MSc Project - Reflective Essay

Project Title:	Comparative Evaluation of Uncertainty Quantification of Bayesian Neural Networks
Student Name:	Pranav Narendra Gopalkrishna
Student Number:	231052045
Supervisor Name:	Paulo Rauber
Programme of Study:	MSc. Artificial Intelligence

Abbreviations

BI	: Bayesian Inference
NN	: Neural Network
ANN	: Artificial Neural Network (means the same as NN)
BNN	: Bayesian Neural Network
MCMC	: Markov Chain Monte Carlo
HMC	: Hamiltonian Monte Carlo
VI	: Variational Inference

Overview

This comparative study explores Bayesian neural networks (BNNs), which is a kind of neural network that can quantify the uncertainty of its predictions. Modelling data using NNs is often a black-box approach, wherein the NN's confidence in its predictions is unknown. BNNs are a means to achieve a level of transparency in neural networks by using BI to quantify its uncertainty and explain – mathematically – how it is quantified. Hence, BNNs are a valuable area to explore in deep learning, and in trying to add as much value to this exploration as possible, I worked on: (1) bridging the gap between BNN's theory and its implementation, (2) implementing two essentially different approaches to quantifying the uncertainty of an NN's predictions, (3) evaluating the results, and (3) making my project reproducible to anyone new to BI.

Repository : https://github.com/pranigopu/mastersProject
Code : https://github.com/pranigopu/mastersProject/tree/main/code
Notes : https://github.com/pranigopu/mastersProject/tree/main/conceptual-notes

Approach

Since BI was an unfamiliar topic at the start of the project, I first studied it from its basics. A key part that bridged the gap between Bayes' rule and computational BI was in grasping the fact that BI was based on distributing the parameterisations of a generative model based on (1) prior knowledge/assumptions and (2) observed data, as explained by Ox educ (2014) and expanded on by Murphy (2012).

My next step was to understand computational BI, which was key to any case where the posterior cannot be reached analytically; in deep learning, where the model's parameterisations are high-dimensional, this is certainly the case. To this end, my first reference was Martin et al. (2021), which explores computational BI both theoretically and practically. In particular, I explored (theoretically, then practically) MCMC and VI. A lot of time was spent on theory, due to its necessity in understanding the implementations.

In particular, I chose to implement HMC as the MCMC method since HMC is a class of MCMC methods that is designed to overcome the weakness of most MCMC methods in multimodal distributions with barely overlapping modes (such as the posterior of an NN's weights), more precisely the fact that exploring such modes requires passing through regions of low probability density, which, by the very nature of MCMC, are rare events (Training BNNs with HMC, 2019). Implementing HMC presupposes grasping its theory in terms of an accurate physical analogy, since HMC is based on Hamiltonian mechanics in physics. Furthermore, a solid mathematical understanding was vital to grasping and thus replicating existing HMC implementations.

I chose to implement VI using an adversarial BNN, as defined in *torchbnn* (Kim, 2020; Lee et al., 2021), a PyTorch-based implementation. Although it is not a VI method explicitly, I posit (and have tried to show in the dissertation) that it can be formulated as a VI method. Due to its availability, ease of application and time constraints (having spent a lot of time on the HMC BNN), I chose to use this method rather than implement a method that is more explicitly VI.

The comparative evaluation of the quality of uncertainty quantification was done using graphs, specifically using the Seaborn package in Python to show (in one plot) the whole range of the model's predictions, the average model predictions and the 95% confidence interval of the distribution of model predictions. All the main evaluation was done using the graphs, which, while quite insightful, is admittedly lacking in rigour.

Strengths and Weaknesses

Though there are implementations of HMC for regression problems, I chose to do my own implementation of an HMC-based BNN for the sake of explainability, using the HMC kernel from the TensorFlow Probability MCMC package. Furthermore, using my own implementation allowed me to document my approach in detail, further enhancing its explainability. While a strength, this is also a weakness since I am forced to rely upon an untested and (as I shall show later) flawed implementation of an HMC BNN.

A key aspect about the HMC BNN is that the bulk of its training is that of a traditional NN. HMC is only applied on the trained model, and thus is used not to train the model but to quantify the uncertainty of the model's predictions. This approach has the advantage of being straightforward in theory, because the weights of a trained model can be expected to represent a high probability sample from the posterior by which the weights are distributed, and thus these weights can serve as a strong starting point for HMC sampling, potentially improving the convergence of the samples toward the true posterior distribution.

If a method to distribute all the weights of the NN can be implemented, then repeated reinitialisation and retraining of the NN followed by HMC sampling starting from the trained weights can help more accurately estimate the posterior of the NN's weights. However, my HMC BNN implementation does not distribute all the weights of the NN, but only the hidden layer's weights. While an advantage of such an approach is that it is relatively much easier to implement, the downside is that it likely does not capture the whole range of potential parameterisations of the NN, since NNs are often sensitive to the initialisation of their weights (as was also observed in my case). Hence, the distribution of the hidden layer's weights while keeping other layers constant fails to explore the many modes of the NN's weights' posterior.

I chose to use a pre-existing implementation for a VI-based BNN, using *torchbnn*, a PyTorch-based implementation presented in Kim, 2020, which draws from the work of Lee et al. (2021). This implementation of a BNN is based on a more well-established source, which is a strength as it allowed me to validate – to some extent – my HMC BNN implementation. However, a weakness of this implementation is that it uses a method that, while technically a VI method, does not (in my knowledge) provide a rigorous mathematical basis such as the Bayes-by-Backpropagation and Evidence Lower Bound (ELBO) method.

For the sake of parity between the two implementations, the VI BNN also only distributes the hidden layer, which, while posing the same issues as in HMC BNN, is a strength in that it allows me to compare my implementation of HMC BNN (which is somewhat untested) to a more well-established BNN implementation.

A key strength in the chosen domain, namely synthetic regression problems, is that the uncertainty of the data is well-defined and is an explicit aspect of the data generation process, which means it is possible to measure how close the uncertainty estimated by the BNN is to the theoretical uncertainty in the data generation process.

Overall, the main strength of my project is its accessibility and thus replicability, since (1) it covers a wide range of topics relevant to anyone new to BI, and (2) it provides well-documented demonstrations of the concepts backed up by a wealth of practical and theoretical notes, and, (3) it explores concepts at progressively more complex stages, starting from basic methods (e.g. Metropolis-Hastings) and basic models (e.g. binomial model).

The key weakness of my project is its highly constrained scope (i.e. relatively small and simple regression problems) and significant theoretical issues (e.g. the fact that only one layer of the NN is distributed). Furthermore, while the performance of the methods show promise as a proof-of-concept, they fail to accurately capture the variability in the observed data. Finally, while there are papers on the evaluation of BNNs using a variety of metrics, I was unable to implement them due to my lack of comprehension. My own evaluation metric (i.e. the KS-test between actual and predicted values, which was not presented in the dissertation but is available in "*code/evaluation.ipynb*" in my project's repository) is of dubious validity (though I have given my reasoning), and even if it is to be accepted, the models perform very poorly even by its standards.

Relating Theory to the Practical Implementation

While a large portion of my time and effort were devoted toward understanding the theory behind HMC, the contribution of my theoretical understanding served mostly to understand how to define the likelihood, prior and thus the unnormalised posterior used for the HMC kernel as defined in the TensorFlow Probability MCMC package. In fact, the likelihood and prior are implemented exactly as per their theoretical basis for the given synthetic regression problems. As for the HMC algorithm, while I studied it to explain it, its implementation was outsourced to the aforementioned *HamiltonianMonteCarlo* kernel.

In theory, HMC is supposed to eventually sample exactly from the posterior of a chosen model parameter, and thus has a theoretical guarantee to converge to the posterior. However, while HMC's gradient-based approach may make it scale well to higher dimensions and, in principle, to more complex geometries compared to other MCMC methods (Martin et al., 2021), it is also the case that gradients may be too complex for high dimensional posteriors (e.g. as in the posterior of an NN's weights) to be computed with high accuracy. Combined with the fact that there we cannot always guarantee that a chosen momentum would take the sampler along the contours of points with the same or similar acceptance probability, errors during HMC sampling for high dimensional posteriors may be too high to ensure high acceptance rates for the proposed samples, leading to slow – if any – convergence

Hence, even in theory, it hard to say what to expect from the HMC samples of the NN's weights. Moreover, the empirical confirmation (or lack thereof) of this convergence in my implementations cannot be relied upon due to the theoretical issues in distributing the weights of only the hidden layer of the NN. Hence, despite MCMC's and thus HMC's theoretical guarantees, these guarantees cannot be relied upon for the practical implementation of HMC BNN as used in my project.

The VI BNN implementation uses two forms of loss: the mean squared error (MSE) loss of measuring errors in point estimation, and the KL-divergence loss between the variational distribution (a normal distribution parameterised by the mean and standard deviation of the NN's Bayesian layer's weights) and the prior (a standard normal distribution). The KL-divergence loss can be understood as regularisation, which is in fact a way to understand priors as such; in particular, the prior $P(\theta) = \mathcal{N}(0, \sigma \mathrm{I})$ ($\mathcal{N}$ denoting a normal distribution) is equivalent to a weighted ℓ_2 regularization (with weights $1/\sigma$) when training a point estimate network (Jospin et al., 2020). The MSE loss can be understood as a means to adjust the variational distribution's parameters according to the observed data, thus closing the gap between the variational distribution and the posterior that it is meant to approximate. This theoretical framework allows us to understand the *torchbnn* implementation of BNNs as a VI method.

Theory is key to interpreting the practical results and hinting at potential areas of exploration. In theory, the following can be proposed: (1) gaps in data should produce more uncertainty in the associated area, (2) data produced by more complex functions are likely to result in a greater uncertainty in their predictions due to complexity, and (3) sparser data should produce more uncertainty in the predictions. These theoretical expectations help evaluate the results of the implementations. In large part, theory aligns with observations. However, in cases such as problem D, where predictions for the gaps in the data showed negligible uncertainty whereas predictions for the denser areas of the data were more varied and less confidence, the theoretical expectations hints at other potential causes, such as (1) the model's simplicity (hence, would a more complex model accurately model the uncertainty?) or (2) the fact that only the hidden layer is being distributed (hence, would a full BNN accurately model the uncertainty?).

Further Work

The scope of my project is relatively small, since it focuses on synthetic regression problems that are relatively easy to train for using basic NN architectures. Hence, one area of expansion of this work is to explore the use BNNs in more complex domains. However, even in its current domain of synthetic regression problems, the project can be expanded in a few significant ways.

A clear first step forward would be to implement a BNN that distributes all its weights and not just the weights of its hidden layer; this would allow a more thorough exploration of the potential of BNNs and their ability to accurately estimate the uncertainty of their predictions. Implementing such BNNs was the original aim, but this aim was abandoned due to implementation issues in the HMC BNN (as discussed in the code files).

Secondly, more well-founded evaluation metrics for measuring the performance of BNNs, such as ones mentioned in Jospin et al. (2020) and used in Brian and Da Veiga (2022), can help more rigorously identify and compare the strengths and weaknesses of the BNN implementations used. A major challenge here is translating the theory behind the given metrics into interpretable implementations.

Thirdly, this project involves only two types of BNNs; a wider variety of BNNs can be explored, as in Brian and Da Veiga (2022) and Foong et al. (2019), such as the use of ELBO, Monte Carlo dropout, stochastic gradient MCMC, etc. To this end, a variety of

MCMC and VI methods can be compared within each group, leading to a stronger grasp of effective Bayesian approaches, be they exact or approximate.

Awareness of Legal and Ethical Issues

Due to the data used being synthetic, no legal or ethical issues arise from the procurement of the data. This project involves no violation of intellectual property and uses publicly available resources, such as implementations, explanations and papers; all the resources used are explicitly referenced, both in the paper and in the project’s code. As for the area of study, i.e. BNNs, they offer a potential for improving the transparency of an AI model, thereby being a potential step toward improving the trustworthiness and interpretability of AI systems.

Citations

[1] Foong, A. Y. K., Burt, D. R., Li, Y. and Turner, R. E. (2019). On the Expressiveness of Approximate Inference in Bayesian Neural Networks. *arXiv preprint arXiv:1909.00719.*

[2] Jospin, L. V., Laga, H., Boussaid, F., Buntine, W. and Bennamoun, M. (2020). Hands-on Bayesian Neural Networks – A Tutorial for Deep Learning Users. *arXiv preprint arXiv:2007.06823.*

[3] Kim, H. (2020). *Bayesian-Neural-Network-Pytorch*. [online] Available from: https://github.com/Harry24k/bayesian-neural-network-pytorch [Accessed 21 August 2024].

[4] Lee, S., Kim, H. and Lee, J. (2021). GradDiv: Adversarial Robustness of Randomized Neural Networks via Gradient Diversity Regularization. *arXiv preprint arXiv:2107.02425.*

[5] Murphy, K. P. (2012). '3. Generative models for discrete data'. *Machine Learning: A Probabilistic Perspective*. London, England: The MIT Press. pp. 65-87.

[6] Ox educ. (2014). *Bayesian vs frequentist statistics*. [online] Available from: https://youtu.be/r76oDIvwETI?si=Fk8-Z6kLIhqULcTq [Accessed 21 August 2024].