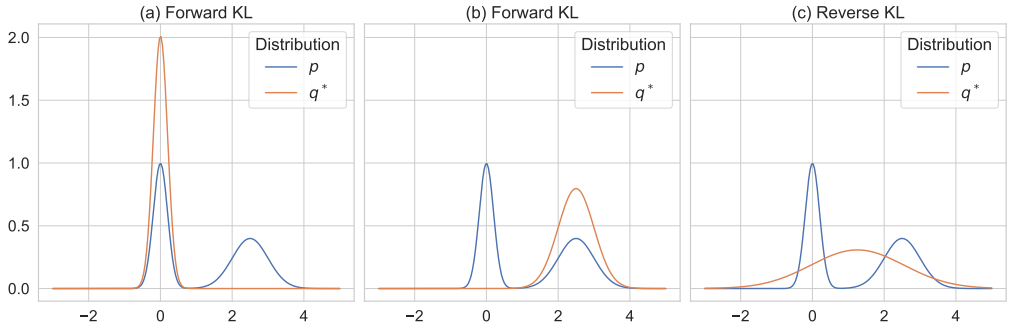

Figure 2.2: The optimal density q^* returned by the forward and reverse KL. Here we define the target density p as a mixture of two Gaussians, and the approximate density q as a single Gaussian. The first two plots show the results of the forward KL, which returns q^* that captures one of the two modes of p depending on how the mean of q is initialised. The third plot shows the result of the reverse KL, which returns q^* that attempts to capture all the support of p.

the expected log-likelihood $\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{\theta})}[\log p(\mathcal{D}|\boldsymbol{\theta}, \mathcal{M})]$ which encourages the variational posterior $q_{\boldsymbol{\phi}}(\boldsymbol{\theta})$ to fit the training data well, and the negative KL term $-\mathrm{KL}[q_{\boldsymbol{\phi}}(\boldsymbol{\theta})||p(\boldsymbol{\theta}|\mathcal{M})]$ which acts as a regulariser because it prevents $q_{\boldsymbol{\phi}}(\boldsymbol{\theta})$ from diverging from the prior $p(\boldsymbol{\theta}|\mathcal{D}, \mathcal{M})$. The ELBO is thus an intuitive objective: it attempts to find a proxy posterior $q_{\boldsymbol{\phi}}(\boldsymbol{\theta})$ which can explain the data and is grounded in the prior distribution.

2.2.1.2 The KL divergence

To understand the behaviour of VI, we need to examine the KL divergence [40]. Given the target distribution p and the approximate distribution q, the KL divergence between q and p has two properties making it suitable as an objective function: it only admits non-negative value and it equals zero when q and p are exactly the same. However, the KL divergence is not symmetric: $\mathrm{KL}[q||p] \neq \mathrm{KL}[p||q]$. We thus have two options to define our objective function for a pair of distributions. We call $\mathrm{KL}[q||p]$ the *forward KL* and $\mathrm{KL}[p||q]$ the *reverse KL* between q and p. The difference between the forward and reverse KL is that the former is the expected difference between the log density of q and p with respect to q, while the latter is the expected difference between the log density of p and q with respect to p. As a result, the distribution q^* that minimises the forward KL is very different from the one that minimises the reverse KL.

In the forward KL, the difference in log density is weighted by q; thus, for all z where $q(z) = 0$, the difference between $q(z)$ and $p(z)$ does not contributes to the overall function. The forward KL then focuses only on minimising this difference when $q(z) > 0$. It is, therefore, easier for q to minimise the forward KL if it focuses most of its mass in one region of high density of p and returns low densities everywhere else, resulting in a distribution q^* that underestimates the support of p. This behaviour is known as *zero-forcing* [41]. The reverse

KL, on the other hand, will stretch the distribution q to cover all the region where $p(z) > 0$, as it weights the log density difference using p. Consequently, there are regions where $p(z) = 0$ and $q(z) > 0$ ignored by the reverse KL. This behaviour is called zero-avoiding [41] and it results in q^* overestimating the support of p. We visualise the difference between forward and reverse KL in Figure 2.2.

In VI, the forward KL is used, which has the advantage of being simpler to evaluate than the reverse KL since we calculate the expectation with respect to the tractable variational distribution q instead of the intractable target distribution p. Expectation propagation, another approximate Bayesian inference algorithm, minimises the reverse KL [42].

2.2.1.3 Application of VI to BNNs

To train BNNs using VI, we need to address two practical problems:

- We need to efficiently calculate the gradient of the loss function with respect to the variational parameters.
- We need to modified the ELBO so that it is suitable for minibatch SGD training typically used with NNs.

The authors of [43] introduced one typical application of VI to BNNs effectively addressing these problems. We refer to this method as BNN-VI. We also note that this work is built upon the works in [44] and [45]. BNN-VI solves the first problem by using a factorised Gaussian distribution as the variational posterior:

$$q_{\boldsymbol{\phi}}(\boldsymbol{\theta}) = \prod_{i=1}^{P} \mathcal{N}(\theta_i|\mu_i, \sigma_i^2) \tag{2.17}$$

where P is the number of weights in the network. This allows the usage of the reparameterisation trick [46, 47] to efficiently calculate the gradient of the loss function with respect to the variational parameters $\boldsymbol{\phi} = \{(\mu_i, \sigma_i^2)\}_{i=1}^{P}$. Specifically, in the forward pass, the sample of each weight θ_i is generated by:

$$\theta_i = \mu_i + \sigma_i \epsilon, \qquad \epsilon \sim \mathcal{N}(0, 1) \tag{2.18}$$

which allows a simple calculation of the gradient of the loss function $\mathcal{L}$ with respect to the mean μ_i and standard deviation σ_i:

$$\frac{\partial \mathcal{L}}{\partial \mu_i} = \frac{\partial \mathcal{L}}{\partial \theta_i}, \qquad \frac{\partial \mathcal{L}}{\partial \sigma_i} = \frac{\partial \mathcal{L}}{\partial \theta_i}\epsilon \tag{2.19}$$

Since each weight θ_i in the BNN is represented by a Gaussian distribution with a separate mean μ_i and variance σ_i^2, the number of parameters in the BNN is twice that of a similar deterministic NN. To solve the second problem, BNN-VI uses a modified negative ELBO as the loss function:

$$\mathcal{L}_{\pi_i}(\mathcal{D}_i, \boldsymbol{\theta}) = -\frac{1}{|\mathcal{D}_i|}\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{\theta})}\left[\log p(\mathcal{D}_i|\boldsymbol{\theta}, \mathcal{M})\right] + \pi_i \mathrm{KL}\left[q_{\boldsymbol{\phi}}(\boldsymbol{\theta})||p(\boldsymbol{\theta}|\mathcal{M})\right] \tag{2.20}$$

where $\mathcal{D}_i$ denotes the i-th subset of the full dataset $\mathcal{D}$. The loss function above differs from the ELBO in Equation (2.15) in that it calculates the expected log-likelihood of a minibatch $\mathcal{D}_i$ instead of the full dataset $\mathcal{D}$, and it introduces a weighting coefficient π_i for the KL term. We need the coefficient π_i to reduce the influence of the KL term on the variational parameters, because the expected log-likelihood is calculated over a different minibatch in each update while the KL term stays the same. BNN-VI calculates the coefficient of the i-th iteration in one epoch as $\pi_i = \frac{2^{M-i}}{2^M-1}$ where M is the number of minibatches. Within each epoch, this scheme allows the weights to be more influenced by the KL term initially and become more influenced by the data in the latter stage. Practically, the loss function in Equation (2.20) is approximated as follows:

$$\widehat{\mathcal{L}}_{\pi_i}(\mathcal{D}_i, \boldsymbol{\theta}) = \frac{1}{|\mathcal{D}_i|} \sum_{j=1}^{|\mathcal{D}_i|} \Big(-\log p(\mathcal{D}_i^j | \boldsymbol{\theta}^j, \mathcal{M}) + \log q_{\boldsymbol{\phi}}(\boldsymbol{\theta}^j) - \log p(\boldsymbol{\theta}^j | \mathcal{M}) \Big) \tag{2.21}$$

where $\mathcal{D}_i^j$ is the j-th observation in $\mathcal{D}_i$ and $\boldsymbol{\theta}^j$ is the j-th sample drawn from the variational posterior $q_{\boldsymbol{\phi}}(\boldsymbol{\theta})$. For the prior $p(\boldsymbol{\theta}|\mathcal{M})$, the authors of [43] choose the mixture of Gaussians: $p(\boldsymbol{\theta}|\mathcal{M}) = \rho\mathcal{N}(\mathbf{0}, \sigma_1^2 I) + (1-\rho)\mathcal{N}(\mathbf{0}, \sigma_2^2 I)$ where $\sigma_1^2 \ll \sigma_2^2$.

BNN-VI is efficient and can theoretically applied to large NNs. In practice, however, it only provides comparable performance to deterministic NNs on small NN architectures and simple datasets despite having twice the number of parameters [43]. For more complex datasets and larger architectures, this method returns underperforming models [13, 30, 48, 49]. This is because the factorised Gaussian posterior causes high variance in the gradient estimates preventing training from properly converging [30, 50–52]. Furthermore, the factorised posterior does not capture the correlations among the weights which might contribute to the bad performance of BNN-VI [22, 35, 53]. These challenges are addressed in depth in Section 2.4.

2.2.2 Markov Chain Monte Carlo

2.2.2.1 Definition

Markov Chain Monte Carlo (MCMC) is a family of algorithms that construct a Markov chain whose stationary distribution is the target distribution $p(\boldsymbol{\theta})$. Using this chain, we can efficiently draw samples from the target distribution. Let $\Theta = \{\boldsymbol{\theta}_k\}_{k=1}^K$ denote the set of K samples drawn from the constructed Markov chain. Using these samples, we can approximate the expected value of any function $\mathbf{g}(\boldsymbol{\theta})$ with respect to p using the *MCMC estimator*:

$$\widehat{\mathbf{g}}_K = \frac{1}{K} \sum_{k=1}^{K} \mathbf{g}(\boldsymbol{\theta}_k) \tag{2.22}$$

which converges to the true expectation as $K \to \infty$:

$$\lim_{K\to\infty} \widehat{\mathbf{g}}_K = \mathbb{E}_{p(\boldsymbol{\theta})}\left[\mathbf{g}(\boldsymbol{\theta})\right] = \int_{\boldsymbol{\theta}} \mathbf{g}(\boldsymbol{\theta}) p(\boldsymbol{\theta}) d\boldsymbol{\theta} \tag{2.23}$$

Currently, Hamiltonian Monte Carlo (HMC) [54] is one of the best methods for sampling in high-dimensional spaces, because it leverages the gradient information from the target density to increase efficiency. This makes HMC a suitable algorithm for training BNNs, and the application of HMC to BNNs is first investigated in [23]. We will present HMC in the next section.

2.2.2.2 Hamiltonian Monte Carlo

Hamiltonian Monte Carlo (HMC) [54] uses Hamiltonian mechanics to derive a procedure that samples from a target distribution $p(\boldsymbol{\theta})$ using its gradient information. Instead of directly samples from $p(\boldsymbol{\theta})$, HMC chooses to sample from the *canonical distribution* $\pi(\boldsymbol{\theta}, \mathbf{v})$, which is a joint distribution between the random variables of interest $\boldsymbol{\theta}$ and the auxiliary random variables $\mathbf{v}$ whose marginal distribution $\pi(\boldsymbol{\theta})$ is exactly the target distribution $p(\boldsymbol{\theta})$:

$$\pi(\boldsymbol{\theta}, \mathbf{v}) = \pi(\mathbf{v}|\boldsymbol{\theta})\pi(\boldsymbol{\theta}) = \pi(\mathbf{v}|\boldsymbol{\theta})p(\boldsymbol{\theta}) \tag{2.24}$$

The auxiliary variables $\mathbf{v}$ are called the *momentum variables* and they are introduced to aid in the definition of the dynamical system. Likewise, the variables of interest $\boldsymbol{\theta}$ are considered the *position variables*. Both $\boldsymbol{\theta}$ and $\mathbf{v}$ have the same number of dimensions D. To generate samples from the canonical distribution $\pi(\boldsymbol{\theta}, \mathbf{v})$ using the Hamiltonian system, we first consider the Hamiltonian function $H(\boldsymbol{\theta}, \mathbf{v})$, the total energy of the system, as the negative log density of $\pi(\boldsymbol{\theta}, \mathbf{v})$:

$$\begin{aligned} H(\boldsymbol{\theta}, v) &:= -\log \pi(\mathbf{v}|\boldsymbol{\theta}) - \log p(\boldsymbol{\theta}) && (2.25) \\ &= K(\mathbf{v}) + U(\boldsymbol{\theta}) && (2.26) \end{aligned}$$

The first term $K(\mathbf{v}) = -\log \pi(\mathbf{v}|\boldsymbol{\theta})$ is considered the *kinetic energy* and the second term $U(\boldsymbol{\theta}) = -\log p(\boldsymbol{\theta})$ is the *potential energy* of the system. While the potential energy is the negative log density of the distribution of interest $p(\boldsymbol{\theta})$, the kinetic energy usually takes the form of an unnormalised Gaussian distribution with zero mean and covariance matrix $\mathbf{M}$:

$$K(\mathbf{v}) = \frac{\mathbf{v}^\top \mathbf{M} \mathbf{v}}{2} \tag{2.27}$$

The new sample of $(\boldsymbol{\theta}, \mathbf{v})$ is then obtained through the Hamiltonian equations, which define the evolution of $\boldsymbol{\theta}$ and $\mathbf{v}$ over the time t:

$$\frac{d\boldsymbol{\theta}}{dt} = \frac{\partial H}{\partial \mathbf{v}} = \frac{\partial K}{\partial \mathbf{v}} = \mathbf{M}^{-1}\mathbf{v} \tag{2.28}$$

$$\frac{d\mathbf{v}}{dt} = -\frac{\partial H}{\partial \boldsymbol{\theta}} = -\frac{\partial U}{\partial \boldsymbol{\theta}} \tag{2.29}$$

These equations conserve the total energy of the system, meaning that $H(\boldsymbol{\theta}, \mathbf{v})$ remains constant as $\boldsymbol{\theta}$ and $\mathbf{v}$ are updated. This condition is needed so that the canonical distribution $\pi(\boldsymbol{\theta}, \mathbf{v})$ remains invariant under this dynamical

system [55], which means the target distribution $p(\boldsymbol{\theta})$ is the stationary distribution of this Markov chain. In reality, these equations are approximated using the *leapfrog integrator*, which is a numerical integrator suitable for this problem:

$$\mathbf{v}_{t+0.5} = \mathbf{v}_t - \frac{\epsilon}{2}\frac{\partial U}{\partial \boldsymbol{\theta}_t} \tag{2.30}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \epsilon \frac{\partial K}{\partial \mathbf{v}_{t+0.5}} \tag{2.31}$$

$$\mathbf{v}_{t+1} = \mathbf{v}_t - \frac{\epsilon}{2}\frac{\partial U}{\partial \boldsymbol{\theta}_{t+1}} \tag{2.32}$$

where ϵ is the step size. To generate a new sample, starting from an initial value $(\boldsymbol{\theta}_0, \mathbf{v}_0)$, the Hamiltonian equations are integrated for L steps using the leapfrog method which produces a new sample $(\boldsymbol{\theta}_L, \mathbf{v}_L)$. Here the step size ϵ and number of steps L are two important hyperparameters that need tuning to obtain good performance. However, this integrator introduces numerical errors, which can be corrected through the Metropolis update with the acceptance probability α:

$$\alpha = \min\left[1, \exp\left(-H(\boldsymbol{\theta}_L, \mathbf{v}_L) + H(\boldsymbol{\theta}_0, \mathbf{v}_0)\right)\right] \tag{2.33}$$

Overall, HMC contains two main steps: the first step proposes a new sample generated via a numerical integrator (the leapfrog method) to simulate the Hamiltonian dynamics for L steps, and the second step accepts or rejects this new sample to alleviate the numerical errors introduced by the integrator.

2.2.2.3 Application of MCMC to BNNs

To train a BNN, we need to infer its posterior distribution. Thus, we consider the potential energy $U(\boldsymbol{\theta})$ in HMC as the unnormalised negative log posterior:

$$U(\boldsymbol{\theta}) = -\log \tilde{p}(\boldsymbol{\theta}|\mathcal{D}, \mathcal{M}) = -\log p(\boldsymbol{\theta}|\mathcal{M}) - \log p(\mathcal{D}|\boldsymbol{\theta}, \mathcal{M}) \tag{2.34}$$

From here we can apply the HMC algorithm as presented in the previous section to sample from the posterior [23]. However, this method only works for small NNs and small datasets, since we need to calculate the likelihood of the entire dataset which has high computational complexity in cases of large datasets and large NNs [33, 34]. Therefore, we need a new sampling method that can use the stochastic gradients of minibatch training. Specifically, this method will need to use the potential energy calculated on a minibatch $\tilde{D}$ of random samples from $\mathcal{D}$:

$$\tilde{U}(\boldsymbol{\theta}) = -\log p(\boldsymbol{\theta}|\mathcal{M}) - \frac{|\mathcal{D}|}{|\tilde{\mathcal{D}}|}\log p(\tilde{\mathcal{D}}|\boldsymbol{\theta}, \mathcal{M}) \tag{2.35}$$

One such method was introduced in [34] and it is called *Stochastic Gradient Hamiltonian Monte Carlo* (SG-HMC). The authors of [34] noticed that if one naively uses the potential energy in Equation (2.35) in the Hamiltonian equations, the random noise in the stochastic gradient will increase the entropy of

the canonical distribution $\pi(\boldsymbol{\theta}, \mathbf{v})$, which means it is no longer invariant under this dynamics. To solve this problem, SG-HMC first assumes that the gradient noise $\delta(\boldsymbol{\theta})$ follows a Gaussian distribution $\mathcal{N}(\mathbf{0}, \mathbf{V}(\boldsymbol{\theta}))$, and employs a modified Hamiltonian dynamics that can use the stochastic gradients to sample from the target distribution:

$$\frac{d\boldsymbol{\theta}}{dt} = \mathbf{M}^{-1}\mathbf{v} \tag{2.36}$$

$$\frac{d\mathbf{v}}{dt} = -\frac{\partial \tilde{U}}{\partial \boldsymbol{\theta}} - \mathbf{C}\mathbf{M}^{-1}\mathbf{v} + \hat{\delta} \qquad \hat{\delta} \sim \mathcal{N}\left(\mathbf{0}, 2(\mathbf{C} - \widehat{\mathbf{B}})\right) \tag{2.37}$$

Here $\mathbf{C}$ is the user-defined *friction term*, $\widehat{\mathbf{B}} = 0.5\widehat{\mathbf{V}}(\boldsymbol{\theta})$ and $\widehat{\mathbf{V}}(\boldsymbol{\theta})$ is the empirical estimate of $\mathbf{V}(\boldsymbol{\theta})$. The discretised version of the above equations with step size ϵ is:

$$\Delta\boldsymbol{\theta} = \epsilon\mathbf{M}^{-1}\mathbf{v} \tag{2.38}$$

$$\Delta\mathbf{v} = -\epsilon\Delta_{\boldsymbol{\theta}}\tilde{U}(\boldsymbol{\theta}) - \epsilon\mathbf{C}\mathbf{M}^{-1}\mathbf{v} + \epsilon\hat{\delta} \qquad \hat{\delta} \sim \mathcal{N}\left(\mathbf{0}, 2(\mathbf{C} - \widehat{\mathbf{B}})\right) \tag{2.39}$$

which is equivalent to the following system of equations:

$$\Delta\boldsymbol{\theta} = \mathbf{m} \tag{2.40}$$

$$\Delta\mathbf{m} = -\lambda\Delta_{\boldsymbol{\theta}}\tilde{U}(\boldsymbol{\theta}) - \alpha\mathbf{m} + \gamma \qquad \gamma \sim \mathcal{N}(\mathbf{0}, 2(\alpha - \hat{\beta})\lambda) \tag{2.41}$$

where $\mathbf{m} = \epsilon\mathbf{M}^{-1}\mathbf{v}$, $\lambda = \epsilon^2\mathbf{M}^{-1}$, $\alpha = \epsilon\mathbf{M}^{-1}\mathbf{C}$ and $\hat{\beta} = \epsilon\mathbf{M}^{-1}\widehat{\mathbf{B}}$. From these equations, SG-HMC can be interpreted as sampling via adding stochastic noise γ to the gradient of SGD with momentum $1 - \alpha$ and learning rate λ [34]. Another stochastic gradient MCMC (SG-MCMC) method called *Stochastic Gradient Langevin Dynamics* (SGLD) [33], which predates SG-HMC, can similarly be considered as sampling via injecting stochastic noise to the gradient of vanilla SGD (without momentum).

In practice, performance of SG-HMC is very sensitive to hyperparameter choices, specifically with respect to the values of ϵ, $\mathbf{M}$, $\mathbf{C}$ and $\widehat{\mathbf{B}}$ [34, 56]. Regarding the step size ϵ, we can avoid the costly Metropolis update by annealing ϵ to zero during sampling, since the rejection rate decreases along with the step size [33]. However, ϵ should only be decreased to a small, non-zero value to prevent high correlations between samples, which introduces small sampling errors [33]. Additional guidelines on how to adaptively set the value for $\mathbf{M}$, $\mathbf{C}$ and $\widehat{\mathbf{B}}$ are provided in [56].

2.2.3 Comparison between VI and MCMC

The main drawback of VI is that it approximates the complex posterior with a simpler proxy distribution and therefore can never returns the exact target density [39]. On the other hand, MCMC does not make any assumption about the posterior and theoretically can recover the target density in the limit of infinite samples [57]. Nevertheless, it is more natural to extend VI to use stochastic gradients than MCMC [39, 43, 45]. As presented in the previous section, additional

mechanisms are needed to make sure that the target distribution is theoretically invariant under SG-MCMC, and even then the asymptotic guarantee is no longer hold true as there is a trade-off between sampling accuracy and efficiency [33, 34]. Furthermore, VI is generally faster than MCMC [39]. As a result, in Bayesian modelling, VI is more favourable when a large amount of data is present, and MCMC is better if accurate posterior inference is required [39].

Regarding applications in BNNs, having a parametric variational posterior allows VI to be applied in continual learning or transfer learning since the parametric posterior of the previous task can be used as the prior of the next task [58]. On the other hand, SG-MCMC methods produce better predictive performance than VI in conventional supervised learning tasks [59]. Therefore, MCMC is more suitable for standard learning tasks where the entire dataset is known beforehand, whereas VI is preferred if we need a parametric representation of the posterior for subsequent Bayesian updates [30, 58].

2.3 Uncertainty quantification

In this section, we will discuss *predictive uncertainty quantification* using BNNs. We will focus on uncertainty quantification in supervised learning tasks. The predictive uncertainty reflects how confident a model is in its prediction and can be divided into two major types [60]:

- *Aleatoric uncertainty* is the uncertainty in the data [25, 60]. Some possible reasons for this uncertainty includes: (i) the data is inherently stochastic; (ii) there are measurement errors in the features due to imprecise instruments; (iii) there are labelling errors in the target outputs; and (iv) there are missing features in the inputs that are useful in explaining the data. Since aleatoric uncertainty is inherent in the data, it cannot be reduced by collecting more samples.

- *Epistemic uncertainty* is the uncertainty in the modelling process, which comes from our limited knowledge about the underlying true model that generates the data [25, 60]. This uncertainty is high if we choose a complex hypothesis class such as NNs for our problem, since there are many models within the large hypothesis space that are equally good at explaining the limited number of training samples. Therefore, epistemic uncertainty can be reduced by collecting more training samples.

Together, these two types of uncertainty constitute the predictive uncertainty of a model. Decomposing the predictive uncertainty into these two parts allows us to identify the main source of uncertainty in the model's prediction with respect to an input sample, namely, if the uncertainty comes from noise in the sample or from the model itself. Typically, epistemic uncertainty increases if the input comes from a region of low density of training samples and thus is important for detecting OOD samples [25, 61], while aleatoric uncertainty reflects the amount of noise in the observation.

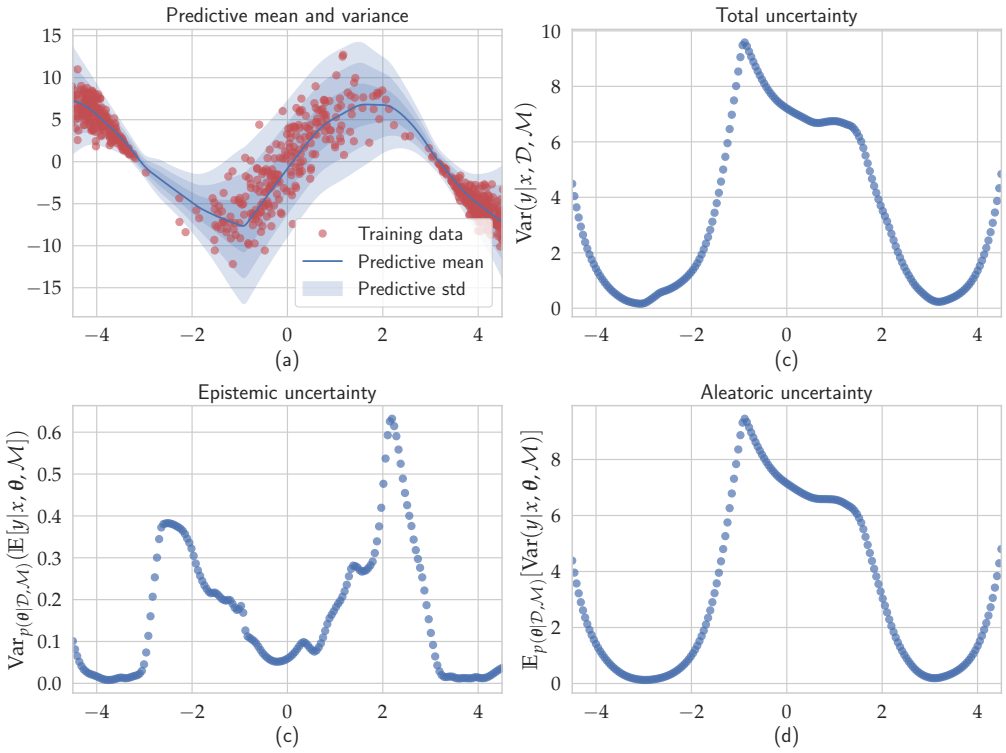

Figure 2.3: A simple 1D regression problem to demonstrate the behaviour of two types of uncertainty.

One approach to quantify and decompose the predictive entropy was introduced in [62], which uses the entropy $\mathrm{H}(\mathbf{y}|\mathbf{x},\mathcal{D},\mathcal{M})$ of the posterior predictive distribution $p(\mathbf{y}|\mathbf{x},\mathcal{D},\mathcal{M})$ as the measurement of total uncertainty:

$$\mathrm{H}(\mathbf{y}|\mathbf{x},\mathcal{D},\mathcal{M}) = -\int_{\mathbf{y}} p(\mathbf{y}|\mathbf{x},\mathcal{D},\mathcal{M}) \log p(\mathbf{y}|\mathbf{x},\mathcal{D},\mathcal{M}) d\mathbf{y} \tag{2.42}$$

The aleatoric uncertainty is defined as the expected entropy of the likelihood with respect to the posterior distribution $\mathbb{E}_{q(\boldsymbol{\theta}|\mathcal{D},\mathcal{M})}\left[\mathrm{H}(\mathbf{y}|\mathbf{x},\boldsymbol{\theta},\mathcal{M})\right]$ [62]:

$$\mathrm{H}(\mathbf{y}|\mathbf{x},\boldsymbol{\theta},\mathcal{M}) = -\int_{\mathbf{y}} p(\mathbf{y}|\mathbf{x},\boldsymbol{\theta},\mathcal{M}) \log p(\mathbf{y}|\mathbf{x},\boldsymbol{\theta},\mathcal{M}) d\mathbf{y} \tag{2.43}$$

$$\mathbb{E}_{p(\boldsymbol{\theta}|\mathcal{D},\mathcal{M})}\left[\mathrm{H}(\mathbf{y}|\mathbf{x},\boldsymbol{\theta},\mathcal{M})\right] = \int_{\boldsymbol{\theta}} \mathrm{H}(\mathbf{y}|\mathbf{x},\boldsymbol{\theta},\mathcal{M}) p(\boldsymbol{\theta}|\mathcal{D},\mathcal{M}) d\boldsymbol{\theta} \tag{2.44}$$

The epistemic uncertainty is then the difference between the total uncertainty and the aleatoric uncertainty, which is the conditional mutual information $\mathrm{I}(\mathbf{y},\boldsymbol{\theta}|\mathbf{x},\mathcal{D},\mathcal{M})$ between $\mathbf{y}$ and $\boldsymbol{\theta}$ given the test input $\mathbf{x}$, the architecture $\mathcal{M}$, and the training data $\mathcal{D}$ [62]:

$$\mathrm{I}(\mathbf{y},\boldsymbol{\theta}|\mathbf{x},\mathcal{D},\mathcal{M}) = \mathrm{H}(\mathbf{y}|\mathbf{x},\mathcal{D},\mathcal{M}) - \mathbb{E}_{p(\boldsymbol{\theta}|\mathcal{D},\mathcal{M})}\left[\mathrm{H}(\mathbf{y}|\mathbf{x},\boldsymbol{\theta},\mathcal{M})\right] \tag{2.45}$$

In practice, we approximate Equation (2.42) and Equation (2.44) using Monte Carlo estimators. For a simple 1D regression task, we can also use the variance of the predictive posterior $\mathrm{Var}(y|x,\mathcal{D},\mathcal{M})$ as indicator of uncertainty and

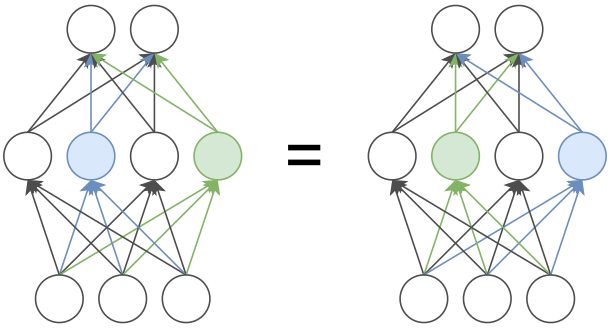

Figure 2.4: Illustration of one transformation in weight space conserving an NN's outputs. If we exchange the incoming and outgoing weights of the blue and green hidden nodes (represented by arrows of similar colours), the network's outputs remain unchanged.

decompose it into aleatoric and epistemic uncertainty using the law of total variance [62]:

$$\mathrm{Var}(y|x,\mathcal{D},\mathcal{M}) = \underbrace{\mathrm{Var}_{p(\boldsymbol{\theta}|\mathcal{D},\mathcal{M})}\left(\mathbb{E}\left[y|x,\boldsymbol{\theta},\mathcal{M}\right]\right)}_{\text{Epistemic uncertainty}} + \underbrace{\mathbb{E}_{p(\boldsymbol{\theta}|\mathcal{D},\mathcal{M})}\left[\mathrm{Var}(y|x,\boldsymbol{\theta},\mathcal{M})\right]}_{\text{Aleatoric uncertainty}} \tag{2.46}$$

To demonstrate the behaviour of epistemic and aleatoric uncertainty, we replicate a toy regression problem in [62], where we train a small feedforward NN with two hidden layers of size 20 and ReLU activation function on a 1D toy regression data generated from the function: $y = 7\sin(x) + 3|\cos(x/2)|\epsilon$ where $\epsilon \sim \mathcal{N}(0,1)$. We sample 750 values of x from an equally-weighted mixture of Gaussians with three components with means $(-4, 0, 4)$ and standard deviations $(0.4, 0.9, 0.4)$. Since the data is heteroscedastic, we train our network to output both the mean and standard deviation for each input. We choose HMC as our inference method because this is a small experiment. We plot the predictive mean and variance of the trained BNN as well as the uncertainty decomposition in Figure 2.3 which we calculate using Equation (2.46). From this figure, we can see that the aleatoric uncertainty reflects the amount of noise in an input sample, with the region of high aleatoric uncertainty in Figure 2.3d coincides with the highly noisy input region from -2 to 2. The two peaks of epistemic uncertainty Figure 2.3c correspond to the two regions of low density of training samples. The total uncertainty is shown in in Figure 2.3b, which is almost similar to the aleatoric uncertainty in Figure 2.3d since aleatoric uncertainty is much larger than epistemic uncertainty in this problem. Nonetheless, both types of uncertainty exhibit their typical behaviours as we have mentioned previously.

2.4 Challenges in training BNNs

Training a BNN is difficult due to the following reasons:

1. First, we have to perform probabilistic inference on a large number of random variables in the form of the NN's weights. The number of weights can range from a few hundred thousands for small NNs to more than tens of millions for large NNs.

2. Second, these random variables are arranged into a non-linear hierarchical structure defined by the NN architecture, which makes their posterior distribution highly multi-modal and introduces weight symmetries. Weight symmetries are transformations that can be applied to the weights of an NN without changing its outputs. For example, Figure 2.4 shows that if we exchange the incoming and outgoing weights of two nodes within a single hidden layer, the outputs of the network remain unchanged [63]. For a ReLU network, we can multiply the weights of a layer with a non-zero scalar α and multiply the weights of the next layer with $1/\alpha$ and the network will still represent the same function [63].

Hence, performing inference on BNNs presents challenges for both VI and MCMC in terms of computation cost and convergence. Furthermore, it is unclear how we can effectively define the prior for the weights of a BNN. Below, we will discuss each of these challenges in their respective subsections.

2.4.1 Computation cost

When considering computation cost, we need to analyze both the time and space complexity of the inference method.

Regarding time complexity, as we have presented in Section 2.2, both VI and MCMC needs adapting to use stochastic gradients calculated on data batches due to the high computational cost of computing the likelihood of the entire dataset using an NN [33, 34, 43, 45]. While this extension is straightforward for VI [43, 45], transforming MCMC to SG-MCMC introduces sampling errors thus breaking the asymptotic guarantee of this algorithm [33, 34].

In terms of space complexity, the large number of random variables prevents VI from using more expressive variational posteriors since they require a large number of variational parameters. Hence, a typical variational posterior is the factorised Gaussian distribution where each weight is represented by an independent Gaussian distribution with a separate mean and variance, doubling the number of parameters in the model. One major disadvantage is that this factorised Gaussian does not learn the correlations between the weights. Nevertheless, using a multivariate Gaussian posterior with a non-diagonal covariance matrix to capture the correlations between the weights is not an option due to the prohibitive quadratic cost in the number of parameters. This is one of the reasons why VI performs poorly in practice when being applied to BNNs [22, 35, 53]. Regarding MCMC, it could become expensive to store a large number of MCMC samples of large NNs since the number of stored parameters scales linearly with respect to the number of samples [64].

2.4.2 Convergence

As we have discussed as the end of Section 2.2.1.3, VI with a Gaussian posterior fails to converge properly when being applied to large NNs [30, 50–52]. This problems originates from the fact that in high-dimensional spaces, the Gaussian posterior concentrates most of its probability mass $p(\boldsymbol{\theta})d\boldsymbol{\theta}$ on a thin shell far away from the mean [65, 66]. As we draw samples from the Gaussian posterior to compute the Monte Carlo approximations for the loss function in Equation (2.21), these approximations will exhibit high variability, thus leading to high variance in the gradient estimates preventing training from proper convergence [30]. As a result, additional heuristics are needed in practice so that BNNs trained by VI can achieve comparable predictive performance compared to their deterministic counterparts [30, 31, 58].

While SG-MCMC does not suffer from severe convergence issues like VI, it could stuck in a local mode due to the high modality of the posterior distribution, especially since we usually anneal the step size to a small value to avoid the Metropolis update [33, 34]. A recent development in [36] proposed using a cyclical learning rate schedule for SG-MCMC. This learning rate schedule consists of repeated cycles where we first decay the learning rate to a small value for mode exploration and then reset the learning rate to the initial large value so that the chain can jump to another neighbourhood. While this scheme does allow efficient traversal between multiple posterior modes, SG-MCMC might explore modes representing equivalent functions due to weight symmetries, thereby reducing the effectiveness of Bayesian marginalisation and wasting computational effort. Furthermore, even with a cyclical learning rate schedule, weight samples in one chain still exhibit more functional similarity than weight samples collected from independent runs with different random seeds [67] (an approach called Deep ensembles presented in Section 2.5). Finally, as previously noted in Section 2.2.2.3, SG-MCMC needs extensive hyperparameter tuning to reach good predictive performance.

2.4.3 Prior specification

In classical Bayesian modelling, choosing the prior distribution is trivial because the models in these cases only contain small numbers of random variables with clearly defined roles and connections. Furthermore, the learning tasks of these models are often well-understood, thus allowing expert knowledge to be used for prior elicitation [68]. On the other hand, choosing a prior distribution for the weights of an NN is a very challenging task, since the sheer number of parameters and the complexity of the learning task make it impossible to understand the meaning of each individual weights to the task beforehand. Therefore, a typical approach is to assumed a Gaussian prior $\mathcal{N}(0, \sigma^2)$ for every weight [22, 23, 30, 33–36]. However, such simplistic prior specification can be detrimental to the model's predictive performance because the prior combined with the model's architecture controls the prior hypothesis space and the inductive bias of the model [28] as discussed in Section 2.1. This argument is supported by a systematic study in [31], which showed that the current bad

prior specification is at least partly responsible for the poor predictive performance of BNNs trained by SG-MCMC. To solve this issue, a current heuristic in SG-MCMC is to replace the standard log posterior $U(\boldsymbol{\theta}) = -\log p(\boldsymbol{\theta}|\mathcal{D}, \mathcal{M})$ with the cold posterior $U_T(\boldsymbol{\theta}) = -\log p(\boldsymbol{\theta}|\mathcal{D}, \mathcal{M})/T$ where the temperature T is set to a value smaller than 1. This posterior increases the influence of the likelihood while reducing the effect of the prior [31]. An equivalent heuristic in VI is to downweight the KL term in the ELBO [31]. Consequently, there is potential to improve predictive performance of BNNs with a suitable prior distribution [31, 37, 38].

2.5 Recent approaches

In this section, we outline some recent approaches in scalable BNNs and predictive uncertainty estimation, which will be used in the experiment section as baselines. We will focus on BNN approaches that return parametric approximations of the posterior, as our method also belongs to this category.

2.5.1 Deep ensembles

Perhaps the simplest way to estimate predictive uncertainty is running standard SGD training multiple times with different random seeds and initialisations to produce multiple trained deep NNs and then combining the predictions of these models via averaging. This method aptly named Deep ensembles (DEs) was introduced in [14]. The only disadvantage of DEs is that their computational and memory complexity scales linearly with respect to the ensemble size. On the other hand, DEs produce the best generalisation performance compared to other approaches [59], because each member in an ensemble represents a distinct mode in the loss landscape. Hence, DEs serve as references for us to see how close our method is to achieving the best possible results.

2.5.2 MC-Dropout

MC-Dropout [13] repurposes Dropout [69] as an approximate VI method. The intuition is that we can view each random dropout mask as corresponding to a sample from the variational posterior. Thus, this method consists of training a model with dropout applied before every weight layer and keeping dropout active during testing to draw multiple weight samples for predictive uncertainty estimation.

Formally, MC-Dropout approximates the posterior by multiplying each input node of each weight layer with a Bernoulli random variable. This is equivalent to using a mixture of two Gaussians as the posterior of each individual weight θ:

$$q(\theta) = \rho\mathcal{N}(0, \sigma^2) + (1 - \rho)\mathcal{N}(m, \sigma^2) \tag{2.47}$$

where ρ is the dropout rate and the variance σ^2 is set to a very small value.

MC-Dropout is a very efficient method, does not introduce additional variational parameters, and it can return good predictive performance in practice

when being applied to large NNs. In order to achieve good performance, however, we need to carefully select the dropout rate ρ, which controls both accuracy and uncertainty of the model [13]. This hyperparameter tuning can be expensive, especially since we might need to find the optimal dropout rate for each layer. In practice, MC-Dropout returns worse generalisation performance than Rank-1 BNNs and SWAG, two methods discussed below. Nonetheless, MC-Dropout has been successfully applied to computer vision [25] and reinforcement learning [70, 71] as a reliable method for uncertainty estimation.

2.5.3 Rank-1 BNNs

Rank-1 BNNs [49] approximate the weight posterior of each layer using a multiplicative rank-1 matrix as follows:

$$\mathbf{W} = \mathbf{U} \circ \left(\mathbf{r}\mathbf{s}^\top\right), \qquad \mathbf{r} \sim q(\mathbf{r}),\, \mathbf{s} \sim q(\mathbf{s}) \tag{2.48}$$

where $\mathbf{W}$ and $\mathbf{U}$ are $m \times n$ weight matrices, $\mathbf{r}$ is an m-dimensional random vector and $\mathbf{s}$ is an n-dimensional random vector. The posterior of $\mathbf{W}$ is induced by the posteriors of $\mathbf{r}$ and $\mathbf{s}$ as well as the deterministic matrix $\mathbf{U}$. VI is used to train Rank-1 BNNs where the parameters are the weights $\mathbf{U}$ and the variational parameters of $q(\mathbf{r})$ and $q(\mathbf{s})$.

The parameterisation of Rank-1 BNNs reduces the number of random variables within each layer from mn to $m + n$, allowing them to scale easily to large NNs without introducing too many additional parameters. Taking advantage of this reduction, the authors of [49] further placed a mixture of Gaussians posterior on the random variables $\mathbf{r}$ and $\mathbf{s}$ to increase the expressiveness of the model. Rank-1 BNNs return state-of-the-art performance on standard image classification benchmarks while having small numbers of variational parameters. Nonetheless, Rank-1 BNNs require replicating each minibatch K times during training to train all K components of the mixture posterior in parallel, thus increasing the training complexity. Their performance, moreover, is still worse than DEs, because a Rank-1 BNN can only approximate a local mode in the weight posterior.

2.5.4 SWAG

Stochastic Weight Averaging Gaussian (SWAG) [48] is an alternative approach to VI relying on the SGD iterates to approximate a local posterior mode using a multivariate Gaussian distribution. During training, starting from a pretrained solution, SWAG will run SGD with a constant learning rate to collect T iterates from the SGD trajectory, and use these iterates to approximate the mean and covariance of the Gaussian posterior. Specifically, given a model with P parameters, after each collected iterates $\boldsymbol{\theta}_i \in \mathbb{R}^P$ where $i = 1, \ldots, T$, the following update equations are applied:

$$\overline{\boldsymbol{\theta}} \leftarrow \frac{n\overline{\boldsymbol{\theta}} + \boldsymbol{\theta}_i}{n+1}, \quad \overline{\boldsymbol{\theta^2}} \leftarrow \frac{n\overline{\boldsymbol{\theta^2}} + \boldsymbol{\theta}_i^2}{n+1}, \quad \mathbf{D}^{(i)} \leftarrow \boldsymbol{\theta}_i - \overline{\boldsymbol{\theta}} \tag{2.49}$$

where $\overline{\boldsymbol{\theta}}$ and $\overline{\boldsymbol{\theta}^2}$ approximate the first and second moments and $\mathbf{D}$ is a $P \times T$ matrix. A new weight sample $\tilde{\boldsymbol{\theta}}$ is then generated as follows:

$$\boldsymbol{\Sigma}_{\text{diag}} = \text{diag}(\overline{\boldsymbol{\theta}^2} - \overline{\boldsymbol{\theta}}^2), \qquad \boldsymbol{\Sigma}_{\text{low-rank}} = \frac{1}{K-1}\widehat{\mathbf{D}}\widehat{\mathbf{D}}^\top \tag{2.50}$$

$$\tilde{\boldsymbol{\theta}} = \overline{\boldsymbol{\theta}} + \frac{\boldsymbol{\Sigma}_{\text{diag}}^{\frac{1}{2}}\mathbf{z}_1}{\sqrt{2}} + \frac{\widehat{\mathbf{D}}\mathbf{z}_2}{\sqrt{2(K-1)}}, \qquad \mathbf{z}_1 \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_P),\ \mathbf{z}_2 \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_K) \tag{2.51}$$

which is equivalent to sampling from $\mathcal{N}\left(\overline{\boldsymbol{\theta}}, \frac{1}{2}(\boldsymbol{\Sigma}_{\text{diag}} + \boldsymbol{\Sigma}_{\text{low-rank}})\right)$. Here $\widehat{\mathbf{D}}$ is a $P \times K$ matrix containing the last K columns of $\mathbf{D}$, meaning that only the last K SGD iterates are used to create a rank K approximation of the covariance matrix. The last equation allows sampling from a non-diagonal Gaussian posterior without explicitly storing the full covariance matrix, thus avoiding the prohibitive quadratic cost.

SWAG cannot reach performance of DEs since it only captures one mode in the posterior. However, SWAG has the same training cost as a similar deterministic model, can be applied to large NNs, and it achieves great performance in practice because it captures the correlations between the weights in the covariance matrix.

2.5.5 Radial BNNs

As we have mentioned in Section 2.4.2, samples from a high-dimensional Gaussian posterior are concentrated in a region far away from the mean, which causes high variance in the gradient estimates of the ELBO, preventing VI from convergence. Radial BNNs [30] tackle this challenge by using the radial posterior, a replacement for the factorised Gaussian posterior. Formally, the following equation is used to draw weight samples $\tilde{\boldsymbol{\theta}}$ from the radial posterior with mean $\boldsymbol{\mu}$ and variance $\boldsymbol{\sigma}^2$:

$$\tilde{\boldsymbol{\theta}}^{(\ell)} := \boldsymbol{\mu}^{(\ell)} + \boldsymbol{\sigma}^{(\ell)} \circ \frac{\boldsymbol{\epsilon}}{||\boldsymbol{\epsilon}||}|r|, \qquad r \sim \mathcal{N}(0,1),\ \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \tag{2.52}$$

Here $\tilde{\boldsymbol{\theta}}^{(\ell)}, \boldsymbol{\mu}^{(\ell)}, \boldsymbol{\sigma}^{(\ell)}$ and $\boldsymbol{\epsilon}$ are vectors of size d where d is the number of parameters in the ℓ-layer of the network, and $\circ$ denotes the Hadamard product. For each layer, the distance from the samples to the mean of this distribution is then explicitly controlled by the random variable r, thus stabilising the gradients and alleviating the converge issue of VI with large NNs. Experiments in [30] showed that the radial posterior helps VI scale to large architectures while producing competitive performance compared to deterministic models and MC-Dropout. Therefore, we choose Radial BNNs in our baselines as strong representatives of the standard VI method.

2.6 Summary

In this chapter, we reviewed the background on BNNs necessary for understanding the rest of this thesis. We first provided a formal definition of BNNs,

which treat their weights as random variables. Training a BNN is then equivalent to inferring the posterior distribution of its weights. We presented two standard approximate Bayesian inference methods: VI and MCMC, and we discussed how they can be applied to BNNs. Next, we explained two types of uncertainty: aleatoric and epistemic uncertainty, and how we can quantify each of them in a model's prediction. We then outlined some major challenges in training large BNNs. We finally presented some recent works on scalable BNNs which are used as baseline methods in our experiments.

Chapter 3

Implicit Bayesian neural networks

As we have discussed in the previous chapter, training BNNs is difficult because we need to infer the posterior of a large number of random variables. However, the eventual goal of treating weights as random variables is to induce a distribution over the outputs of the model so that we can estimate predictive uncertainty. Therefore, we propose to instead shift the stochasticity from the weights of each layer to its input nodes to achieve the same effect. We call this approach *node parameterisation*. Our motivation originates from a simple observation that the total number of nodes in an NN is much smaller than the total number of weights. By considering nodes as random variables while treating weights as deterministic parameters, we can substantially reduce the number of random variables to be inferred. Thus, we propose *implicit Bayesian neural networks* (iBNNs), BNN models that utilise node parameterisation to achieve efficiency and scalability to large NN architectures and datasets.

We first provide a formal definition of iBNNs in Section 3.1. We then explain the reasoning behind their formulation in Section 3.2. Section 3.3 and Section 3.4 present the training objective. Section 3.5 outlines the training procedure of iBNNs. Section 3.6 compares iBNNs to other methods. We conclude this chapter in Section 3.7.

3.1 Formal definition

Let $\boldsymbol{\theta} = \{(\mathbf{U}^{(\ell)}, \mathbf{v}^{(\ell)})\}_{\ell=1}^{L}$ denote all deterministic parameters of an iBNN with L layers, where $\mathbf{U}^{(\ell)}$ and $\mathbf{v}^{(\ell)}$ are the weights and biases of the ℓ-th layer. Let $\mathbf{Z} = \{(\mathbf{z}_{\mathbf{U}}^{(\ell)}, z_{\mathbf{v}}^{(\ell)})\}_{\ell=1}^{L}$ denote all latent variables in the model, where the $\mathbf{z}_{\mathbf{U}}^{(\ell)}$ and $z_{\mathbf{v}}^{(\ell)}$ are the latent variables corresponding to the weights $\mathbf{U}^{(\ell)}$ and biases $\mathbf{v}^{(\ell)}$. Let $\sigma^{(\ell)}$ be the activation function of the ℓ-th layer. Then an iBNN with L layers is defined as follows:

$$\mathbf{f}^{(0)}(\mathbf{x}) := \mathbf{x}, \tag{3.1}$$

$$\mathbf{f}^{(\ell)}(\mathbf{x}) = \sigma^{(\ell)}\big(\mathbf{U}^{(\ell)}\big(\mathbf{z}_{\mathbf{U}}^{(\ell)} \circ \mathbf{f}^{(\ell-1)}(\mathbf{x})\big) + \mathbf{v}^{(\ell)} z_{\mathbf{v}}^{(\ell)}\big), \tag{3.2}$$

$$\mathbf{z}_{\mathbf{U}}^{(\ell)} \sim p(\mathbf{z}_{\mathbf{U}}^{(\ell)}), \qquad z_{\mathbf{v}}^{(\ell)} \sim p(z_{\mathbf{v}}^{(\ell)}), \qquad \forall \ell = 1, .., L \tag{3.3}$$

where $\mathbf{x}$ is the input vector to the model and $\circ$ denotes the Hadamard product. Compared to a conventional BNN, here the predictive distribution of the model is induced by the latent variables $\mathbf{Z}$, while the weights and biases $\boldsymbol{\theta}$ are treated as *ordinary, non-stochastic parameters*, which can either be *optimised or initialised from pretrained solutions*. In the next section, we will explain our intuition for the formulation of iBNNs.

3.2 Layer-wise input priors

In this section, we explain our motivation for using multiplicative latent variables by showing a connection between an iBNN and a conventional BNN. To this end, we consider a single original layer in an iBNN whose input vector $\tilde{\mathbf{x}}$ is assumed to be stochastic and comes from a distribution $p(\tilde{\mathbf{x}}|\mathbf{x})$ where $\mathbf{x}$ is the unaltered input vector:

$$\mathbf{y} = \sigma(\mathbf{U}\tilde{\mathbf{x}}) \tag{3.4}$$
$$\tilde{\mathbf{x}} \sim p(\tilde{\mathbf{x}}|\mathbf{x}) \tag{3.5}$$

Here $\mathbf{U}$ is the weight matrix of dimensions $d_{out} \times d_{in}$. We omit the biases because we assume that the last element in the input vector $\mathbf{x}$ is an auxiliary feature with a constant value of 1, which means the bias vector is the last column of $\mathbf{U}$.

To find a suitable form of $p(\tilde{\mathbf{x}}|\mathbf{x})$, we compare this layer with a single-layer of a standard BNN

$$\mathbf{y} = \sigma(\mathbf{W}\mathbf{x}) \tag{3.6}$$
$$\mathbf{W} \sim p(\mathbf{W}) \tag{3.7}$$

where $\mathbf{W}$ is the stochastic weight matrix of the same dimensions as $\mathbf{U}$. Similar to the iBNN's layer, here we also absorb the stochastic bias vector into $\mathbf{W}$. Letting each pair of corresponding summands in the matrix multiplications of both layers be equal gives us

$$W_{i,j}x_j = U_{i,j}\tilde{x}_j, \tag{3.8}$$

where $W_{i,j}$ and $U_{i,j}$ are the entries at the i-th row and j-th column of matrices $\mathbf{W}$ and $\mathbf{U}$ respectively, and x_j is the j-th element of the input vector $\mathbf{x}$. When $x_j \neq 0$, we can represent the stochastic weight $W_{i,j}$ via the deterministic weight $U_{i,j}$ and the stochastic input feature $\tilde{x}_j$:

$$W_{i,j} = U_{i,j}z_j \tag{3.9}$$
$$z_j = \frac{\tilde{x}_j}{x_j}. \tag{3.10}$$

In the above equations, we define z_j as the stochastic latent variable. Equation (3.9) shows that the product of the latent variable z_j and the deterministic weight $U_{i,j}$ defines the distribution of the stochastic weight $W_{i,j}$. In other

words, the distribution of the stochastic weight matrix $\mathbf{W}$ is:

$$\mathbf{W} = \mathbf{U}(\operatorname{diag} \mathbf{z}) \tag{3.11}$$

$$\mathbf{z} \sim p(\mathbf{z}), \tag{3.12}$$

which is the result of perturbing the input vector $\mathbf{x}$ with the same multiplicative latent vector $\mathbf{z}$:

$$\tilde{\mathbf{x}} = \mathbf{x} \circ \mathbf{z} \tag{3.13}$$

So far, in our derivation, we consider the biases as the last column of the weight matrix. If we explicitly separate out the bias vector and consider the latent vector $\mathbf{z} = \begin{bmatrix} \mathbf{z_U} \\ z_\mathbf{v} \end{bmatrix}$, then we will get the formulation of iBNNs in the previous section.

Equation (3.9) shows that iBNNs share *the same noise distribution between weights interacting with the same input feature* for fully-connected layers.

For a convolution layer, each channel in its input tensor is a feature map whose elements are highly-correlated. It is, therefore, sensible to define a latent variable for each input channel. Specifically, given an iBNN's convolution layer with a deterministic weight tensor $\mathbf{U}$ of dimensions $C_{out} \times C_{in} \times H_F \times W_F$, the latent vector $\mathbf{z}$ of this layer will be of dimension C_{in}. The input tensor $\mathbf{x}$ of this layer is of dimensions $C_{in} \times H_I \times W_I$, and the corresponding perturbed input $\tilde{\mathbf{x}}$ will be a tensor of similar dimensions. Each element $\tilde{x}_{j,m,n}$ of $\tilde{\mathbf{x}}$ is defined as:

$$\tilde{x}_{j,m,n} = x_{j,m,n} z_j \tag{3.14}$$

The output tensor $\mathbf{y}$ of this layer is obtained through performing convolution between the weight tensor $\mathbf{U}$ and the perturbed input tensor $\tilde{\mathbf{x}}$. This is equivalent to performing convolution between a stochastic weight tensor $\mathbf{W}$ and the input tensor $\mathbf{x}$, where $\mathbf{W}$ has the same dimensions as $\mathbf{U}$ and each element $W_{i,j,m,n}$ of $\mathbf{W}$ is defined as:

$$W_{i,j,m,n} = U_{i,j,m,n} z_j \tag{3.15}$$

This equation shows that iBNNs share *the same noise distribution between weights that operate on the same input channel* for convolution layers.

This multiplicative perturbation is intuitive, in that the variance of $p(\tilde{x})$ depends on the magnitude of the corresponding feature x, meaning that an input feature providing a stronger signal will also admit a wider range of sample values.

However, the implicit weight posterior induced by the latent variables is not as expressive as a full weight posterior, leading to an underestimation of predictive uncertainty. To increase the expressiveness of the model, we use a *mixture of Gaussians* as the variational posterior of the latent variables, which we will discuss more in the following sections.

3.3 Variational inference

Given a dataset $\mathcal{D}$, to train an iBNN with L layers, we infer the posterior $p(\mathbf{Z}|\mathcal{D}, \boldsymbol{\theta})$ of the layer-wise latent input variables $\mathbf{Z} = \{\mathbf{z}^{(\ell)}\}_{\ell=1}^{L}$, while treating the weights and biases $\boldsymbol{\theta} = \{(\mathbf{U}^{(\ell)}, \mathbf{v}^{(\ell)})\}_{\ell=1}^{L}$ as deterministic parameters. Let $q_{\boldsymbol{\phi}}(\mathbf{Z})$ denote the variational posterior with the variational parameters $\boldsymbol{\phi}$. We use the negative ELBO as our loss function to be minimised:

$$\mathcal{L}(\boldsymbol{\phi}, \boldsymbol{\theta}) = -\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{Z})}\left[\log p\left(\mathcal{D}|\mathbf{Z}, \boldsymbol{\theta}\right)\right] + \beta \mathrm{KL}\left[q_{\boldsymbol{\phi}}(\mathbf{Z})||p(\mathbf{Z})\right] \tag{3.16}$$

Here, β controls the influence of the KL term in the ELBO. When training iBNNs, we typically set β to 0 at the beginning and slowly increase it to 1 as training progress.

Regarding the deterministic parameters $\boldsymbol{\theta}$, we either jointly optimise them alongside the variational parameters $\boldsymbol{\phi}$ or initialise them using the weights of a pretrained deterministic model.

3.4 Variational ensemble posterior

Leveraging the reduction in the number of random variables, we consider using a *mixture distribution* as the variational posterior for the latent variables $\mathbf{Z}$, where each component in the mixture is a *factorised Gaussian distribution*:

$$q_{\boldsymbol{\phi}}\left(\mathbf{Z}\right) = \frac{1}{K}\sum_{k=1}^{K} q_{\boldsymbol{\phi}^{(k)}}\left(\mathbf{Z}\right) \tag{3.17}$$

$$q_{\boldsymbol{\phi}^{(k)}}\left(\mathbf{Z}\right) = \prod_{\ell=1}^{L} q_{\boldsymbol{\phi}^{(k,\ell)}}\left(\mathbf{z}^{(\ell)}\right) \tag{3.18}$$

$$q_{\boldsymbol{\phi}^{(k,\ell)}}\left(\mathbf{z}^{(\ell)}\right) = \mathcal{N}\left(\mathbf{z}^{(\ell)} \middle| \boldsymbol{\mu}^{(k,\ell)}, \mathrm{diag}\left(\boldsymbol{\sigma}^{(k,\ell)}\right)^2\right) \tag{3.19}$$

where K is the number of mixture components, L is the number of layers, and the variational parameters $\boldsymbol{\phi} = \{(\boldsymbol{\mu}^{(k,\ell)}, \boldsymbol{\sigma}^{(k,\ell)})\}_{k=1,\ell=1}^{K,L}$ are the means and standard deviations. We assume a diagonal Gaussian prior the latent variables $\mathbf{Z}$:

$$p\left(\mathbf{Z}\right) = \prod_{\ell=1}^{L} p\left(\mathbf{z}^{(\ell)}\right) \tag{3.20}$$

$$p\left(\mathbf{z}^{(\ell)}\right) = \mathcal{N}\left(\mathbf{z}^{(\ell)} \middle| \mathbf{m}, \mathrm{diag}\, \mathbf{s}^2\right). \tag{3.21}$$

As this is a prior for multiplicative noise, a good default value for the prior mean $\mathbf{m}$ is $\mathbf{1}$.

We can approximate $\mathrm{KL}\left[q_{\boldsymbol{\phi}}(\mathbf{Z})||p(\mathbf{Z})\right]$ as follows:

$$\mathrm{KL}\left[q_{\boldsymbol{\phi}}(\mathbf{Z})||p(\mathbf{Z})\right] = \mathrm{H}\left(q_{\boldsymbol{\phi}}(\mathbf{Z}), p(\mathbf{Z})\right) - \mathrm{H}\left(q_{\boldsymbol{\phi}}(\mathbf{Z})\right) \tag{3.22}$$

$$\mathrm{H}\left(q_{\boldsymbol{\phi}}(\mathbf{Z}), p(\mathbf{Z})\right) = -\int_{\mathbf{Z}} q_{\boldsymbol{\phi}}(\mathbf{Z}) \log p(\mathbf{Z}) d\mathbf{Z} = -\int_{\mathbf{Z}} \left(\frac{1}{K}\sum_{k=1}^{K} q_{\boldsymbol{\phi}^{(k)}}(\mathbf{Z})\right) \log p(\mathbf{Z}) d\mathbf{Z}$$

$$= \frac{1}{K}\sum_{k=1}^{K} \mathrm{H}\left(q_{\boldsymbol{\phi}^{(k)}}(\mathbf{Z}), p(\mathbf{Z})\right) \tag{3.23}$$

$$\mathrm{H}\left(q_{\boldsymbol{\phi}}(\mathbf{Z})\right) = -\int_{\mathbf{Z}} q_{\boldsymbol{\phi}}(\mathbf{Z}) \log q_{\boldsymbol{\phi}}(\mathbf{Z}) d\mathbf{Z} \approx -\frac{1}{M}\sum_{m=1}^{M} \log q_{\boldsymbol{\phi}}(\mathbf{Z}_m) \tag{3.24}$$

where $\mathrm{H}\left(q_{\boldsymbol{\phi}}(\mathbf{Z}), p(\mathbf{Z})\right)$ is the cross entropy between q and p, and $\mathrm{H}\left(q_{\boldsymbol{\phi}}(\mathbf{Z})\right)$ is the entropy of q. The cross entropy term can be calculated analytically using Equation (3.23). The entropy term can be approximated using a Monte Carlo estimator as in Equation (3.24) where $\{(\mathbf{Z}_m)\}_{m=1}^{M}$ are M samples drawn from $q_{\boldsymbol{\phi}}\left(\mathbf{Z}\right)$.

Instead of using the asymptotic KL in Equation (3.22), we propose to calculate the KL divergence between $\hat{q}\left(\mathbf{Z}\right)$ and $p(\mathbf{Z})$ where

$$\hat{q}\left(\mathbf{Z}\right) = \prod_{\ell=1}^{L} \hat{q}\left(\mathbf{z}^{(\ell)}\right) \tag{3.25}$$

$$\hat{q}\left(\mathbf{z}^{(\ell)}\right) = \mathcal{N}\left(\hat{\boldsymbol{\mu}}^{(\ell)}, \mathrm{diag}\left(\hat{\boldsymbol{\sigma}}^{(\ell)}\right)^2\right) \tag{3.26}$$

$$\hat{\boldsymbol{\mu}}^{(\ell)} = \frac{1}{K}\sum_{k=1}^{K} \boldsymbol{\mu}^{(k,\ell)}, \quad \left(\hat{\boldsymbol{\sigma}}^{(\ell)}\right)^2 = \frac{1}{K^2}\sum_{k=1}^{K} \left(\boldsymbol{\sigma}^{(k,\ell)}\right)^2 \tag{3.27}$$

which is the distribution of $\widehat{\mathbf{Z}} = \frac{1}{K}\sum_{k=1}^{K} \mathbf{Z}^{(k)}$, where $\mathbf{Z}^{(k)}$ is a sample from the k-th component of the original $q_{\boldsymbol{\phi}}(\mathbf{Z})$. This alternative KL term only requires the component means $\boldsymbol{\mu}^{(k,\ell)}$ to *center* around the prior mean $\mathbf{m}$ and the component variances to be close to the prior variance $\mathbf{s}^2$ while placing no penalty on how far each component resides from the prior mean. It is, therefore, a more relaxed constraint compared to the one in Equation (3.22). This allows the posterior components to freely explore the loss landscape surrounding the prior mean. As we will demonstrate later in the next chapter, combining this relaxed KL term with a large learning rate for the variational parameters leads to a substantial improvement in performance of iBNNs compared to the asymptotic KL in Equation (3.22).

3.5 Training algorithm

In this section, we present the training procedure of iBNNs. We use SGD to iteratively update both the deterministic weights $\boldsymbol{\theta}$ and the variational parameters $\boldsymbol{\phi}$ via backpropagation where the loss function in Equation (3.16) is

Algorithm 1 Training procedure

Require: L: number of layers, B: batch size, N: number of epochs, K: number of components of the posterior, S: number of minibatch repetitions, λ_0: learning rate of deterministic parameters $\boldsymbol{\theta}$, λ_1: learning rate of variational parameters $\boldsymbol{\phi}$, $\mathcal{N}(1, \rho^2)$ and $\mathcal{N}(\eta, \tau^2)$: Gaussian initialisers for the mean and standard deviation of each component.

1: Initialise the deterministic parameters $\boldsymbol{\theta}$ using a random initialiser or weights from a pretrained model.
2: Initialise the means $\{\boldsymbol{\mu}^{(k,\ell)}\}_{k=1,\ell=1}^{K,L}$ using $\mathcal{N}(1, \rho^2)$ and initialise the standard deviations $\{\boldsymbol{\sigma}^{(k,\ell)}\}_{k=1,\ell=1}^{K,L}$ using $\mathcal{N}(\eta, \tau^2)$.
3: **for** $i = 1$ **to** N **do**
4: **for all** mini-batches $\mathcal{B}$ of size B from the dataset **do**
5: Repeat each sample in the minibatch S times to form a larger minibatch $\mathcal{B}'$ of size SB.
6: Assign the j-th sample in $\mathcal{B}'$ to the k-th component where $k = j \mod K$.
7: Perform a forward pass using the samples and their assigned components in the minibatch.
8: Calculate the loss $\mathcal{L}$ according to Equation (3.16).
9: Update the deterministic parameters $\boldsymbol{\theta}$ and the variational parameters $\boldsymbol{\phi}$ using SGD:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \lambda_0 \nabla_{\boldsymbol{\theta}} \mathcal{L}$$
$$\boldsymbol{\phi} \leftarrow \boldsymbol{\phi} - \lambda_1 \nabla_{\boldsymbol{\phi}} \mathcal{L}$$

10: **end for**
11: **end for**

evaluated on a minibatch of training samples. However, since the roles of $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ are different, each group of parameters is optimised using a different learning rate. We denote λ_0 as the learning rate for $\boldsymbol{\theta}$ and λ_1 as the learning rate for $\boldsymbol{\phi}$.

Additionally, to encourage functional diversity among the posterior components, they are trained on different permutations of the training set. For each minibatch $\mathcal{B} = \{(x_i, y_i)\}_{i=1}^{B}$ containing B samples, we first replicate each training sample S times to form a larger minibatch of size SB: $\mathcal{B}' = \{(x_i, y_i)_{\times S}\}_{i=1}^{B}$. The j-th sample in $\mathcal{B}'$ is then assigned to the k-th component using the modulo operation:

$$k = j \mod K \tag{3.28}$$

where K is the number of components in the posterior. Under this procedure, each component is trained on an overlapping slice containing SB/K samples of a minibatch, thus encouraging the components to learn different representations of the data. Algorithm 1 presents the full training procedure.

3.6 Connection to other methods

Standard BNNs We present the connection between iBNNs and conventional BNNs in the context of fully-connected NNs, which is highlighted in Equation (3.11). From this equation, if we place a Gaussian posterior $q(\mathbf{z}) = \mathcal{N}(\boldsymbol{\mu}, \operatorname{diag} \boldsymbol{\sigma}^2)$ and a Gaussian prior $p(\mathbf{z}) = \mathcal{N}(\mathbf{1}, \operatorname{diag} \mathbf{s}^2)$ on the latent variables $\mathbf{z}$ of a *layer* of an iBNN, then it will be equivalent to a layer in a standard BNN whose posterior $q(\mathbf{W})$ and prior $p(\mathbf{W})$ of the stochastic weights $\mathbf{W} \in \mathbb{R}^{d_{out} \times d_{in}}$ are:

$$q(\mathbf{W}) = \prod_{j=1}^{d_{in}} q(\mathbf{W}_{:,j}) = \prod_{j=1}^{d_{in}} \mathcal{N}\left(\mathbf{U}_{:,j} \circ \boldsymbol{\mu}, \operatorname{diag}\ (\mathbf{U}_{:,j} \circ \boldsymbol{\sigma})^2\right) \tag{3.29}$$

$$p(\mathbf{W}) = \prod_{j=1}^{d_{in}} p(\mathbf{W}_{:,j}) = \prod_{j=1}^{d_{in}} \mathcal{N}\left(\mathbf{U}_{:,j}, \operatorname{diag}\ (\mathbf{U}_{:,j} \circ \mathbf{s})^2\right) \tag{3.30}$$

where $\mathbf{U} \in \mathbb{R}^{d_{out} \times d_{in}}$ is the deterministic weights of the iBNN, $\mathbf{U}_{:,j}$ and $\mathbf{W}_{:,j}$ denote the j-th column of $\mathbf{U}$ and $\mathbf{W}$. From the equations above, we can view iBNNs as BNNs with shared variational parameters between weights belonging to a column of a layer's weight matrix.

MC-Dropout [13] iBNNs are more general than MC-Dropout because MC-Dropout can be considered as an iBNN with Bernoulli latent node variables.

Multiplicative normalising flows for BNNs [72] Normalising flows [73–76] are transformations that can be applied sequentially to transform a simple distribution to a more complex one. This method also utilises layer-wise multiplicative latent input variables like iBNNs. Unlike iBNNs, however, this method uses normalising flows in the posterior of the latent node variables while still maintaining a fully factorised Gaussian posterior over weights, as its main goal is to improve the flexibility of the factorised Gaussian weight posterior using the latent posterior.

Rank-1 BNNs [49] To see the similarity between Rank-1 BNNs and iBNNs, we first note that from Equation (2.48), one layer of Rank-1 BNNs can be implemented efficiently by multiplying it input nodes and output nodes with the random vectors $\mathbf{s}$ and $\mathbf{r}$ respectively:

$$\mathbf{y} = (\mathbf{U} \circ (\mathbf{r}\mathbf{s}^\top))\mathbf{x} = (\mathbf{U}\,(\mathbf{x} \circ \mathbf{s})) \circ \mathbf{r} \tag{3.31}$$

Thus, a Rank-1 BNN can be viewed as a more general model than an iBNN since it introduces latent variables to both the input and output nodes of each layer. As a result, a Rank-1 BNN has roughly twice the number of variational parameters compared to a similar iBNN. However, we will demonstrate later in Section 5.1 that Rank-1 BNNs have comparable performance to iBNNs, showing that it might not be necessary to use latent variables at both input and output nodes.

Probabilistic meta-representations of NNs [77] This method assigns each node in an NN a latent vector $\mathbf{z}_i^{(\ell)} \in \mathbb{R}^D$ where ℓ is the index of the layer and i is the index of the input node of the ℓ-th layer. If we denote $w_{i,j}^{(\ell)}$ as the weight connecting the i-th node in the ℓ-th layer to the j-th node in the $(\ell+1)$-th layer, then the distribution of $w_{i,j}^{(\ell)}$ is:

$$w_{i,j}^{(\ell)} \sim p\left(w_{i,j}^{(\ell)} \middle| g_{\boldsymbol{\xi}}(\mathbf{z}_i^{(\ell)}, \mathbf{z}_j^{(\ell+1)}, \mathbf{z}_s)\right), \tag{3.32}$$

$$\mathbf{z}_i^{(\ell)} \sim p(\mathbf{z}_i^{(\ell)}), \quad \mathbf{z}_j^{(\ell+1)} \sim p(\mathbf{z}_j^{(\ell+1)}), \quad \mathbf{z}_s \sim p(\mathbf{z}_s) \tag{3.33}$$

where $\mathbf{z}_s \in \mathbb{R}^D$ is a global latent vector shared between all weights and $g_{\boldsymbol{\xi}}$ is a function parameterised by $\boldsymbol{\xi}$. The authors of [77] represented $g_{\boldsymbol{\xi}}$ using an NN which they called the hyper-prior network. By comparing Equation (3.32) with Equation (3.9), one can see that iBNNs are special cases of this method where each weight $w_{i,j}^{(\ell)}$ is represented by the product between its own separate deterministic coefficient $u_{i,j}^{(\ell)} \in R$ and the latent input variable $z_i^{(\ell)} \in \mathbb{R}$.

3.7 Conclusion

In this chapter, we introduced iBNNs, which are efficient and scalable BNNs. Section 3.1 first provided a formal definition of iBNNs, which treat weights as deterministic parameters and induce stochasticity in the outputs via layer-wise multiplicative latent input variables. Next, the motivation behind this formulation of iBNNs was explained in Section 3.2. We then defined the training objective which is a simple negative ELBO in Section 3.3. We discussed the choice of the variational posterior in Section 3.4. Taking advantage of the low dimensionality of the latent variables, we chose a Gaussian mixture posterior for the latent variables and introduced an alternative KL constraint for the loss function. The training procedure of iBNNs, designed to encourage diversity among posterior components, was presented in Section 3.5. We concluded the chapter by briefly comparing iBNNs to other methods in Section 3.6.

Chapter 4

Ablation studies

In this chapter, we study the properties of iBNNs. We first present the experiment settings in Section 4.1, which will also be used for the main experiments in the next chapter. Section 4.2 visualises the predictive uncertainty of iBNNs on a 1D regression task. In Section 4.3, we empirically compare the two KL constraints presented in Section 3.4, which shows the advantage of the alternative KL term that we proposed. Section 4.4 and Section 4.5 study the effect of the variational learning rate λ_1 and the number of data replications S, which are two hyperparameters crucial to performance of iBNNs. Section 4.6 visualises the regions in the loss landscapes captured by the posteriors of iBNNs, and Section 4.7 concludes this chapter.

4.1 Experiment settings

In this section, we will present the experiment protocol used in the rest of this thesis, namely the NN architectures, the datasets and the performance metrics.

Architectures We use VGG-16 [78] and WRN-28x10 [79] in our experiments. These two architectures are depicted in Figure 4.1.

Dataset We use CIFAR dataset [80] in our experiments. This dataset contains 60000 colour images of size 32×32, where 50000 images are used for training and 10000 images are used for testing. This dataset has two variants, CIFAR-10 and CIFAR-100, which differ from each other in the number of classes. CIFAR-10 has 10 mutually exclusive classes, each containing 5000 training images and 1000 testing images. CIFAR-100 has 100 classes, each containing 500 training images and 100 testing images. Based on these statistics, CIFAR is a balanced dataset ideal for benchmarking.

Evaluation metrics We use prediction error rate, negative log-likelihood (NLL), and expected calibration error (ECE). They are three important metrics for measuring a model's generalisation capability, and they are calculated on a

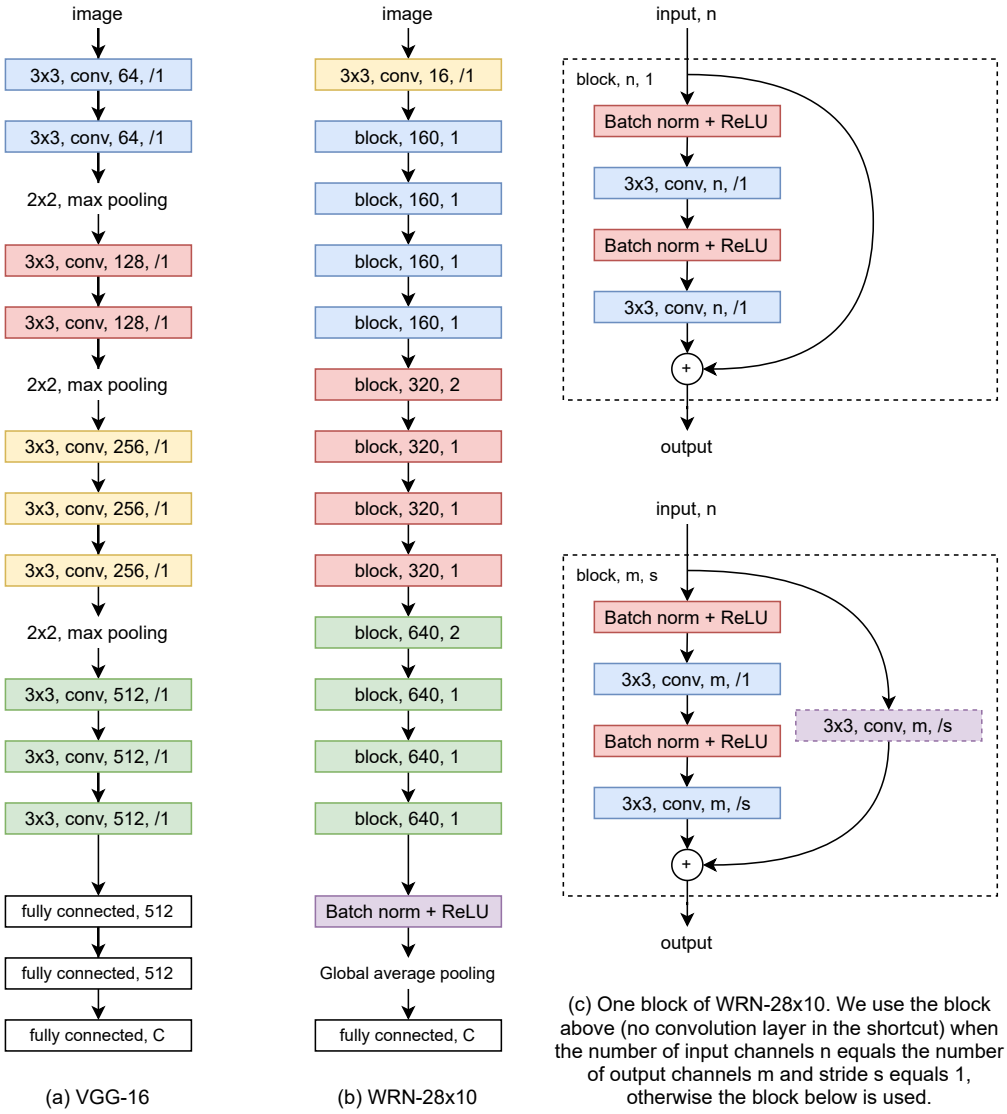

Figure 4.1: The architectures of VGG-16 (a) and WRN-28x10 (b, c) with C target classes. $\boldsymbol{k \times k}$**, conv,** $\boldsymbol{m}$**,** $\boldsymbol{/s}$ denotes a convolution layer with kernel size k, m output channels and stride s. All convolution layers of both networks use zero padding of size one. For VGG-16, ReLU activation is used after every convolution and fully connected layers. One block of WRN-28x10 is depicted in (c) where n is the number of input channels, m is the number of output channels, and s is the stride.

test dataset $\mathcal{D}' = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$. We calculate these metrics using the *posterior predictive distribution* $p(\mathbf{y}|\mathbf{x}, \mathcal{D}, \mathcal{M})$ defined in Equation (2.5). For all three metrics, lower is better. We will explain each metric below:

- Prediction error rate is simply the percentage of test samples that are wrongly predicted, which indicates the accuracy of the model's predictions. Let C be the set of class indices. Then a model's prediction $\hat{y}_i$ given an input $\mathbf{x}_i$ is:

$$\hat{y}_i = \underset{c \in C}{\operatorname{argmax}}\, p(y = c|\mathbf{x}_i, \mathcal{D}, \mathcal{M}) \tag{4.1}$$

  Thus the prediction error of a model on the test data $\mathcal{D}'$ is:

$$\text{Error} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}(\hat{y}_i \neq y_i) \tag{4.2}$$

  where $\mathbb{1}(.)$ equals 1 if the condition in parentheses is correct and equals 0 otherwise.

- NLL is the negative log probability that a model assigns to the test dataset $\mathcal{D}'$. Assuming that $\mathcal{D}'$ contains only i.i.d. samples, NLL is calculated by summing the negative log probabilities of all the samples in $\mathcal{D}'$:

$$\text{NLL} = -\sum_{i=1}^{N} \log p(y_i|\mathbf{x}_i, \mathcal{D}, \mathcal{M}) \tag{4.3}$$

  In this thesis, however, we calculate NLL by averaging instead of summing so that the final value does not depend on the size of the test dataset.

- ECE [81] measures how well-calibrated is the confidence $\hat{p}_i$ of a model's prediction $\hat{y}_i$ given an input $\mathbf{x}_i$. A well-calibrated prediction means that its confidence reflects its true correctness likelihood [10]. This metric is calculated by first dividing all the model's predictions $\{(\hat{y}_i, \hat{p}_i)\}_{i=1}^{N}$ on the test set $\mathcal{D}'$ into M equally-spaced bins on the $[0, 1]$ interval based on the confidences $\hat{p}_i$. Let B_m denote the set of indices of all predictions in the m-th bin. Then ECE is calculated as follows:

$$\text{ECE} = \sum_{m=1}^{M} \frac{|B_m|}{N} \big|\text{acc}(B_m) - \text{conf}(B_m)\big| \tag{4.4}$$

  where $\text{acc}(B_m)$ and $\text{conf}(B_m)$ are:

$$\text{acc}(B_m) = \frac{1}{|B_m|}\sum_{i \in B_m} \mathbb{1}(\hat{y}_i = y_i), \tag{4.5}$$

$$\text{conf}(B_m) = \frac{1}{|B_m|}\sum_{i \in B_m} \hat{p}_i \tag{4.6}$$

Experiment protocol In this chapter, we will mainly use VGG-16 with CIFAR-100 to study the behaviour of iBNNs. We choose VGG-16 because it is smaller than WRN-28x10, thus allowing us to perform multiple experiments under different hyperparameters in a shorter time, and CIFAR-100 is chosen because it is challenging enough to show the differences in performance under various settings. For testing, given an iBNN with K posterior components, we draw $32/K$ samples from each component to approximate the posterior predictive distribution. We use 5000 samples from the training data of CIFAR-100 as validation.

4.2 Visualising predictive uncertainty

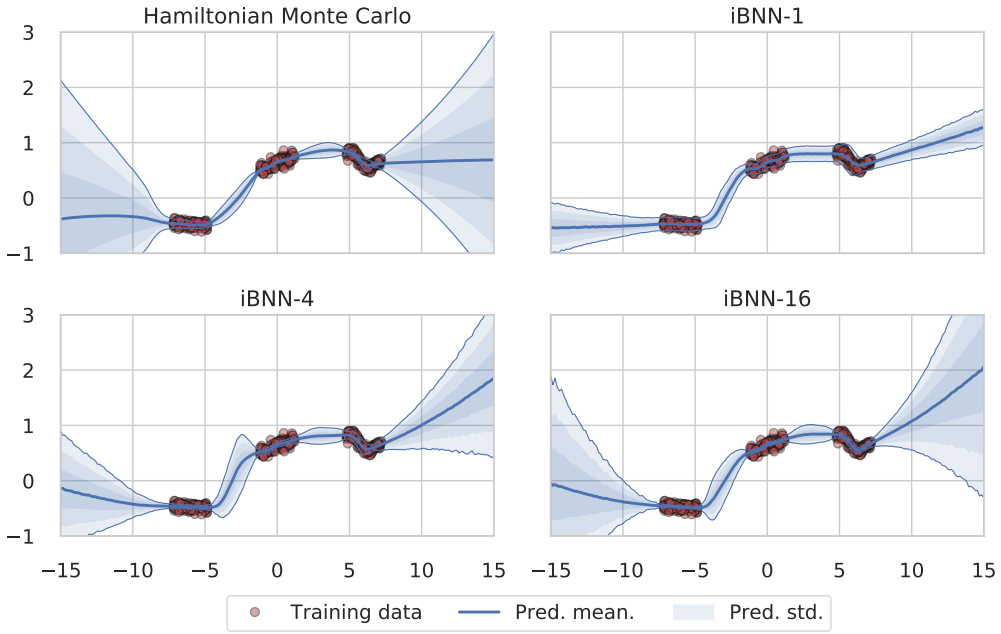

Figure 4.2: Behaviour of iBNNs on a 1D regression toy dataset as the number of posterior components K increases, compared to the behaviour of a BNN trained by HMC. iBNN-K denotes an iBNN with K components.

We first observe the predictive uncertainty of iBNNs on a 1D homoscedastic regression task. We use the toy dataset of [82] with 400 samples, and a feedforward network with 4 layers of sizes $(200, 50, 50, 50)$ and ReLU activation. For comparison, we train a standard BNN using HMC. We use 1000 samples for BNN-HMC. The results are visualised in Figure 4.2. This figure shows that the predictive uncertainty of iBNNs is well-behaved, since it increases as we move away from the training samples. The uncertainty is higher for iBNNs with more posterior components, with iBNN-16 produces comparable uncertainty to that of BNN-HMC. We note that iBNN-16 has roughly the same number of parameters as a single deterministic NN, while BNN-HMC in this experiment has to store 1000 weight samples.

4.3 Comparing KL approximation

We compare between two variants of KL divergence discussed in Section 3.4: the first one called *KL exact* is the asymptotic KL approximation defined in Equation (3.22), and the second one called *KL mean* is the alternative KL between $\hat{q}$ and the prior p where $\hat{q}$ is defined in Equation (3.26). For this experiment, we use VGG-16 and CIFAR-100. We set $K = 8$ and $S = 4$ where K is the number of posterior components and S is the number of data replications. The training data contains 45000/50000 samples from the original training set, and the validation set contains the remaining 5000 samples. The results are shown in Figure 4.3. Overall, using the KL mean allows the model to reach much better performance than KL exact across all three metrics and under different values of the variational learning rate λ_1. These results support our argument in Section 3.4 that allowing the components more freedom to move around the parameter space improves performance of iBNNs. Thus, we will use *KL mean* in all of our subsequent experiments.

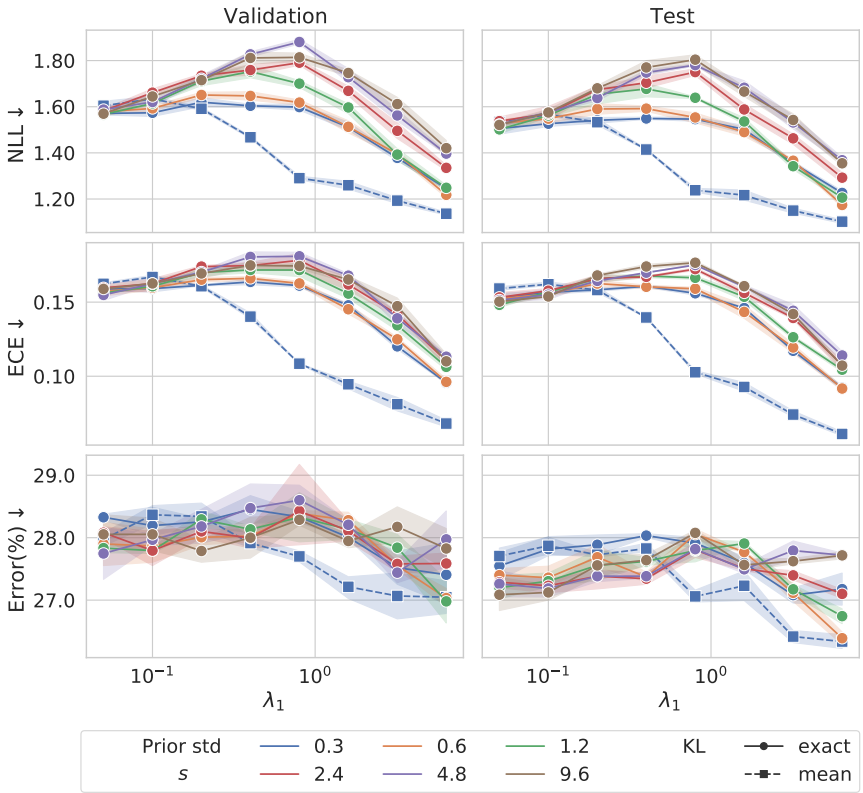

Figure 4.3: Comparison between two KL variants. For KL exact, we run experiments for all values of the prior standard deviation s in $\{0.3, 0.6, 1.2, 2.4, 4.8, 9.6\}$. For KL mean, we fix the value of s at 0.3. We track the model's performance under both KL terms for all values of λ_1 in $\{0.05, 0.10, 0.20, 0.40, 0.80, 1.60, 3.20, 6.40\}$, where λ_1 is the learning rate of the variational parameters ϕ.

4.4 Effects of the variational learning rate λ_1

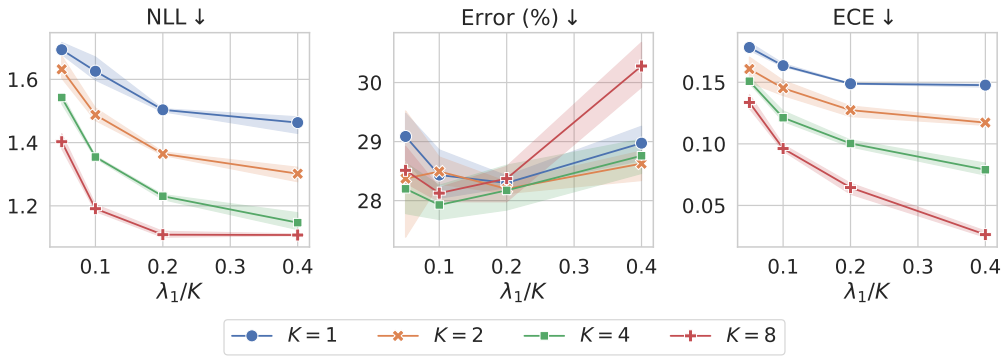

Figure 4.4: Performance of iBNNs with different number of components K under VGG-16/CIFAR-100. We use a batch size of 128 and $S = 1$ in this experiment.

We hypothesise that using a larger learning rate λ_1 for the variational parameters than the learning rate λ_0 for the deterministic parameters can be beneficial to the model's performance. Since our KL constraint allows the posterior components to be far away from the mean, a large λ_1 improves diversity between components by encouraging them to explore a wide region in the parameter space around the prior mean. Furthermore, due to the training procedure in Section 3.5, each component is trained on a minibatch slice of size $SB/K \leq B$, as we typically use $S \leq K$. The update step of each component after one iteration, therefore, is less accurate than that of the deterministic weights, whose gradient is calculated on the entire minibatch. Setting λ_1 to a large value relatively to the value of λ_0 thus allows these components to quickly adapt to the changes of the deterministic weights.

We empirically confirm our hypothesis by training iBNNs with VGG-16 on CIFAR-100. Similar to the previous section, we use 45000/50000 samples from the original training set as training data, and we plot the results on the validation set containing the remaining 5000 samples in Figure 4.4 as λ_1 increases. We plot λ_1/K on the x-axis since we scale λ_1 based on the number of components K. Figure 4.4 shows that both NLL and ECE decrease as λ_1 increases, and we need to use larger λ_1 for larger K. The increase in validation error is because we use a batch size of 128 and $S = 1$ in this experiment, causing the size of the minibatch slice for each component to quickly decrease as K increases, thus preventing the variational parameters to converge to high-performing values. This problem can be alleviated by increasing the number of data repetitions S, as shown in the next section.

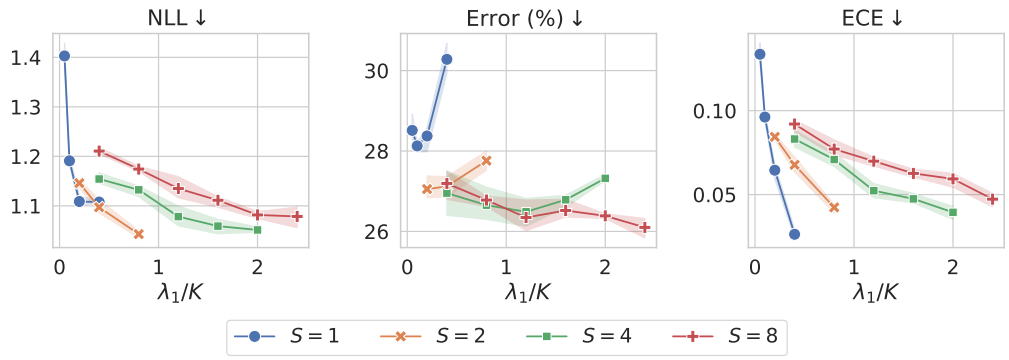

Figure 4.5: Performance of iBNNs with $K = 8$ components under VGG-16/CIFAR-100 with different number of data repetitions S.

4.5 Effects of the number of data replications S

We study the importance of the number of data replications S to performance of iBNNs. We use a similar setting as in the previous section with $K = 8$ components. Figure 4.5 shows the results on the validation set under different values of S as λ_1 increases. Larger S permits the usage of larger λ_1 without increasing classification error since each component can be trained on a larger minibatch slice. From the input augmentation perspective, using $S > 1$ means that S out of K components are chosen during the forward pass, producing S different perturbed versions of the input of each layer. The deterministic weights of each layer, therefore, are trained to accommodate different input perturbations, thus improving robustness and generalisation. Nonetheless, we only want to set S to a small and appropriate value, as the training complexity scales linearly with respect to S. Figure 4.5 suggests that $S = 4$ is a good value.

4.6 Visualising the loss landscape

In this section, we visualise the regions on the loss landscapes captured by the posteriors of iBNNs. We first observe that, from Equation (3.9) and Equation (3.15), we can represent each component in the posterior of an iBNN with its mean in the weight space $\boldsymbol{\theta}^{(k)}$:

$$\boldsymbol{\theta}^{(k)} = \left[\mathrm{vec}(\mathbf{W}^{(k,1)}) \quad \mathrm{vec}(\mathbf{W}^{(k,2)}) \quad \cdots \quad \mathrm{vec}(\mathbf{W}^{(k,L)})\right]^\top \tag{4.7}$$

where k is the index of the component, L is the number of layers, vec denotes the operation that reshapes a matrix or tensor into a 1D row vector, and $\mathbf{W}^{(k,\ell)}$ is defined as follows:

- If the ℓ-th layer is a fully-connected layer, then based on Equation (3.9), we have:

$$W_{i,j}^{(k,\ell)} = U_{i,j}^{(\ell)} \mu_j^{(k,\ell)} \tag{4.8}$$

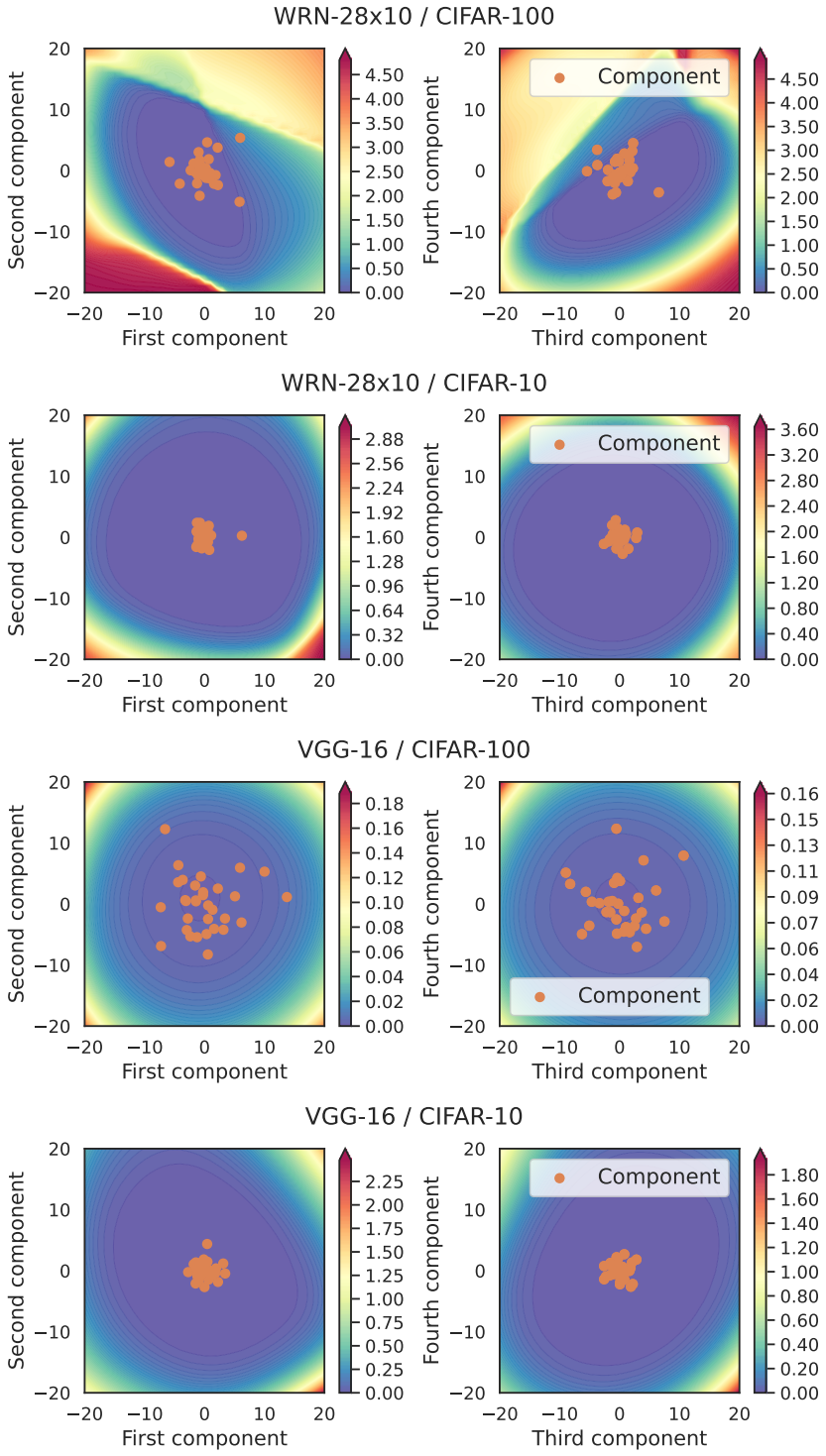

Figure 4.6: The loss regions captured by iBNNs under different pairs of architecture-dataset in the first and second components (left) and in the third and fourth components (right).

- If the ℓ-th layer is a convolution layer, then based on Equation (3.15), we have:

$$W_{i,j,m,n}^{(k,\ell)} = U_{i,j,m,n}^{(\ell)} \mu_j^{(k,\ell)} \tag{4.9}$$

where $\mathbf{U}^{(\ell)}$ contains the deterministic weights, and $\boldsymbol{\mu}^{(k,\ell)}$ is the mean vector of the k-th component in the latent posterior of the ℓ-th layer. For fully-connected layers, $\mathbf{U}^{(\ell)}$ and $\mathbf{W}^{(k,\ell)}$ are 2D weight matrices, and for convolution layers, $\mathbf{U}^{(\ell)}$ and $\mathbf{W}^{(k,\ell)}$ are 4D weight tensors.

From K weight values $\{\boldsymbol{\theta}^{(k)}\}_{k=1}^{K}$, we use *principal component analysis* (PCA) to get the first four principle components $\{\mathbf{v}_i\}_{i=1}^{4}$ of the weight space. We then visualise the loss landscape in the first two components and in the next two components for each pair of architecture-dataset in Figure 4.6.

To generate each contour plot in Figure 4.6, we first define a 2D grid centering around 0. Then for each coordinate (x, y) in the 2D grid, we calculate the weight value $\boldsymbol{\theta}(x, y)$ and use that weight value to calculate the NLL of the training data. Each weight value $\boldsymbol{\theta}(x, y)$ is defined as:

$$\boldsymbol{\theta}(x, y) := \overline{\boldsymbol{\theta}} + x \frac{\mathbf{v}_x}{||\mathbf{v}_x||_2} + y \frac{\mathbf{v}_y}{||\mathbf{v}_y||_2} \tag{4.10}$$

$$\overline{\boldsymbol{\theta}} := \frac{1}{K} \sum_{k=1}^{K} \boldsymbol{\theta}^{(k)} \tag{4.11}$$

where $\mathbf{v}_x$ and $\mathbf{v}_y$ are the principle components corresponding to the x-axis and y-axis respectively. Figure 4.6 shows that the components of an iBNN populate a local mode in the loss landscape, thus putting iBNNs into the category of methods providing single-mode approximations such as SWAG and Radial BNNs.

4.7 Conclusion

This section provided an in-depth study of the behaviours and characteristics of iBNNs. We first visualised the predictive uncertainty of iBNNs in a 1D toy regression problem in Section 4.2, showing that iBNNs with a sufficient number of posterior components produce well-behaved uncertainty estimates comparable to those of standard BNNs. Through the experiment in Section 4.3, we found that using a relaxed KL term allows iBNNs to reach better generalisation performance. Section 4.4 showed that it is importance to set the variational learning rate λ_1 to a large value. Section 4.5 showed that replicating each mini-batch S times during training is required to maintain good performance and suggested that $S = 4$ is a good value. Finally, Section 4.6 utilised PCA to show that an iBNN approximates a local mode in the loss landscape.

Chapter 5

Experiments

In this chapter, we evaluate iBNNs in two aspects: the first one is their generalisation performance on clean test data, and the second one is their robustness against OOD input samples. We use standard benchmark datasets and modern deep NN architectures for image classification in our experiments.

To achieve our first goal, in Section 5.1, we compare iBNNs to deterministic NNs, DEs, and other scalable BNN approaches presented in Section 2.5 using clean testing samples, which are i.i.d. to the training samples, to observe how well iBNNs perform under ideal test settings.

For our second goal, in Section 5.2.1, we artificially introduce noise to the clean test data and visualise the changes in predictive uncertainty of iBNNs as the noise intensity increases. We then test iBNNs under two different scenarios of distributional shift. The first one is *covariate shift* [8], where the target labels of the test samples are similar to the training data but the inputs come from a different distribution. To simulate this scenario, in Section 5.2.2, we use the test dataset of common image corruptions introduced in [83]. The second scenario is *completely OOD*, where both the inputs and the ground truth labels come from an entirely different dataset. We evaluate the capability of iBNNs to detect OOD samples in Section 5.2.3 using the testing procedure in [84].

We demonstrate in Section 5.3 that an iBNN can reuse weights from a pre-trained deterministic model and achieve better predictive performance for a few training epochs.

Section 5.4 concludes this chapter.

5.1 Experiments on CIFAR

5.1.1 Experimental setup

Dataset We use CIFAR dataset presented in Section 4.1.

Evaluation metrics We calculate prediction error rate, NLL, and ECE of each method for comparison. We have discussed these metrics in Section 4.1.

Baselines Our baselines include Radial BNNs [30], MC-Dropout [13], SWAG [48], DEs [14] and Rank-1 BNNs [49], which we have discussed in Section 2.5. We also include the results of deterministic models in our comparison.

Evaluation protocol We run each experiment 5 times using different random seeds and report the mean and standard deviation of each metric (except for DEs, which we only report the mean for each metric). To ensure a fair comparison between BNN methods, we give each of them an equal sampling budget during testing. We draw 32 weight samples during testing and average the predictions from these samples for Radial BNNs, SWAG and MC-Dropout; while for iBNNs with K posterior components, we draw $32/K$ samples from each component for averaging. For a comprehensive evaluation, we report the results for iBNNs with 1, 2, 4, 8, 16 and 32 component(s) in the posterior.

Hyperparameters We include all information about the hyperparameters and implementations of each method in the appendix.

5.1.2 Results

Figure 5.1 shows the results of iBNNs and baseline methods on CIFAR dataset. Overall, iBNNs greatly benefit from having multiple components in their posteriors. We see improvement across all metrics and under different settings as the number of components increases. We do not consider using $K > 32$, as our initial experiments show diminishing returns in performance gain. This is in accordance with our observation in Section 4.6 that the components of an iBNN populate around a local minimum of the loss landscape which puts an upper bound on the diversity among the predictions from these components. In other words, as we add more components to the posterior, the likelihood of a new component being similar to one of the already existing components increases, leading to diminishing improvements. This phenomenon is also observed for SWAG, which uses a multivariate Gaussian posterior to approximate a local mode, as its performance does not improve when we use more than 30 weight samples during inference.

For CIFAR-10, iBNNs are consistently better than SWAG and provide competitive performance compared to DEs. For CIFAR-100, iBNNs are better than SWAG in terms of accuracy but worse than SWAG in terms of NLL. iBNNs achieve better ECE than SWAG in most cases, with the only exception being VGG-16/CIFAR-100. In all cases, DEs achieve better accuracy and NLL while being less calibrated than iBNNs. While Radial BNNs solve the convergence issue of VI, they still return subpar performance, as their fully factorised posterior does not capture the correlations between weights. MC-Dropout does not perform as well as iBNNs and SWAG because it combines predictions from smaller models sharing the same computational graph.

Despite having competitive performance compared to SWAG and DEs, iBNNs have the fewest parameters. Each component in the posterior of an iBNN only requires a relatively small number of variational parameters,

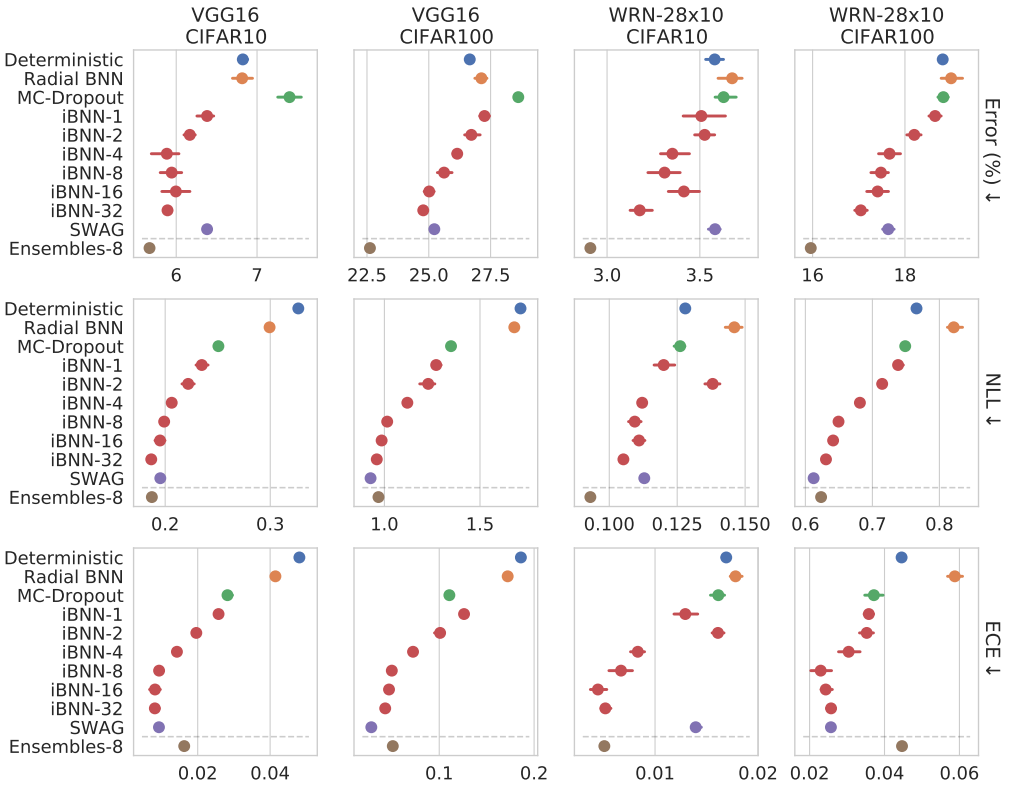

Figure 5.1: Results for iBNNs and baseline methods on CIFAR dataset. Each setting is run 5 times with different random seeds (except for DEs). Lower is better in all quantities. iBNN-K denotes models with K posterior components. We use 8 models for DEs.

whereas a DE with 8 models contains 8 times the number of parameters of a single deterministic model, and a SWAG model needs to store 20 copy of deterministic weights to approximate the multivariate Gaussian posterior. For example, on WRN-28x10 with a total of 9475 input nodes, each component of an iBNN only adds $2 \times 9475 = 18950$ variational parameters to the model. We visualise the number of parameters of all methods in Figure 5.2, showing that iBNNs have significantly lower memory footprint during inference than other methods, except for MC-Dropout. This advantage allows iBNNs to efficiently scale up to large NNs.

Comparison to Rank-1 BNNs As discussed in Section 3.6, Rank-1 BNNs are quite similar to iBNNs. However, the KL constraint in the loss function of Rank-1 BNNs is different from that of iBNNs. For a Rank-1 BNN with K posterior components, let $\mathbf{Z} = \{(\mathbf{s}^{(\ell)}, \mathbf{r}^{(\ell)})\}_{\ell=1}^{L}$ denote all the latent variables in the model, where $\mathbf{s}^{(\ell)}$ and $\mathbf{r}^{(\ell)}$ are the latent input vector and the latent output vector of the

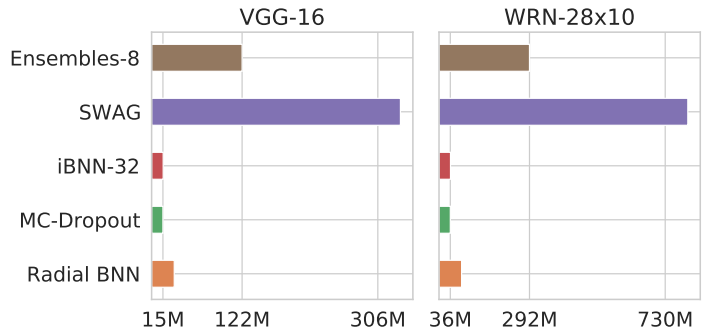

Figure 5.2: Number of parameters of different methods

ℓ-th layer, respectively. The KL divergence is then defined as:

$$\begin{aligned}\mathrm{KL}\left[q(\mathbf{Z})||p(\mathbf{Z})\right] &:= \sum_{k=1}^{K} \mathrm{KL}\left[q_k(\mathbf{Z})||p(\mathbf{Z})\right] \\ &= \sum_{k=1}^{K}\sum_{\ell=1}^{L} \mathrm{KL}\left[q_k(\mathbf{s}^{(\ell)})||p(\mathbf{s}^{(\ell)})\right] + \sum_{k=1}^{K}\sum_{\ell=1}^{L} \mathrm{KL}\left[q_k(\mathbf{r}^{(\ell)})||p(\mathbf{r}^{(\ell)})\right] \quad (5.1)\end{aligned}$$

where q_k is the k-th component of the posterior. As this constraint controls the distance between each component mean and the prior mean, the posterior components are not free to explore regions far away from the prior in the loss landscape. We thus hypothesise that it could be too restrictive and could reduce the diversity among the predictions returned by different components, thereby reducing the model's performance. To test our hypothesis, we train Rank-1 BNNs under different settings and compare their test results to iBNNs, which employ a less restrictive KL constraint in the loss function. Figure 5.3 shows that while Rank-1 BNNs are slightly more calibrated, iBNNs are able to achieve better accuracy and NLL, thus providing empirical evidence supporting our hypothesis. This also suggests that we might not need to include the latent variables for both input and output nodes since the latent input variables are enough to achieve good performance.

5.2 Experiments in OOD settings

5.2.1 Visualising predictive uncertainty

In this section, we artificially introduce noise to the test samples of CIFAR to observe the changes in the aleatoric and epistemic uncertainty of the predictions

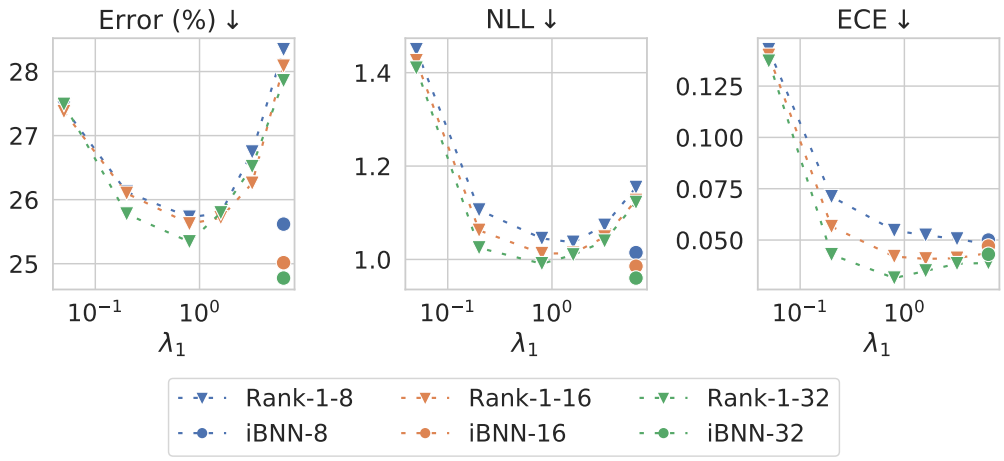

Figure 5.3: Results for Rank-1-K ($\triangledown$ marks) with K components as the variational learning rate λ_1 increases. For comparison, we plot the results for iBNN-K ($\circ$ marks) in the value of λ_1 corresponding to its best performance. We use VGG-16/CIFAR-100 for this experiment. Each result is averaged over 5 runs.

from an iBNN as the noise intensity increases. We consider two types of noise:

$$\textit{Gaussian noise:} \qquad x' := (1-p)x + p\epsilon \qquad \epsilon \sim \mathcal{N}(0,1) \tag{5.2}$$

$$\textit{Salt and pepper noise:} \qquad x' := \begin{cases} 1, \text{ if } \epsilon > 1 - p/2 \\ 0, \text{ if } \epsilon < p/2 \\ x, \text{ otherwise} \end{cases} \qquad \epsilon \sim \mathcal{U}(0,1) \tag{5.3}$$

where $p \in [0,1]$ controls the noise intensity, x' is an element in the corrupted sample $\mathbf{x}'$ with x as the corresponding element in the original sample $\mathbf{x}$, and $\mathcal{U}(0,1)$ denotes the uniform distribution between 0 and 1.

We use the equations in Section 2.3 to calculate the epistemic and aleatoric uncertainty for each test sample, and visualise the mean and standard deviation of both types of uncertainty as a function of noise intensity p in Figure 5.4a and Figure 5.4b. These figures show that an iBNN increases its uncertainty as inputs converge towards noise, which is an ideal behaviour. We also observe that epistemic uncertainty is higher for models with more posterior components, which is intuitive since adding more components will increase diversity among the predictions from different components. Overall, we can conclude that iBNNs produce reliable predictive uncertainty.

5.2.2 Experiments on CIFAR-C

In this section, we examine performance of iBNNs under covariate shift using the CIFAR-C dataset introduced in [83]. This dataset contains the test images from the original CIFAR dataset, but these images have been corrupted using 19 common corruption types with 5 levels of intensity. Since we still have the true

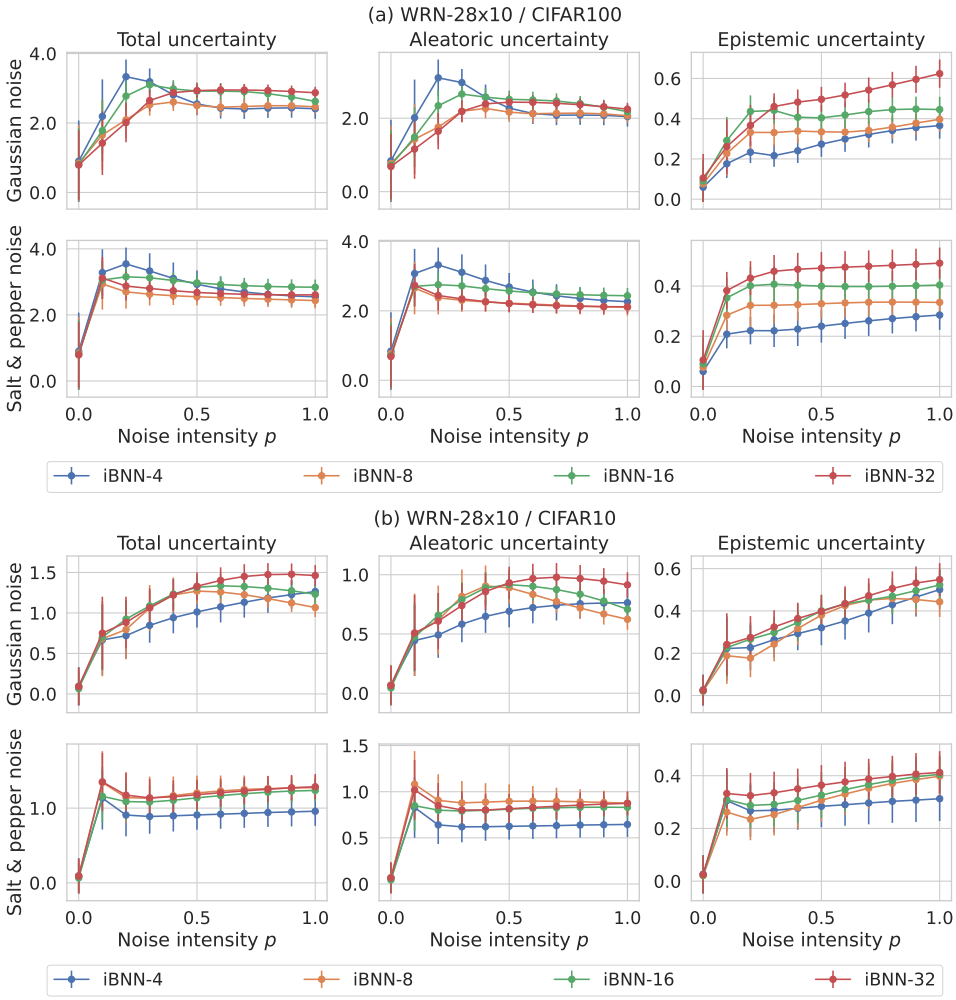

Figure 5.4: The progression of aleatoric and epistemic uncertainty represented by the average entropy over all test samples as the noise intensity increases.

target labels in this scenario, we use prediction error, NLL and ECE as evaluation metrics to measure how well iBNNs generalise to corrupted test samples. For baselines, we use SWAG, MC-Dropout and DEs. For each method, we report the result of each intensity level averaged over all corruption types in Figure 5.5 and Figure 5.6.

Both Figure 5.5 and Figure 5.6 show that all methods perform worse as the corruption intensity increases. Figure 5.5 shows that with WRN-28x10, iBNNs return the lowest prediction errors in both CIFAR-10-C and CIFAR-100-C; while for NLL, iBNNs are better than both SWAG and DEs in CIFAR-10-C, and are only worse than DEs in CIFAR-100-C. However, both SWAG and DEs either perform similar to or better than iBNNs in terms of ECE. Figure 5.6 shows that with VGG-16, iBNNs return the second best results in terms of prediction error and NLL, while the best results belong to DEs. Regarding ECE, iBNNs provide the best results in CIFAR-10-C and are only worse than SWAG in CIFAR-100-

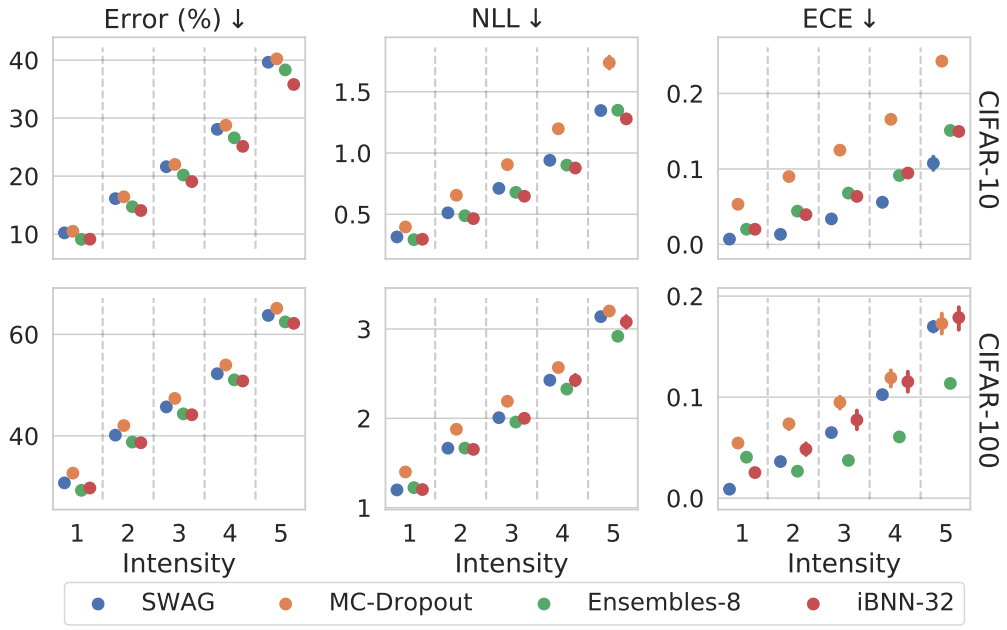

Figure 5.5: Results for iBNN-32, MC-Dropout, SWAG, and DEs with WRN-28x10/CIFAR-C. Each experiment is run 5 times.

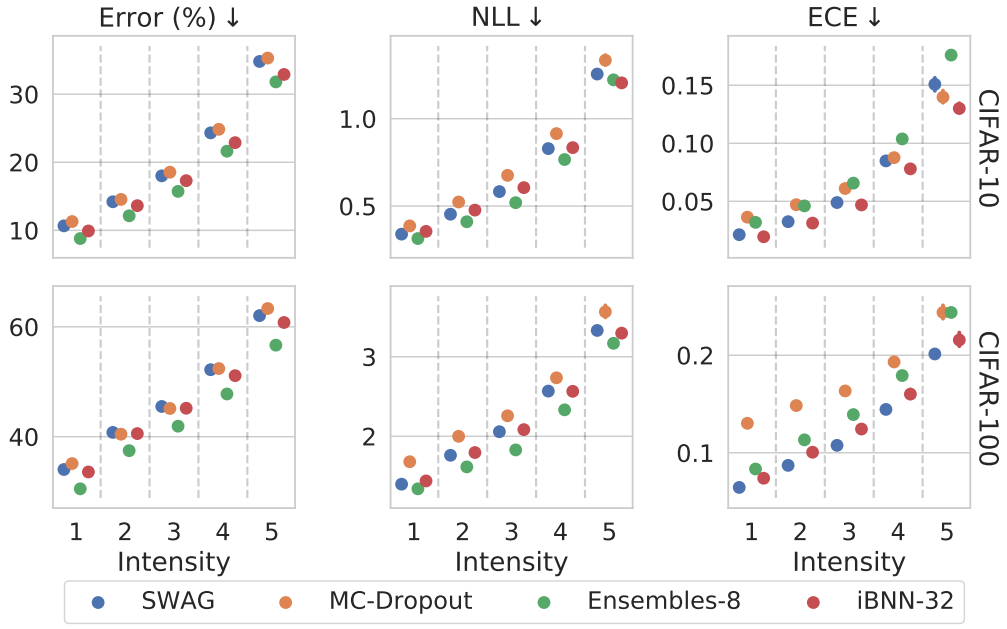

Figure 5.6: Results for iBNN-32, MC-Dropout, SWAG, and DEs with VGG-16/CIFAR-C. Each experiment is run 5 times.

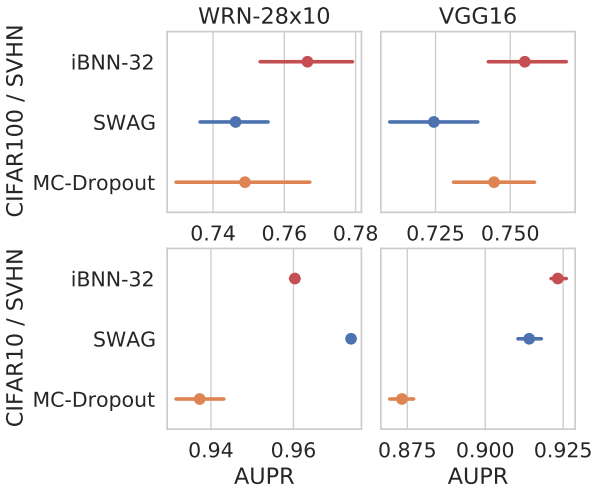

Figure 5.7: Results for iBNN-32, SWAG and MC-Dropout models trained on CIFAR in the OOD detection task with SVHN as the OOD dataset, averaged over 5 runs.

C. Overall, these results show that iBNNs generalise well to perturbed input samples.

5.2.3 OOD detection

In this section, we further evaluate the predictive uncertainty of iBNNs using the OOD detection task. Following the testing procedure in [84], we consider this task as a binary classification problem where we assign the positive label to all samples in the original test dataset and the negative label to all test samples from a completely different dataset. The confident score of a prediction is the maximum probability in the posterior predictive distribution. Since we train our model on CIFAR which is a dataset of 32×32 colour images, we use the test set of SVHN [85] as our OOD samples, because these two datasets contain inputs of the same dimensions while having completely different sets of target labels. This is an imbalanced binary classification task because CIFAR contains 10000 test images whereas SVHN contains 26032 test images. Therefore, we choose Area Under the Precision-Recall curve (AUPR) [86] as the evaluation metric due to it being a threshold-independent metric in binary classification suitable for imbalanced test data. A higher AUPR indicates a better classifier, as the Precision-Recall curve plots precision against recall as the confidence threshold increases from 0 to 1. Figure 5.7 shows that iBNN-32 is better than both SWAG and MC-Dropout in most scenarios and only performs worse than SWAG in the WRN-28x10/CIFAR-10 setting.

Table 5.1: Results for ResNet-50 on the validation set of IMAGENET using pretrained parameters from `torchvision.models`. Each experiment is run once.

	Error (%) (↓)	NLL (↓)	ECE (↓)
Deterministic	23.99	0.9629	0.0378
iBNN-4	23.18	0.9265	0.0300
iBNN-8	23.04	0.9182	0.0249
iBNN-16	23.05	0.9111	0.0191
iBNN-32	**22.91**	**0.9048**	**0.0163**

5.3 Improving pretrained IMAGENET models

In this section, we demonstrate the ability of iBNNs to use pretrained deterministic weights to save training time and improve performance of pretrained models. Despite saving memory during inference, iBNNs require a longer training time due to the need of replicating each minibatch S times during training in order to achieve good performance, as we have discussed in Chapter 4. For example, on one Tesla-V100 GPU, training a WRN-28x10 model on CIFAR using SWAG takes about 9 hours, which is similar to the training time of a single deterministic model, while training an iBNN with $S = 4$ will take around 34 hours. To reduce the training time of iBNNs, especially in the context of very large models and datasets, we propose instead to initialise the deterministic weights $\boldsymbol{\theta}$ of an iBNN with weights from a corresponding pretrained deterministic model and only train the variational parameters $\boldsymbol{\phi}$ for a few more epochs. This subsequent training procedure thus takes much less time than fully training an iBNN from the beginning. In this way, an iBNN improves performance of the pretrained model by letting the components in the ensemble posterior explore the local neighbourhood of the deterministic weights.

For the experiments in this section, we use the ResNet-50 [4] architecture and IMAGENET dataset [87], which is a large scale dataset for image classification. This dataset contains 1200000 training images, 50000 validation images and 100000 testing images. We only report the results on the validation set because the labels from the testing set are not published. The images in the dataset are of various sizes and are divided into 1000 classes. The number of images per class in the training data ranges from 730 to 1300 images, while the validation set contains 50 images per class. We use the standard data augmentation methods for training IMAGENET models available on the `Pytorch` repository [1]. For evaluation, each input image is first resized to 256×256, and then cropped to a region of 224×224 at the center of the image. We initialise the deterministic weights $\boldsymbol{\theta}$ of an iBNN using the pretrained weights of ResNet-50 published in the `torchvision.models` package, and train the variational parameters $\boldsymbol{\phi}$ for 15 epochs. For evaluation, we draw $32/K$ samples from each component of iBNN-K for averaging. We report the classification error, NLL and ECE in

[1] https://github.com/pytorch/examples/blob/master/imagenet/main.py

Table 5.1, showing that iBNNs improve performance of the pretrained ResNet-50 across all metrics. This result is important as we can convert state-of-the-art NNs, which have been growing quickly in size due to the abundance of training data and computational resources, to iBNNs with minimal overhead for predictive uncertainty estimation.

5.4 Conclusion

In this chapter, we thoroughly evaluated generalisation performance and predictive uncertainty of iBNNs using architectures and datasets in image classification. We first compared iBNNs against other scalable BNN methods under ideal test settings in Section 5.1. We then studied the behaviour of iBNNs on OOD inputs, where we first visualised the progression of aleatoric and epistemic uncertainty as the test samples converge towards noise in Section 5.2.1. Next, we evaluated iBNNs under covariate shift in Section 5.2.2, showing that iBNNs can generalise well to perturbed inputs. Section 5.2.3 presented the competitive results of iBNNs in the OOD detection task. Finally, in Section 5.3, we empirically showed that iBNNs can reuse weights from pretrained deterministic models to achieve good performance with reduced training time.

Chapter 6

Discussion

6.1 Advantages of iBNNs

In this section, we present the advantages of iBNNs. We attribute these advantages to the combination of (i) node parameterisation, (ii) multi-modal latent posterior, (iii) a suitable loss function with a relaxed KL constraint, and (iv) an intuitive training algorithm.

Good generalisation performance The experimental results in Chapter 5 show that iBNNs perform well under various settings. This achievement is due to iBNNs addressing three significant problems of training standard BNNs using VI, which we have discussed in Section 2.4:

1. Difficulty in convergence caused by high variance in the gradient estimates due to the high-dimensional Gaussian posterior.
2. Difficulty in prior specification.
3. Inability to use a complex posterior without incurring a high variational parameter cost.

These problems are consequences of inferring the posterior of a large number of random variables in the form of NNs' weights. iBNNs address them using node parameterisation. Specifically, they only infer the posterior of the layer-wise latent input variables, which has a much smaller dimension than the full weight posterior, thus mitigating the problematic behaviour of the high-dimensional Gaussian posterior. This also simplifies prior specification, as an intuitive prior for the multiplicative latent variables is $\mathcal{N}(\mathbf{1}, \operatorname{diag} \mathbf{s}^2)$, where a good value of $\mathbf{s}$ can be found through cross-validation or maximising the marginal log-likelihood. Furthermore, the low-dimensional latent variables permit the usage of a mixture posterior without adding a substantial number of variational parameters, which significantly improves performance of iBNNs, as we have observed in Chapter 5. The additional performance gain is attained through a good KL constraint, an intuitive training algorithm and good hyperparameter settings, as studied in Chapter 4.

Robustness against OOD samples The authors of [88] showed that one can improve robustness against OOD samples of an ensemble by promoting functional diversity among its members. Likewise, an iBNN achieves its robustness as we apply various measures to improve diversity among posterior components. In particular, we use (i) a training algorithm that trains each component on a different permutation of the training data, (ii) a relaxed KL term allowing components to be far away from each other in the parameter space, and (iii) a large variational learning rate preventing the components from collapsing to a small region. Furthermore, as we jointly train the components, they learn to produce complementary predictions to each other, thus improving the effectiveness of the ensemble. We theorise that this is why iBNNs are better than SWAG in OOD detection, even though both approaches approximate a local mode in the loss landscape.

Low computational and memory cost Compared to methods performing weight posterior inference, iBNNs have lower computational and memory complexity. As a result, iBNNs are an attractive option for (i) applications on edge devices with small computational and memory budgets such as mobile phones, and (ii) current state-of-the-art deep NN architectures whose parameter counts range from hundreds of millions to a few billions.

Ease of hyperparameter selection Finding good hyperparameters for iBNNs is trivial, as we mostly need to focus on tuning the hyperparameters of the variational parameters, while for deterministic parameters, we can reuse the hyperparameters of a corresponding deterministic model. For the variational parameters, we need to select the prior variance $\mathbf{s}^2$, the number of data replications S and the variational learning rate λ_1. For $\mathbf{s}^2$, we can find a good value through cross-validation or maximising the marginal log-likelihood. As we have demonstrated in Chapter 4, a good value for S is 4 and we should set λ_1 to a large value.

Ability to reuse weights of pretrained NNs We have demonstrated in Section 5.3 that an iBNN can reuse deterministic weights from a pretrained NN to save training time and achieve better generalisation performance.

6.2 Disadvantages of iBNNs

Increased training time The training time of iBNNs is higher than SWAG and MC-Dropout, which have the same training time as a similar deterministic NN, since we need to replicate each minibatch S times during training of iBNNs to maintain good predictive performance. While this is unfavourable for very large NNs and datasets, we have shown in Section 5.3 that we can save training time via initialising the deterministic weights of an iBNN using a pretrained model.

Single-mode approximation As an iBNN only approximates a local mode, it is not as good as a DE, which captures multiple modes in the loss landscape. Nonetheless, iBNNs are still useful as they achieve good predictive performance and robustness against OOD samples with lower computational and memory cost.

6.3 The role of low-dimensional posteriors in BNNs

iBNNs follow a line of research finding compact posteriors for BNNs providing good predictive performance and uncertainty [49, 82, 89–91]. The good results of iBNNs and other methods, such as Rank-1 BNNs [49] and Subspace Inference [82], are evidence of the potential for low-dimensional posteriors in BNNs. This is further supported by the study in [92], which showed that the degrees of freedom needed by an NN to solve a particular learning problem are much smaller than the number of weights in the model. Related to this observation is a method called Subspace Inference [82], which first constructs a mapping from the parameter space to a lower-dimensional subspace and then performs Bayesian inference in this subspace to train a BNN. For instance, this method can train a BNN with the WRN-28x10 architecture [79] on CIFAR-100 by performing inference in a 5-dimensional subspace [82]. We also observed this phenomenon in the PCA embeddings of iBNNs in Figure 4.6, showing that the posterior of an iBNN effectively captures high-performing weights within a local, Gaussian-like region in the subspace spanned by the first few PCA components. While the PCA subspaces in Figure 4.6 indicate that the multiplicative latent variables of iBNNs might not be sufficient to capture multiple modes of the loss landscape, we theorise that there are other types of parameterisation that could achieve this goal when combined with an expressive posterior. Therefore, we believe that reducing the dimension of the posterior through a suitable method is a viable way to train a large BNN.

6.4 Future work

Experiments with other NN architectures This thesis focuses mainly on CNN architectures. However, many other popular architectures exist, such as recurrent neural networks [93, 94] and Transformer [5]. It will be interesting to study the behaviour of iBNNs with these architectures.

Experiments with different types of latent variables Here we introduce multiplicative latent variables, which can only capture a single mode in a loss landscape when paired with a mixture of Gaussians posterior. There are other ways to introduce latent variables. For example, we could concatenate the latent vector to the input vector of each layer instead of using point-wise multiplication. With the right type of latent variables and an expressive posterior, we can potentially capture multiple modes in the loss landscape and reach performance of DEs.

Using more expressive posterior distributions Due to the low dimensionality of the latent variables, we can employ more expressive posterior distributions, such as normalising flows [73–76], which can approximate the true posterior of the latent variables more accurately. Thus, it potentially could capture multiple modes in the loss landscape when paired with the right type of latent variables.

Additional measures to improve ensemble diversity Ensemble diversity is vital to predictive performance [67] and robustness against OOD samples [88]. Here we increase the diversity among members of an iBNN's ensemble posterior by training different components on different permutations of the training set and employing a large variational learning rate to prevent them from converging to similar functions. Other methods could also be employed, such as applying data augmentations (either to the inputs, the target labels, or both) or adding additional constraints to the loss function.

Chapter 7

Conclusions

This thesis introduced *implicit Bayesian neural networks* (iBNNs), BNN models that can be applied to deep architectures and large datasets without substantially increasing the computational and memory complexity. This model achieves efficiency by performing posterior inference over the layer-wise latent input variables and treating weights as deterministic variables that can either be trained or initialised from pretrained models, thus simplifying the training problem and mitigating the challenges of training a conventional BNN. We proposed an ensemble variational posterior for the latent variables, as well as a learning objective and a training procedure promoting diversity among ensemble members. Using popular datasets and NN architectures for image classification, we showed that iBNNs achieve competitive performance compared to other scalable BNN methods while having much smaller numbers of parameters. Furthermore, we showed that iBNNs produce reliable uncertainty and are robust against OOD samples.

Even though we jointly optimised the deterministic weights and latent variables in the main experiments, we also showed that iBNNs can utilise pretrained weights and only need to learn the latent variables to induce uncertainty and improve generalisation performance. This notion can help reduce the training time significantly by splitting the procedure into two phases, where we only optimise the deterministic weights in the first phase and then jointly train both the weights and the variational parameters in the second phase. One exciting application is that we can convert large pretrained NNs into iBNNs with minimal training overhead for uncertainty estimation and better predictive performance.

As state-of-the-art NNs become larger, we believe that inducing predictive uncertainty through latent variables is a more viable research direction for BNNs than the standard weight posterior approach. We have demonstrated in this thesis that a suitable type of latent variables paired with an expressive posterior can both improve predictive performance and provide reasonable uncertainty while having low computational and memory cost. Therefore, an interesting research direction is to discover other types of latent variables as well as experiment with more expressive posteriors such as normalising flows [73–76].

Bibliography

[1] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, December 1989.

[2] H. N. Mhaskar. Approximation properties of a multilayered feedforward artificial neural network. *Advances in Computational Mathematics*, 1(1):61–80, February 1993.

[3] Yoshua Bengio and Olivier Delalleau. On the Expressive Power of Deep Architectures. In Jyrki Kivinen, Csaba Szepesvári, Esko Ukkonen, and Thomas Zeugmann, editors, *Algorithmic Learning Theory*, pages 18–36, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[4] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016.

[5] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS*, 2017.

[6] Steffen Schneider, Alexei Baevski, Ronan Collobert, and Michael Auli. wav2vec: Unsupervised pre-training for speech recognition. In *INTERSPEECH*, pages 3465–3469, 2019.

[7] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, May 2015.

[8] Joaquin Quiñonero-Candela, Masashi Sugiyama, Neil D Lawrence, and Anton Schwaighofer. *Dataset shift in machine learning*. MIT Press, 2009.

[9] Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *CVPR*, pages 427–436, 2015.

[10] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On calibration of modern neural networks. In *ICML*, pages 1321–1330, 2017.

[11] M. Hein, M. Andriushchenko, and J. Bitterwolf. Why ReLU networks yield high-confidence predictions far away from the training data and how to mitigate the problem. In *CVPR*, pages 41–50, 2019.

[12] Agustinus Kristiadi, Matthias Hein, and Philipp Hennig. Being Bayesian, even just a bit, fixes overconfidence in ReLU networks. In *ICML*, pages 5436–5446, 2020.

[13] Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *ICML*, 2016.

[14] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *NIPS*, pages 6405–6416, 2017.

[15] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[16] Li Shen, Laurie R. Margolies, Joseph H. Rothstein, Eugene Fluder, Russell McBride, and Weiva Sieh. Deep learning to improve breast cancer detection on screening mammography. *Scientific Reports*, 9(1):12495, August 2019.

[17] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565*, 2016.

[18] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

[19] Mark van der Wilk, Carl Edward Rasmussen, and James Hensman. Convolutional Gaussian processes. In *NIPS*, 2017.

[20] Kenneth Blomqvist, Samuel Kaski, and Markus Heinonen. Deep convolutional Gaussian processes. In *ECML PKDD*, 2019.

[21] Vincent Dutordoir, Mark van der Wilk, Artem Artemev, and James Hensman. Bayesian image classification with deep convolutional Gaussian processes. In *AISTATS*, 2020.

[22] David J. C. MacKay. A practical Bayesian framework for backpropagation networks. *Neural Computation*, 4(3):448–472, May 1992.

[23] Radford M. Neal. *Bayesian Learning for Neural Networks*. Lecture Notes in Statistics. Springer-Verlag, New York, 1996.

[24] Christopher M. Bishop. *Neural networks for pattern recognition*. Oxford University Press, 1995.

[25] Alex Kendall and Yarin Gal. What uncertainties do we need in Bayesian deep learning for computer vision? In *NIPS*, pages 5580–5590, 2017.

[26] Yaniv Ovadia, Emily Fertig, Jie Ren, Zachary Nado, David Sculley, Sebastian Nowozin, Joshua V Dillon, Balaji Lakshminarayanan, and Jasper Snoek. Can you trust your model's uncertainty? Evaluating predictive uncertainty under dataset shift. In *NIPS*, 2019.

[27] Fredrik K. Gustafsson, Martin Danelljan, and Thomas B. Schon. Evaluating scalable Bayesian deep learning methods for robust computer vision. In *CVPRW*, pages 1289–1298, 2020.

[28] Andrew Gordon Wilson and Pavel Izmailov. Bayesian deep learning and a probabilistic perspective of generalisation. *arXiv preprint arXiv:2002.08791*, 2020.

[29] José Miguel Hernández-Lobato and Ryan P. Adams. Probabilistic backpropagation for scalable learning of Bayesian neural networks. In *ICML*, pages 1861–1869, 2015.

[30] Sebastian Farquhar, Michael A Osborne, and Yarin Gal. Radial Bayesian neural networks: Beyond discrete support in large-scale Bayesian deep learning. In *AISTATS*, pages 1352–1362, 2020.

[31] Florian Wenzel, Kevin Roth, Bastiaan Veeling, Jakub Swiatkowski, Linh Tran, Stephan Mandt, Jasper Snoek, Tim Salimans, Rodolphe Jenatton, and Sebastian Nowozin. How good is the Bayes posterior in deep neural networks really? In *ICML*, pages 10248–10259, 2020.

[32] David J. C. MacKay. Probable networks and plausible predictions - a review of practical Bayesian methods for supervised neural networks. *Network: Computation in Neural Systems*, 6(3):469–505, 1995.

[33] Max Welling and Yee Whye Teh. Bayesian learning via stochastic gradient Langevin dynamics. In *ICML*, pages 681–688, 2011.

[34] Tianqi Chen, Emily Fox, and Carlos Guestrin. Stochastic gradient Hamiltonian Monte Carlo. In *ICML*, pages 1683–1691, 2014.

[35] Christos Louizos and Max Welling. Structured and efficient variational deep learning with matrix gaussian posteriors. In *ICML*, pages 1708–1716, 2016.

[36] Ruqi Zhang, Chunyuan Li, Jianyi Zhang, Changyou Chen, and Andrew Gordon Wilson. Cyclical stochastic gradient MCMC for Bayesian deep learning. In *ICLR*, 2019.

[37] Ba-Hien Tran, Simone Rossi, Dimitrios Milios, and Maurizio Filippone. All you need is a good functional prior for bayesian deep learning. *arXiv preprint arXiv:2011.12829*, 2020.

[38] Vincent Fortuin, Adrià Garriga-Alonso, Florian Wenzel, Gunnar Rätsch, Richard Turner, Mark van der Wilk, and Laurence Aitchison. Bayesian neural network priors revisited. *arXiv preprint arXiv:2102.06571*, 2021.

[39] David Blei, Alp Kucukelbir, and Jon McAuliffe. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112:859–877, 2017.

[40] S. Kullback and R. A. Leibler. On Information and Sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, March 1951.

[41] Tom Minka. Divergence Measures and Message Passing. Technical report, Microsoft Research, January 2005.

[42] Thomas Minka. Expectation propagation for approximate Bayesian inference. In *UAI*, 2001.

[43] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural network. In *ICML*, pages 1613–1622, 2015.

[44] Geoffrey E. Hinton and Drew van Camp. Keeping the neural networks simple by minimizing the description length of the weights. In *COLT*, pages 5–13, 1993.

[45] Alex Graves. Practical variational inference for neural networks. In *NIPS*, pages 2348–2356, 2011.

[46] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *ICML*, pages 1278–1286, 2014.

[47] Diederik P. Kingma, Maz Welling, and Soumith Chintala. Auto-encoding variational bayes. In *ICLR*, 2014.

[48] Wesley J Maddox, Pavel Izmailov, Timur Garipov, Dmitry P Vetrov, and Andrew Gordon Wilson. A simple baseline for Bayesian uncertainty in deep learning. In *NIPS*, 2019.

[49] Michael Dusenberry, Ghassen Jerfel, Yeming Wen, Yian Ma, Jasper Snoek, Katherine Heller, Balaji Lakshminarayanan, and Dustin Tran. Efficient and scalable Bayesian neural nets with rank-1 factors. In *ICML*, pages 2782–2792, 2020.

[50] Diederik P. Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. In *NIPS*, pages 2575–2583, 2015.

[51] Andrew Miller, Nick Foti, Alexander D' Amour, and Ryan P Adams. Reducing reparameterization gradient variance. In *NIPS*, 2017.

[52] Anqi Wu, Sebastian Nowozin, Edward Meeds, Richard E Turner, Jose Miguel Hernandez-Lobato, and Alexander L Gaunt. Deterministic variational inference for robust Bayesian neural networks. In *ICLR*, 2019.

[53] Andrew YK Foong, David R Burt, Yingzhen Li, and Richard E Turner. On the expressiveness of approximate inference in bayesian neural networks. *arXiv preprint arXiv:1909.00719*, 2019.

[54] Simon Duane, A.D. Kennedy, Brian J. Pendleton, and Duncan Roweth. Hybrid monte carlo. *Physics Letters B*, 195(2):216–222, 1987.

[55] Radford M. Neal. MCMC using Hamiltonian dynamics. *arXiv preprint arXiv:1206.1901*, 2012.

[56] Jost Tobias Springenberg, Aaron Klein, Stefan Falkner, and Frank Hutter. Bayesian optimization with robust Bayesian neural networks. In *NIPS*, 2016.

[57] Christian P. Robert and George Casella. *Monte Carlo Statistical Methods (Springer Texts in Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2005.

[58] Cuong V. Nguyen, Yingzhen Li, Thang D. Bui, and Richard E. Turner. Variational continual learning. In *ICLR*, 2018.

[59] Arsenii Ashukha, Alexander Lyzhov, Dmitry Molchanov, and Dmitry Vetrov. Pitfalls of in-domain uncertainty estimation and ensembling in deep learning. In *ICLR*, 2020.

[60] Armen Der Kiureghian and Ove Ditlevsen. Aleatory or epistemic? Does it matter? *Structural safety*, 31(2):105–112, 2009.

[61] Yarin Gal. *Uncertainty in Deep Learning*. PhD thesis, University of Cambridge, 2016.

[62] Stefan Depeweg, Jose-Miguel Hernandez-Lobato, Finale Doshi-Velez, and Steffen Udluft. Decomposition of uncertainty in Bayesian deep learning for efficient and risk-sensitive learning. In *ICML*, pages 1184–1193, 2018.

[63] Arya A Pourzanjani, Richard M Jiang, and Linda R Petzold. Improving the identifiability of neural networks for Bayesian inference. In *NIPS Workshop on Bayesian Deep Learning*, volume 4, page 29, 2017.

[64] Anoop Korattikara, Vivek Rathod, Kevin Murphy, and Max Welling. Bayesian dark knowledge. In *NIPS*, 2015.

[65] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[66] Michael Betancourt. A conceptual introduction to Hamiltonian Monte Carlo. *arXiv preprint arXiv:1701.02434*, 2018.

[67] Stanislav Fort, Huiyi Hu, and Balaji Lakshminarayanan. Deep ensembles: A loss landscape perspective. *arXiv preprint arXiv:1912.02757*, 2020.

[68] Kathryn Chaloner. Elicitation of prior distributions. *Bayesian biostatistics*, pages 141–156, 1996.

[69] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *JMLR*, 2014.

[70] Yarin Gal, Rowan McAllister, and Carl Edward Rasmussen. Improving PILCO with Bayesian neural network dynamics models. In *Data-Efficient Machine Learning workshop, ICML*, 2016.

[71] Gregory Kahn, Adam Villaflor, Vitchyr Pong, Pieter Abbeel, and Sergey Levine. Uncertainty-aware reinforcement learning for collision avoidance. *arXiv preprint arXiv:1702.01182*, 2017.

[72] Christos Louizos and Max Welling. Multiplicative normalizing flows for variational Bayesian neural networks. In *ICML*, pages 2218–2227, 2017.

[73] Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *ICML*, pages 1530–1538, 2015.

[74] Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. Made: Masked autoencoder for distribution estimation. In *ICML*, 2015.

[75] Durk P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse autoregressive flow. In *NIPS*, 2016.

[76] George Papamakarios, Theo Pavlakou, and Iain Murray. Masked autoregressive flow for density estimation. In *NIPS*, 2017.

[77] Theofanis Karaletsos, Peter Dayan, and Zoubin Ghahramani. Probabilistic meta-representations of neural networks. *arXiv preprint arXiv:1810.00555*, 2018.

[78] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

[79] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *BMVC*, 2016.

[80] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

[81] Mahdi Pakdaman Naeini, Gregory F. Cooper, and Milos Hauskrecht. Obtaining well calibrated probabilities using Bayesian binning. In *AAAI*, 2015.

[82] Pavel Izmailov, Wesley J. Maddox, Polina Kirichenko, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Subspace inference for Bayesian deep learning. In *UAI*, pages 1169–1179, 2020.

[83] Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. In *ICLR*, 2019.

[84] Dan Hendrycks and Kevin Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. In *ICLR*, 2017.

[85] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS*, 2011.

[86] Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA, USA, 1999.

[87] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *IJCV*, 115(3):211–252, 2015.

[88] Tianyu Pang, Kun Xu, Chao Du, Ning Chen, and Jun Zhu. Improving adversarial robustness via promoting ensemble diversity. In *ICML*, pages 4970–4979, 2019.

[89] Melanie F Pradier, Weiwei Pan, Jiayu Yao, Soumya Ghosh, and Finale Doshi-Velez. Projected BNNs: Avoiding weight-space pathologies by learning latent representations of neural network weights. *arXiv preprint arXiv:1811.07006*, 2018.

[90] Jakub Swiatkowski, Kevin Roth, Bastiaan Veeling, Linh Tran, Joshua Dillon, Jasper Snoek, Stephan Mandt, Tim Salimans, Rodolphe Jenatton, and Sebastian Nowozin. The k-tied normal distribution: A compact parameterization of Gaussian mean field posteriors in Bayesian neural networks. In *ICML*, pages 9289–9299, 2020.

[91] Erik Daxberger, Eric Nalisnick, James Urquhart Allingham, Javier Antorán, and José Miguel Hernández-Lobato. Bayesian deep learning via subnetwork inference. *arXiv preprint arXiv:2010.14689*, 2020.

[92] Chunyuan Li, Heerad Farkhoor, Rosanne Liu, and Jason Yosinski. Measuring the intrinsic dimension of objective landscapes. In *ICLR*, 2018.

[93] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, pages 1735–1780, November 1997.

[94] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *EMNLP*, 2014.

Appendix A

Hyperparameters

A.1 CIFAR experiments

Each experiment is run for 300 epochs. We use SGD optimiser with Nesterov momentum of 0.9 for all experiments. The initial learning rate λ_0 for the deterministic parameters is set to 0.05 for VGG-16 and 0.10 for WRN-28x10. This learning rate is linearly annealed to $0.01\lambda_0$ from epoch 150 to epoch 270. We set the weight decay for the deterministic parameters to 5×10^{-4}. All experiments are run on one Tesla V100-PCIE-32GB. Bellow we present the additional hyperparameters for iBNN and baseline methods:

Deterministic models and DEs We train the deterministic models using the settings presented at above. For DEs, we simply combine predictions from deterministic models trained using different random seeds.

iBNN We use the same hyperparameters for the deterministic weights as presented above, however we use a weight decay of 3×10^{-4} for VGG-16. We do not apply weight decay and learning rate annealing to the variational parameters. We set the prior to $\mathcal{N}(1.0, 0.3^2)$ for VGG-16, and $\mathcal{N}(1.0, 0.1^2)$ for WRN-28x10. We initialise the means of the posterior components with $\mathcal{N}(1.0, 0.75^2)$ for VGG-16, and $\mathcal{N}(1.0, 0.5^2)$ for WRN-28x10. We initialise the standard deviations of these components using $\mathcal{N}(0.05, 0.02^2)$ for all experiments. We linearly increase the KL weight from 0 to 1 for 200 epochs. Table A.1 presents all the other hyperparameters along with training time for this methods.

SWAG We use the same settings and codes from authors of the SWAG paper at https://github.com/wjmaddox/swa_gaussian/.

MC-Dropout We use a dropout rate of 0.05 before every fully-connected and convolution layers for both VGG-16 and WRN-28x10.

Radial BNN We adapt the codes from https://github.com/SebFar/radial_bnn and set the prior to $\mathcal{N}(0.0, 1.0^2)$.

Table A.1: Hyperparameters of iBNN. K is the number of posterior components. λ_1 is the learning rate of the variational parameters. S is the number of data replications.

	VGG-16					WRN-28x10				
	CIFAR-10		CIFAR-100		Training time	CIFAR-10		CIFAR-100		Training time
K	λ_1	S	λ_1	S		λ_1	S	λ_1	S	
1	0.8	4	0.8	4	∼4h15m	0.8	2	0.8	2	∼18h30m
2	1.6	4	1.6	4	∼4h15m	1.6	2	1.6	2	∼18h30m
4	3.2	4	3.2	4	∼4h15m	1.6	2	3.2	2	∼18h30m
8	6.4	4	6.4	4	∼4h15m	3.2	2	9.6	2	∼18h30m
16	6.4	4	6.4	4	∼4h15m	6.4	2	9.6	2	∼18h30m
32	6.4	4	6.4	4	∼4h15m	6.4	4	19.2	4	∼34h15m

A.2 IMAGENET experiments

We use the pretrained ResNet-50 weights from `torchvision.models` package. We set the learning rate λ_0 to 5×10^{-4}, weight decay of 1×10^{-4}, learning rate λ_1 to 0.80 and number of data replications S to 1 for all settings. We use a batch size of 1280. We initialise the means of the posterior components using $\mathcal{N}(1.0, 0.5^2)$ and the standard deviations of these components using $\mathcal{N}(0.05, 0.02^2)$. We set the prior to $\mathcal{N}(1.0, 0.1^2)$. We keep the KL weight fixed at 1.0. Each experiment is run for 15 epochs on four Tesla V100-PCIE-32GB.