Uncertainty in Neural Networks

Bayesian Ensembles, Priors & Prediction Intervals

Tim Pearce

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of
Doctor of Philosophy

Queens' College   December 2020

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 65,000 words including appendices, bibliography, footnotes, tables and equations and has fewer than 150 figures.

Tim Pearce
December 2020

Acknowledgements

- My supervisors - Andy Neely, Mohamed Zaki, and Alexandra Brintrup - for empowering me on my PhD journey. It's been a pleasure being your student.
- The Alan Turing Institute; this would be a very different thesis if not for my year spent there. Special thanks to Nicolas Anastassacos, Ayman Boustati, and Jessie Liu.
- The Engineering Department, IfM, CSA, DIAL, and MAG. In particular to Ahmed Alali, Philipp Koebnick, and Adria Salvador Palau.
- PROWLER.io and Felix Leibfried for an enjoyable and productive internship.
- Russell Tsuchida, Bang Xiang Yong, Andrew Y. K. Foong, and David Ratiney for enjoyable collaborations.
- Richard Turner and Laurence Aitchison as examiners. Neil Lawrence and Mark Girolami for providing feedback on this thesis.
- EPSRC for funding my PhD. Queens' College and the Engineering Department for helping with travel grants. The Alan Turing Institute for further financial assistance.
- My parents; my mum for passing on her discipline and attention to detail, and my dad for handing down his knack for unusual ideas. This thesis required an equal measure of each.

Abstract

The breakout success of deep neural networks (NNs) in the 2010's marked a new era in the quest to build artificial intelligence (AI). With NNs as the building block of these systems, excellent performance has been achieved on narrow, well-defined tasks where large amounts of data are available.

However, these systems lack certain capabilities that are important for broad use in real-world applications. One such capability is the communication of uncertainty in a NN's predictions and decisions. In applications such as healthcare recommendation or heavy machinery prognostics, it is vital that AI systems be aware of and express their uncertainty – this creates safer, more cautious, and ultimately more useful systems.

This thesis explores how to engineer NNs to communicate robust uncertainty estimates on their predictions, whilst minimising the impact on usability. One way to encourage uncertainty estimates to be robust is to adopt the Bayesian framework, which offers a principled approach to handling uncertainty. Two of the major contributions in this thesis relate to Bayesian NNs (BNNs).

Specifying appropriate priors is an important step in any Bayesian model, yet it is not clear how to do this in BNNs. The first contribution shows that the connection between BNNs and Gaussian Processes (GPs) provides an effective lens to study BNN priors. NN architectures are derived which mirror the combining of GP kernels to create priors tailored to a task.

The second major contribution is a novel way to perform approximate Bayesian inference in BNNs using a modified version of ensembling. Novel analysis improves an understanding of a technique known as randomised MAP sampling. It's shown this is particularly effective when strong correlations exist between parameters, making it well suited to NNs.

The third major contribution of the thesis is a non-Bayesian technique that trains a NN to directly output prediction intervals for regression tasks through a tailored objective function. This advances over related works that were incompatible with gradient descent, and ignored one source of uncertainty.

Contents

List of Figures **xv**

List of Tables **xxi**

1 Introduction **1**

1.1 Big Picture 1

1.2 Breakout Success of Deep Neural Networks 1

1.3 The Value and Cost of Being Uncertain 2

1.3.1 Auxiliary Uses 3

1.4 Research Goal 4

1.5 Contributions 5

1.6 Thesis Structure 6

1.7 Publication List 7

2 Background **9**

2.1 Anatomy of Uncertainty 10

2.1.1 Regression 11

2.1.2 Classification 12

2.1.3 Other Uncertainties 13

2.2 Neural Networks 13

2.2.1 Origins: Modelling Biological Neurons 14

2.2.2 From Neurons to Deep Neural Networks 15
2.2.3 Learning in NNs 18
2.3 Modelling Uncertainty with Neural Networks 19
2.3.1 Can Standard Neural Networks Model Uncertainty? 19
2.3.2 Overview of Methods 20
2.4 Making Neural Networks Bayesian 22
2.4.1 Distributions Over Parameters 22
2.4.2 Posterior Predictive Distribution 25
2.4.3 Learning in BNNs 25
2.5 Gaussian Processes 27
2.5.1 Sampling Functions Using a Gaussian Distribution 28
2.5.2 Fitting GPs to Data 30
2.5.3 Combining Kernels 31
2.6 Connections Between BNNs and GPs 33
2.6.1 BNNs Converge to GPs 34
2.6.2 Analytical BNN Kernels - Single-Layer NNs 36
2.6.3 Analytical BNN Kernels - Deep NNs 38
2.6.4 Implications of the Connection 39
2.7 Summary: Model vs. Inference 40

3 Expressive Priors in BNNs 41
3.1 GPs as a Lens to Study BNN Priors 42
3.2 Interpreting BNN Priors 42
3.3 Specifying Priors for BNNs via Proxy GPs 47
3.3.1 Illustrative Experiment: Model-Based RL 47
3.4 Kernel Combinations in BNNs 51
3.4.1 Combining BNNs at Output 51
3.4.2 Combining BNNs at Hidden Layers 52

3.4.3 Extensions . . . 54
3.5 Periodic BNN Kernels . . . 56
3.5.1 Cosine Activations . . . 57
3.5.2 Input Warping . . . 57
3.6 Illustrative Experiments . . . 59
3.6.1 Supervised Learning: Time Series . . . 59
3.6.2 Reinforcement Learning: Pendulum Swing up . . . 61
3.7 Related Work . . . 64
3.8 Discussion & Conclusion . . . 65

4 Bayesian Ensembling 67
4.1 Randomised MAP Sampling . . . 69
4.2 RMS Theoretical Results . . . 69
4.2.1 Parameter-Space Derivation . . . 70
4.2.2 Comparison to Prior Work . . . 71
4.2.3 Practical Workaround: General Case . . . 71
4.2.4 Practical Workaround: Special Cases . . . 73
4.3 RMS for Neural Networks . . . 73
4.3.1 Loss Function . . . 74
4.3.2 Validity of RMS in NNs . . . 75
4.3.3 Implementation Practicalities . . . 79
4.4 Experiments . . . 80
4.4.1 Qualitative Tests . . . 80
4.4.2 Convergence Behaviour . . . 81
4.4.3 UCI Regression Benchmarks . . . 81
4.4.4 Out-of-Distribution Classification . . . 84
4.4.5 Reinforcement Learning . . . 86
4.5 Related Work . . . 87

4.6 Discussion & Conclusion 89

5 High-Quality Prediction Intervals 91

5.1 Contributions 92

5.2 Prediction Intervals 93

5.3 A Quality-Driven, Distribution-Free Loss Function 94

5.3.1 Derivation 94

5.3.2 Comparison to LUBE 96

5.3.3 Training QD with Gradient Descent 96

5.3.4 Particle Swarm Optimisation 99

5.4 Ensembles to Estimate Model Uncertainty 100

5.5 Experiments 101

5.5.1 Qualitative Experiments 101

5.5.2 UCI Regression Benchmarks 102

5.6 Limitations 105

5.6.1 Varying Data Noise 105

5.6.2 Uniform Data Noise 106

5.7 Related Work 106

5.8 Discussion & Conclusion 107

6 Discussion & Conclusion 109

6.1 Summary of Contributions 109

6.1.1 Follow up Work 110

6.2 Visualising the Field Today 111

6.3 Looking to the Future 112

6.3.1 Uncertainty in the Brain 113

6.4 Future Work 116

6.5 Conclusions 117

Bibliography **119**

Appendix A Bayesian Ensembling **131**

A.1 Proofs 131

A.2 MAP solution and regularisation interpretation 143

A.3 Numerical Examples 145

A.4 Additional Material 150

A.5 Experimental Details 153

Appendix B High-Quality Prediction Intervals **161**

B.1 Experimental details 161

List of Figures

2.1 Visualising aleatoric uncertainty. Even with a large amount of data, uncertainty may remain. Left: Low homoskedastic noise. Middle: High homoskedastic noise. Right: Heteroskedastic noise. 11

2.2 Visualising epistemic uncertainty. Left: A prediction of y when $\mathbf{x} = 0.5$ should be more certain than when $\mathbf{x} = 4.0$. Right: Uncertainty in $f(\mathbf{x})$ increases further from the training data. 12

2.3 1-D binary classification task showing two sources of uncertainty. Blue lines show plausible estimates for $\hat{P}(y = c_1|\mathbf{x})$. The multiple lines arise from uncertainty in $f(\mathbf{x})$. 13

2.4 Left: Modern diagram of a single neuron. Right: Early sketches of neuons based on thin slices of brain tissue by Ramón y Cajal. 14

2.5 Diagram of an early perceptron by Rosenblatt (1958). 15

2.6 Diagram of a single neuron. Left: Overlaid on biological neuron. Right: Abstracted computational graph. 16

2.7 Single-layer NN (biases not drawn for clarity). 16

2.8 Deep NN with three hidden layers. 17

2.9 NNs with various connectivity patterns. Left: Autoencoder. Middle: Recurrent NN. Right: CNN. 18

2.10 A neuron with point estimates over parameters produces a point estimate at its output. If parameters are modelled with distributions (parametric or empirical), the output will also be a distribution. 22

2.11 Two realisations of a Gaussian white noise process. The x-axis and y-axis labels are dropped in subsequent plots. 28

2.12 Squared exponential kernel, with varying lengthscale. 29

2.13 Periodic kernels. 30

2.14 Above: Prior draws for three different kernels. Below: Marginal log likelihood (MLL) scores for these GP kernels fitted to the same dataset. The SE kernel with small l produces priors that are visually more similar to the data, which is quantified by the larger MLL score. 32

2.15 Left: Dataset suiting an additive kernel combination. Right: Prior draws of a suitable additive kernel combination. 32

2.16 Left: Dataset suiting a multiplicative kernel combination. Right: Prior draws of a suitable multiplicative kernel combination. 33

2.17 Prior draws when different kernels are applied to different dimensions of the input. The right-hand plot uses two SE kernels with differing lengthscales. . 33

3.1 Priors from finite-width BNNs provide an increasingly close approximation to priors drawn from the equivalent GP kernel when BNN width, H, is increased - here for ERF activation and kernel. 42

3.2 Top: Prior draws for infinitely wide single-layer BNNs of various architectures. Bottom: Predictive distributions on toy data. 43

3.3 Final layer variances: σ_{w2}^2 controls the amplitude of the whole function, whilst σ_{b2}^2 controls the spread. ERF activations shown. 44

3.4 Final layer variances for classification: σ_{w2}^2 controls the both absolute confidence and the speed at which class predictions can switch. Single-layer NN with ReLU activations shown. 44

3.5 Prior draws for a single-layer BNN with ERF activations. Increasing $\sigma_{w1}^2, \sigma_{b1}^2$ creates noisier functions. Here, $\sigma_{w2}^2 = 1, \sigma_{b2}^2 = 0$. 45

3.6 Kernel analysis, where $\sigma_{w2}^2 = \sigma_{b2}^2$ and $\sigma_{w2}^2 = 1, \sigma_{b2}^2 = 0$. Input angle, θ^0 between two inputs is varied whilst the norm, $||\mathbf{x}||||\mathbf{x}'|| = 1$ is held constant. 45

3.7 Prior draws (top & middle: regression, bottom: classification) for finite BNNs of increasing depth. These degenerate either into input-independent constants or white noise. In red are prior draws from the 'infinite' kernel; that is, by running the recurrence in section 2.6.3 to infinity. 46

3.8 Using a proxy GP to specify a prior for a single-layer BNN with ERF activations. In step 1, the parameters of the GP ERF kernel (eq. 2.39) are optimised. Note the tuned variance in the GP model is normalised by H when building the BNN. 48

3.9 The cartpole control task is a classic RL problem. Rewards are maximised by balancing the pole upright. The cart is moved left and right to achieve this, forming the action space for the environment. 48

3.10 Performance on a model-based RL task, cartpole, comparing a BNN with prior tuned using the proposed method (via proxy GP) vs. fixed default prior. Mean $\pm$ two standard errors, repeated for three runs. 49

3.11 BNN architecture determines our prior belief about a function's properties. In general BNNs provide little flexibility in this regard - modifying only the activation function and length scale ('Basic BNNs'). This chapter explores how to design BNNs to produce more expressive prior functions ('Combinations of Basic BNNs'). Two prior draws are shown for each BNN architecture. . 50

3.12 Prior draws for a 2-D input. Square brackets designate which dimension(s) each kernel is applied to. 56

3.13 Two time series with seasonal fluctuations and long term trends. ReLU and periodic BNNs are separately unable to capture these patterns (first and second from left). However, they succeed when combined in BNN architectures as proposed (third from left), closely approximating the predictive distributions of exact GP inference with the equivalent kernel combinations (right). See main text for model details. 60

3.14 Learning for three different BNN architectures on the pendulum task. The BNN incorporating a suitable prior for task, 'Periodic$\times$TanH', outperforms basic BNNs. Mean $\pm$ 1 standard error, three runs. 62

3.15 Q-values learnt for the action, torque $= 0$, conditioned on observations of $\dot{\theta} = 0$, and input angle, θ, varied on x-axis and measured in radians. The x-axis range allows observation of three complete rotations of the pendulum (each rotation given by 2π radians). Hence, maximum Q-value is predicted when the pendulum is orientated directly upwards. Periodic$\times$TanH captures the local periodicity of the function most accurately. 63

4.1 An ensemble of NNs, starting from different initialisations and trained with the proposed modification, produce a predictive distribution approximating that of a GP. This improves with number and width of NNs. 67

4.2 Demonstration of (exact) RMS in a 2D parameter space. 70

4.3 Examples of the RMS approximate posterior when $\boldsymbol{\mu}_{\text{anc}} := \boldsymbol{\mu}_{\text{prior}}, \Sigma_{\text{anc}} := \Sigma_{\text{prior}}$. MFVI also shown. 74

4.4 Predictive distributions produced by various inference methods (columns) with varying activation functions (rows) in single hidden layer NNs on a toy regression task. 75

4.5 Empirical plots of conditional likelihoods for 4 randomly sampled parameters in two-layer NNs. 76

4.6 Left: Single layer NN of two hidden nodes. Middle: Draws in parameter space for prior (red), analytical posterior (blue) and anchored posterior (green). Right: Posterior predictive distribution - dashed red lines are elbows of ReLU units. 78

4.7 Comparison of NN ensemble loss choices. 79

4.8 The predictive distribution of an anchored ensemble approaches that of a ReLU GP. 80

4.9 Difference in predictive distributions of an anchored ensemble and a ReLU GP as a function of width and number of NNs. Mean ± 1 standard error. . . 81

4.10 Anchored ensembling creates uncertainty-aware agents. 86

4.11 Anchored ensembling creates robust MBRL agents in noisy environments. Mean and standard error over five runs. 87

5.1 Set up for the toy problem: Left is input data (10 data points linearly spaced at fixed input $x = 1.0$), right is the NN used (one trainable weight). 97

5.2 Error surface of Loss_{QD}, and $\text{Loss}_{QD\text{soft}}$ on toy problem, with effect of hyperparameters λ and s. 98

5.3 Comparison of PI boundaries for GD vs. PSO training methods. Shading is the predicted 95% PI. Ground truth is given by blue lines - the ideal 95% boundaries and mode. 101

5.4 Comparison of PI boundaries for two loss functions, QD vs. MVE, given data noise variance drawn from different distributions. Legend as for figure 5.3. 102

5.5 Estimation of model uncertainty with a QD ensemble. 103

5.6 QD assures global but not pointwise coverage probability. 105

5.7 When fitting a single QD NN to uniform data noise, many PIs will give the same MPIW and PICP. This can lead to diversity in the ensemble, even in regions with much data, and model uncertainty estimates can be overestimated. 106

6.1 Qualitative, approximate view of today's approaches to uncertainty estimation in NNs plotted according to the two criteria targeted in this thesis. Left: The broad classes of techniques. Right: Impact of work in this thesis, comparing the original methods (Deep Ens, LUBE), to the proposals of this thesis (Anch Ens, HQPIs). Circle size roughly represents variability within methods and error of the estimation. 112

A.1 Fashion MNIST OOD data examples. 156

List of Tables

2.1 Ability of standard NNs to quantify uncertainty types. For regression, it's assumed prediction is of a single scalar value. 20

4.1 NLL regression benchmark results. See appendix A.4 for RMSE (table A.1) and variants of our method. Mean ± 1 standard error. 82

4.2 Proportion of predictions that were high confidence on out-of-distribution data, e.g. a single regularised VGG-13 CNN trained on CIFAR-10 made high confidence predictions 54% of the time when asked to predict on MNIST. Mean over five runs (three for CIFAR). 'Reg.' refers to standard L2 regularisation, 'Uncons.' refers to no regularisation, 'Anch.' refers to the proposed anchored regularisation, 'nxNNs' refers to an ensemble of n NNs. 85

5.1 PI quality metrics on ten benchmarking regression datasets; mean $\pm$ one standard error, best result in bold. Best was assessed according to the following criteria. If $PICP \geq 0.95$ for both, both were best for $PICP$, and best $MPIW$ was given to smallest $MPIW$. If $PICP \geq 0.95$ for neither or for only one, largest $PICP$ was best, and $MPIW$ only assessed if the one with larger $PICP$ also had smallest $MPIW$. 104

A.1 Variants of our method on benchmark regression datasets, RMSE. Deep ensembles are included for reference, using published results. 151

A.2 Variants of our method on benchmark regression datasets, NLL. 151

A.3 Comparison against inference methods on UCI benchmark regression datasets, log likelihood. Adapted from Mukhoti et al. (2018), also included is the single layer homoskedastic VI results from Tomczak et al. (2018). 152

A.4 Fashion MNIST results: proportion of predictions made with $\geq 90\%$ probability, and entropy of predicted categorical distribution. Also shown is relative advantage (percentage change) for each method compared to anchored ensembles. Averaged over five runs/random seeds, mean $\pm$ 1 standard error. Best result in blue. 152

A.5 Hyperparameters used for regression benchmark results. 156

B.1 Full permutation results of training method and loss function on synthetic data with differing noise distributions. Mean $\pm$ one standard error. 163

B.2 RMSE and NLL on ten benchmarking regression datasets; mean $\pm$ one standard error, best result in bold. 164

Chapter 1

Introduction

TLDR: Methods for understanding the uncertainty of neural networks are important.

1.1 Big Picture

This thesis contributes to the discipline of building systems that learn from data; machine learning. This promises to deliver flexible, general systems capable of performing tasks requiring some level of cognitive ability. Increasingly, this is referred to under the more grandiose term; artificial intelligence (AI).

AI is beginning to have a major impact on the world. It holds the potential to create new products and services from spam filters to autonomous cars; improve existing ones such as medical diagnosis and language translation; and automate mundane tasks such as production line quality inspection and reading of letter addresses. Projections suggest global GDP could be 14% higher by 2030 as a direct result of AI (Rao and Verweij, 2017).

This thesis focuses on an engineering challenge currently blocking these systems from being more widely applicable in the real-world, and hence unlocking some of this value. It does not focus on a specific application, rather contributing to the fundamental technology that underlies many use cases.

1.2 Breakout Success of Deep Neural Networks

The recent revival of enthusiasm and hope in AI can be put down to the success of a particular class of machine learning model; deep neural networks (NNs). These have proven incredibly effective on specific classes of tasks that traditional approaches struggled with.

Whilst early protoypes of NNs can be traced back as early as 1958 (see section 2.2.1), their value was demonstrated in the 2010s when they were run at scale, training wider, deeper NNs (‘deep learning’), over larger datasets. Notably, these NNs were applied to an image classification task (Russakovsky et al., 2015), delivering a substantial reduction in error rate (Krizhevsky et al., 2012).

As well as a performance improvement, NNs bring an attractive paradigm shift. Previous approaches relied on carefully extracting features using hand-designed rules, followed by a small learnable component. NNs replace this with a single end-to-end learning system.

Successes have followed across a variety of vision, audio, control and natural language tasks (Lillicrap et al., 2016; Ping et al., 2018; Radford et al., 2018). NNs have also been combined with the reinforcement learning (RL) framework, creating systems that surpassed human performance in games such as Go, chess, classic 1980s video-games, and modern strategy video-games (Mnih et al., 2015; Silver et al., 2017; Vinyals et al., 2019).

It might appear that this list of achievements could be straightforwardly extended to include any task of interest. However, this overlooks several limitations of today’s NNs. It’s been observed that, *‘it’s easier to build a world champion chess system than it is a mediocre plumber’* [1]. Many sub-fields in machine learning today are, directly or indirectly, developing solutions to these limitations. This thesis is no exception. It contributes to the research effort of building a fundamental property into these systems; that of knowing their uncertainty.

1.3 The Value and Cost of Being Uncertain

The success stories in deep learning to date have been in scenarios where massive datasets were available or could be simulated. ‘Success’ is also in terms of metrics aggregated over a large number of test examples, drawn from the same distribution as the training data. For example, the ImageNet 2015 challenge contained 1.2 million labelled images, with accuracy reported over 0.1 million similar test images (Russakovsky et al., 2015).

This differs in at least two important ways from systems deployed in the real world.

1. Real-world systems do not have guarantees that the data they see when deployed will always be similar to what they were trained on.

2. In real-world systems the confidence in *individual* (rather than aggregate) predictions can matter.

[1] Lex Fridman in interview with Kai-Fu Lee, 2020

Both these problems are exacerbated by the inability of NNs to properly express their uncertainty.

Regarding point 1), if a system is able to express its uncertainty when presented with an unfamiliar input, there are simple measures that might be implemented, such as referring the case to an expert (a clinician in the example of medical diagnosis), or adopting less risky behaviour (reducing speed in the case of an autonomous vehicle).

To understand point 2), consider a NN predicting the time to failure of a key piece of factory machinery. The system notifies the plant manager that the machine will fail in 60 days. This leaves the manager unsure whether to repair the machine immediately, or if it can be left to run for another 50 days. If the system could communicate its uncertainty with a range this difficulty would be avoided - e.g. 50-70 days with 95% confidence.

In these examples, the value of uncertainty estimates is high. This is in contrast to a typical machine learning benchmark, where classification accuracy might be measured over a large set of test data points, drawn from the same distribution as the training data. Here there is little to be gained from good uncertainty estimates.

As well as there being value to uncertainty estimates, there is also a cost. NNs generally used in practise today are poor at estimating their uncertainty. The techniques reviewed and proposed in this thesis all require modifying these default systems. This brings a cost comprised of two main components; 1) extra implementation effort 2) extra computational requirements. This cost will henceforth be referred to as ‘impact on usability’.

Ideally, one would like uncertainty to have low impact on usability, however, there can be a trade-off between impact on usability and quality of uncertainty estimates. This thesis takes the practical view that methods to quantify the uncertainty of NNs should minimise the impact on usability, whilst producing good quality, or ‘robust’, uncertainty estimates. This is discussed in detail in chapter 6.

1.3.1 Auxiliary Uses

The previous section illustrated the value of uncertainty estimates following deployment of a standard supervised learning system. Aside from this direct use case, uncertainty estimates are valuable in an auxiliary role to many other areas in machine learning. Some of these are listed below.

The most straightforward approach to training NNs requires a large labelled dataset. Whilst for many tasks unlabelled data may be abundant, sourcing labels can be a challenging and resource-intensive activity. Active learning provides a paradigm for label collection - it allows an algorithm to select which data points it believes it would benefit from being labelled most, hence increasing the efficiency of the labelling process. Having good quality uncertainty estimates available can be important in driving this selection process (Hernández-Lobato and Adams, 2015). This was highlighted as important to Tesla's efforts in building autonomous vehicles[2].

RL describes a learning paradigm where an agent takes actions in an environment, receiving rewards and states. One challenging aspect of this set up is the exploration/exploitation dilemma; at each step the agent must decide whether to exploit its current knowledge, by choosing the action it has observed to be best so far, or explore some new action, which might prove better or worse. Having uncertainty estimates provides agents with an effective approach to handling this dilemma - if the agent is certain an action is optimal, it should exploit its knowledge, whilst if its less certain, it should explore more often (Osband et al., 2016).

These illustrations provide two examples of how uncertainty can be valuable to other subfields in machine learning. Other areas that can benefit include adversarial learning (Smith and Gal, 2018), model-based RL (Chua et al., 2018), Bayesian optimisation (Springenberg et al., 2016), meta learning (Finn et al., 2018), continual learning (Nguyen et al., 2018) and behavioural cloning (Menda et al., 2019).

1.4 Research Goal

The previous section motivated the need for uncertainty estimates on the predictions of a NN. This is an active, ongoing research area, and current methods for achieving this, in one way or another, are lacking. This has led to the research goal of this thesis.

Research Goal
Develop techniques enabling NNs to communicate *robust uncertainty estimates* on their predictions, whilst minimising *impact on usability*.

'Impact on usability' was previously defined. The research goal also places emphasis on the uncertainty estimates being useful and reliable - here termed 'robust', due to common usage in the field. In simple terms, the goal states the uncertainty estimates be of maximal value and minimal cost.

[2]Andrej Karpathy, AI for Full-Self Driving, Scaled Machine Learning Conference 2020

There are numerous directions worthy of exploration with this goal in mind. Of the various existing techniques (section 2.3), each has advantages warranting further analysis, and disadvantages warranting modification.

This thesis has chosen to drill down into several distinct areas within the research goal that are of high importance to the field. Each investigation will be presented separately, since they study slightly different perspectives on the same goal, and their effectiveness is most easily demonstrated in isolation. However, it should be noted that they are not necessarily mutually exclusive ideas. Chapter 6 provides a framework to visualise the value produced by the contributions of the thesis.

1.5 Contributions

One of the recurring themes in this thesis is the distinction between Bayesian vs. non-Bayesian methods. The Bayesian framework gives a rigorous and widely accepted approach to handling uncertainty in any predictive model. As such, it is one of the most important approaches to handling uncertainty with NNs.

One challenge of the Bayesian approach in NNs, is that it is not obvious how to specify parameter priors that capture appropriate prior knowledge about a task. The first contribution of the thesis is both a practical tool to do this, and a lens to understand the meaning of parameter priors.

Contribution 1 - Expressive Priors for Bayesian NNs, Chapter 3
This chapter proposes to use the link between GPs and NNs to design better priors for Bayesian NNs. The link allows conversion of a functional prior (in a GP) to a parameter prior (in a Bayesian NN). The main contribution then shows how a popular idea for prior design in GPs (kernel combinations) can be translated to Bayesian NNs.

A second challenge of Bayesian NNs is how to efficiently learn the posterior parameter distribution. Indeed a large proportion of the literature in Bayesian NNs focuses on this. Stochastic gradient descent (SGD) is effective in optimising deterministic NNs, but it is not straightforward to apply this to the setting where parameters are modelled as distributions. On the other hand, ensembles of NNs do use SGD and provide a practical way of computing uncertainty estimates - a number of independent NNs are trained with SGD, and the variance of their predictions is interpreted as uncertainty. Our second contribution reconciles these two approaches.

Contribution 2 - Bayesian Ensembling, Chapter 4
Ensembles of NNs have often been shown to outperform Bayesian methods, though are sometimes criticised for lacking a rigorous theoretical framework. This has led to a difficult choice; select the ad-hoc method that works well empirically, or use the more theoretically grounded method. The chapter introduces a third alternative; a modified ensembling procedure that does have strong to connections to the Bayesian framework. Combining these two paradigms results in performance superior to vanilla ensembling.

The third contribution of the thesis moves away from Bayesian methods and considers a NN with two outputs representing the upper and lower bounds of a prediction interval (PI). This makes for an interesting contrast with earlier contributions in Bayesian methods. On the plus side, the intuitive NN architecture and loss function provide a transparent and directly interpretable set up. On the downside, we document a failure mode of the method, providing an exemplum of the potential perils of non-Bayesian methods.

Contribution 3 - High-Quality Prediction Intervals, Chapter 5
Adopting the axiom that PIs should be as narrow as possible subject to a certain coverage proportion, this chapter proposes a carefully designed objective function that can be used with SGD. This holds a significant advantage over previous work, which was not compatible with SGD. The chapter also shows that previous work failed to capture one source of uncertainty entirely, and proposes combining with a (vanilla) ensemble to address this.

1.6 Thesis Structure

The following is an overview of the thesis structure.

Chapter 1 provides a high-level overview of deep learning, and motivates the need for NNs to be able to estimate their uncertainty. The goals, contributions and structure of the thesis are communicated.

Chapter 2 provides prerequisite knowledge to understand later contributions. The following topics are introduced: uncertainty modelling, NNs, uncertainty modelling with NNs, BNNs, GPs, and the connection between BNNs and GPs. The chapter serves as a broad literature review for the thesis, though each technical chapter also contains a focused related work section.

Chapter 3 presents the first technical contribution of the thesis, which helps specify appropriate priors for BNNs. Broadly, the chapter utilises the link between GPs and BNNs to provide a lens to study the effect of parameter priors in a NN on functions. Most significantly, it translates a popular idea for prior design in GPs (kernel combinations) to BNN architectures for the first time. It draws from publications (Pearce et al., 2020b, 2019b; Tsuchida et al., 2020).

Chapter 4 studies a modified version of ensembling NNs; parameters are regularised around a random draw from the prior. The chapter shows this provides connections to a little-known Bayesian inference technique, 'randomised MAP sampling'. Novel theoretical results as well as empirical experiments are offered. This forms the thesis's second contribution and draws from publications (Pearce et al., 2018a, 2019a).

Chapter 5 derives a loss function allowing a NN to directly output PIs, and shows how it can be made compatible with gradient descent. An explanation for why it should be used together with ensembling is also given. The related publication is (Pearce et al., 2018b).

Chapter 6 firstly summarises the contributions of the thesis, and lists follow up work already completed by other authors. The current field is visualised by plotting existing methods on two axes; robustness of uncertainty estimates and impact on usability. We detail the impact of the contributions of this thesis. The chapter then takes a more speculative view of the future of uncertainty in AI. Drawing from (Pearce et al., 2020a), it argues for exploring new paradigms for uncertainty quantification as AI systems grow ever larger and more general. Finally, directions for future work generated throughout the thesis are curated.

1.7 Publication List

The following publications, preprints and extended abstracts were produced over the course of the PhD. Unless stated, the first author of the paper was responsible for the vast majority of technical work.

- AS Palau, K Bakliwal, MH Dhada, **T Pearce**, AK Parlikad
  Recurrent Neural Networks for real-time distributed collaborative prognostics

IEEE International Conference on Prognostics and Health Management 2018
- T Pearce contributed to the implementation and tuning of the RNN.

- **T Pearce**, M Zaki, A Brintrup, A Neely
  High-Quality Prediction Intervals for Deep Learning: A Distribution-Free, Ensembled Approach
  ICML 2018

- **T Pearce**, R Tsuchida, M Zaki, A Brintrup, A Neely
  Expressive Priors in Bayesian Neural Networks: Kernel Combinations and Periodic Functions
  UAI 2019

- R Tsuchida, **T Pearce**, C Van Der Heide, F Roosta, M Gallagher
  Avoiding Kernel Fixed Points: Computing with ELU and GELU Infinite Networks
  AAAI 2020
  - T Pearce set up the GELU kernel derivation, and ran the benchmarking regression experiments.

- **T Pearce**, N Anastassacos, M Zaki, A Neely
  Bayesian Inference with Anchored Ensembles of Neural Networks, and Application to Exploration in Reinforcement Learning
  Exploration in Reinforcement Learning Workshop, ICML 2018

- **T Pearce**, F Leibfried, M Zaki, A Brintrup, A Neely
  Uncertainty in Neural Networks: Approximately Bayesian Ensembling
  AISTATS 2020

- **T Pearce**, A Brintrup, A Neely
  A Neuroscience-inspired Approach to Building Uncertainty-aware AI
  NAISys 2020

- **T Pearce**, A Y K Foong, A Brintrup
  Structured Weight Priors for Convolutional Neural Networks
  Uncertainty and Robustness in Deep Learning Workshop, ICML 2020

Chapter 2

Background

TLDR: Prerequisite reading for the thesis; uncertainty, neural nets, Bayesian neural nets, Gaussian processes.

This chapter introduces elements important to the broad theme of this thesis: modelling uncertainty with NNs. This is prerequisite knowledge for understanding the contributions presented later in the thesis.

- Section 2.1 - An explanation of the types of uncertainty typically defined in statistical modelling, and how they arise in regression and classification.
- Section 2.2 - Introduces the core model of this thesis: NNs. This is done through a biological lens, which sets up the closing discussion in chapter 6.
- Section 2.3 - Briefly reviews several existing methods which allow NNs to model uncertainty. Chapter 5 will contribute a new method.
- Section 2.4 - A more detailed look at one of the leading methods for modelling uncertainty in NNs: Bayesian NNs. Chapters 3 to 4 contribute in this area.
- Section 2.5 & 2.6 - GPs are naturally able to capture uncertainty. We review this model and describe connections to BNNs. Chapter 3's contributions build on this.

This chapter serves as both a technical primer and macro-literature review with key literature sources being cited. Each technical chapter (3, 4, 5) further contains a focused literature review relevant to the specific contribution.

2.1 Anatomy of Uncertainty

Consider building a model to predict a person's height given their age, sex and race, using a dataset of thousands of people living in a small town in Cambridgeshire, UK. The model is then asked to make height predictions for two people.

1. A Caucasian male of 43 years.
2. An aboriginal Taiwanese male of 21 years.

Based on the dataset, the height range for the first might be given as falling between 5'8" to 6'2". The second one is more difficult, since the dataset may not contain any instances of aboriginal Taiwanese. To be cautious, the model might want to output a greater range of 5'6" to 6'5".

The range on the two answers communicates that there is uncertainty in both, but the **type of uncertainty** is different. The first is uncertainty due to factors outside of one's model of the task (**'aleatoric uncertainty'**), the second is additionally uncertain due to lack of knowledge (**'epistemic uncertainty'**) (Der Kiureghian and Ditlevsen, 2008).

In general, epistemic uncertainty can be reduced by collecting more data on a task (here, one could collect heights of Taiwanese natives). Aleatoric uncertainty cannot, though it might be reduced by collecting additional information about new variables (here perhaps the weight of the individual).

It's important to note that this aleatoric/epistemic distinction is a consequence of modelling choices, and is not some *absolute* property of the task. For example, if instead of using a dataset collected in the Cambridgeshire town, it was collected in Taitung county, the epistemic assignation would be reversed.

A brief philosophical note
Typically, the uncertainty on the outcome of a roll of a dice might be thought of as aleatoric. But is it really? The outcome is some predictable function of release angles, momentum, friction coefficients and so on. In which case why not call it epistemic? Hence, we argue it is more of a modelling choice. Thinking more deeply about the difference between aleatoric and epistemic uncertainty can lead to interesting questions: Is the world inherently stochastic? Does randomness truly exist?

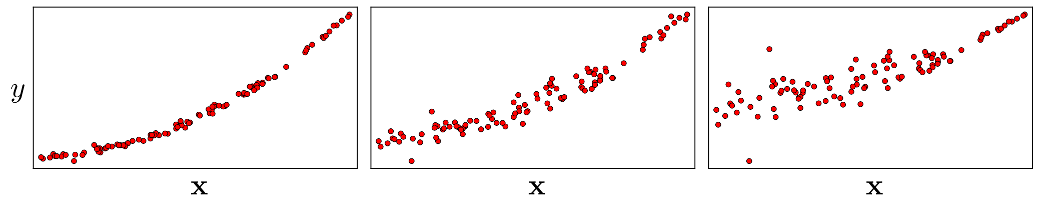


Figure 2.1 Visualising aleatoric uncertainty. Even with a large amount of data, uncertainty may remain. Left: Low homoskedastic noise. Middle: High homoskedastic noise. Right: Heteroskedastic noise.

2.1.1 Regression

We now consider, more formally, these two sources of uncertainty in regression tasks, where we predict some scalar, $y \in \mathbb{R}$, based on some input $\mathbf{x} \in \mathbb{R}^d$. Generally it is not possible to *perfectly* predict the value y, since the input features do not contain every detail required. As such, we often assume a model with additive noise, ϵ,

$$y = f(\mathbf{x}) + \epsilon. \tag{2.1}$$

Following from above, y might represent a person's height, whilst $\mathbf{x}$ is information about their age, sex and race. Knowing these $\mathbf{x}$ variables is not enough to perfectly predict a person's height, hence we model this unknown with a random variable, ϵ. This represents the **aleatoric** uncertainty.

Homoskedastic vs. Heteroskedastic Aleatoric Uncertainty

A simple, practical choice for this random variable is, $\epsilon \sim \mathcal{N}(0, \sigma_\epsilon^2)$, where σ_ϵ^2 is a parameter to be estimated. This implies that ϵ does not depend on $\mathbf{x}$, which is known as **homoskedastic** noise. Alternatively, one could define and learn a new function that is dependent on the data, $f_\epsilon(\mathbf{x})$, which leads to **heteroskedastic** noise, $\epsilon \sim \mathcal{N}(0, f_\epsilon(\mathbf{x}))$. An example of each is visualised in figure 2.1.

Epistemic Uncertainty

We now consider epistemic uncertainty. Our ultimate goal is to capture the uncertainty of y in eq. 2.1. This uncertainty can be quantified as its variance. If ϵ is assumed independent of

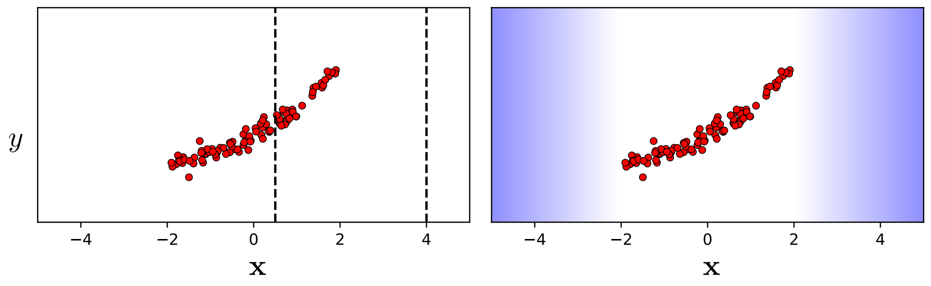


Figure 2.2 Visualising epistemic uncertainty. Left: A prediction of y when $\mathbf{x} = 0.5$ should be more certain than when $\mathbf{x} = 4.0$. Right: Uncertainty in $f(\mathbf{x})$ increases further from the training data.

$\mathbf{x}$ and y, we have,

$$\begin{aligned} y &= f(\mathbf{x}) + \epsilon \\ \mathbb{V}\text{ar}[y] &= \mathbb{V}\text{ar}[f(\mathbf{x}) + \epsilon] \\ &= \mathbb{V}\text{ar}[f(\mathbf{x})] + \mathbb{V}\text{ar}[\epsilon] \\ \text{Total Uncertainty} &= \text{Epistemic} + \text{Aleatoric}. \end{aligned} \tag{2.2}$$

As discussed, $\mathbb{V}\text{ar}[\epsilon]$ is estimated as part of the model - by learning $f_\epsilon(\mathbf{x})$ or σ_ϵ^2. From eq. 2.2 we see we also need to estimate $\mathbb{V}\text{ar}[f(\mathbf{x})]$, which is the estimated variance of the function. Indeed, this uncertainty over $f(\mathbf{x})$ is our epistemic uncertainty. Typically a model might learn $f(\mathbf{x})$ for some region of the input space covered by the training data, but there will be regions outside of that which it will be uncertain about. This is visualised in figure 2.2.

2.1.2 Classification

The previous two types of uncertainty also apply to classification tasks. Consider a binary classification task, with two possible classes, $y \in \{c_1, c_2\}$. Figure 2.3 shows a simple 1-D set up. When $\mathbf{x} = 0$, we can confidently say $P(y = 1) \approx 0.0$. When $\mathbf{x} = -1.2$, we can equally say $P(y = 1) \approx 0.5$. Although we are unsure of the class prediction here, we are quite sure that it is 0.5. This uncertainty arises from inherent noise in the data and is aleatoric. On the other hand, if asked to predict at $\mathbf{x} = -4$, we might again say $P(y = 1) \approx 0.5$, but this time uncertainty derives from having no data in this region of the input space, and hence little knowledge about the underlying function, so our uncertainty and is epistemic.

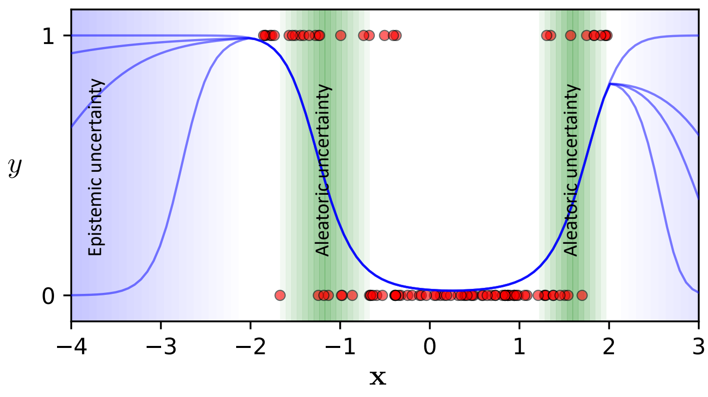


Figure 2.3 1-D binary classification task showing two sources of uncertainty. Blue lines show plausible estimates for $\hat{P}(y = c_1|\mathbf{x})$. The multiple lines arise from uncertainty in $f(\mathbf{x})$.

2.1.3 Other Uncertainties

Above, we have introduced the two sources of uncertainty we will model through this thesis. Care must be taken as these sources appear under various names. Aleatoric uncertainty may be referred to as irreducible noise, data uncertainty, data noise, or known unknowns. Epistemic uncertainty as model uncertainty, or parameter uncertainty.

More confusingly, sometimes the two sources are decomposed or slightly redefined. For example, Lakshminarayanan et al. (2017) talk about calibration (which is related to aleatoric uncertainty) and domain shift / out-of-distribution (OOD) uncertainty (which is related to epistemic uncertainty). Malinin & Gales (2018) explicitly introduce a further source: distributional uncertainty, which allows their proposed model to distinctly model aleatoric and OOD uncertainty, both in a maximum likelihood fashion. They note that this is implicitly covered by epistemic uncertainty under the Bayesian framework.

Finally we note that other types of uncertainty exist that are not considered in this thesis. For example, input uncertainty, where there is uncertainty on the input values.

2.2 Neural Networks

This section provides a primer on NNs. Historically, there have been two broad approaches to building AI systems.

1. Symbolic AI (also ‘expert systems’, ‘Good Old-Fashioned Artificial Intelligence’) takes the view that thought is the manipulation of symbols (words, concepts). Encoding

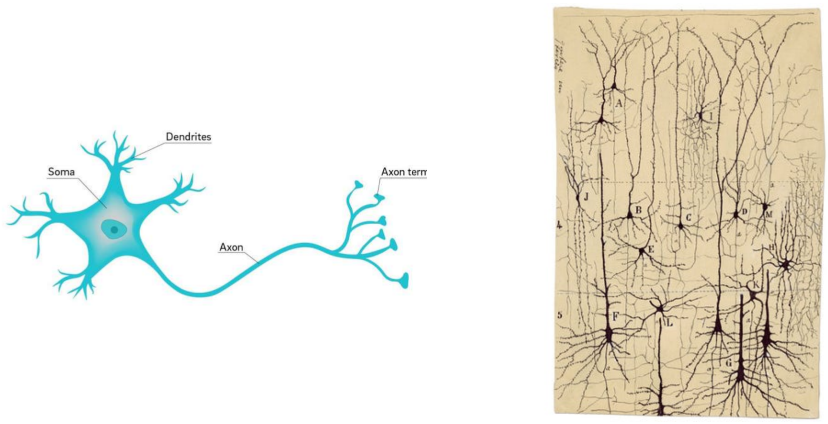


Figure 2.4 Left: Modern diagram of a single neuron. Right: Early sketches of neuons based on thin slices of brain tissue by Ramón y Cajal.

knowledge into a system, operated on by the rules of logic, may lead to an intelligent system.

2. Connectionism takes an alternative view; that intelligence might emerge through simulating the hardware of a biologically intelligent system.

This second view was the original motivation for building artificial NNs. Whilst today the field of machine learning is quite separate from these biological origins, we use the analogy to provide an intuitive introduction to NNs. This connection also motivates our discussion in section 6.

2.2.1 Origins: Modelling Biological Neurons

In 1888, Ramón y Cajal published evidence that the brain consisted of networks of individual neurons (figure 2.4) (López-Muñoz et al., 2006). This marked the birth of the idea that networks of neurons provide the basis of intelligence.

The first major work on the computational modelling of these networks was published in 1943 by McCulloch & Pitts (1943), when it was put forward that neurons with binary activation functions could do first order logic.

> The nervous system is a net of neurons, each having a soma and an axon. Their ... synapses are always between the axon of one neuron and the soma of another. At any instant a neuron has some threshold, which excitation must exceed to

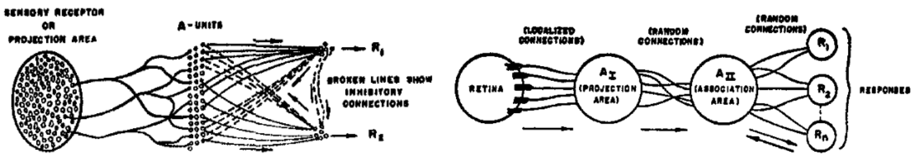


Figure 2.5 Diagram of an early perceptron by Rosenblatt (1958).

initiate an impulse... From the point of excitation the impulse is propagated to all parts of the neuron.

McCulloch & Pitts (1943)

The idea of synaptic weights being the basis of learning was then developed by Hebb (1950), captured by the well known phrase: **cells that fire together wire together**. In 1958, Rosenblatt combined this idea with that of McCulloch & Pitts, proposing a computational model of a neuron with learnable weights (Rosenblatt, 1958). This is perhaps the earliest recognisable relative of modern day NNs (figure 2.5).

2.2.2 From Neurons to Deep Neural Networks

Neurons

Figure 2.6 (left) shows a diagram of a biological neuron, with x_i representing input signals into the neuron, w_i the synaptic weights, b some bias it must overcome in order to fire, and $f(\mathbf{x})$ the output. On the right the figure shows the computational graph of an artificial neuron.

Biological neurons output *spikes* of action potential which vary over the temporal dimension. Conversely, artificial neurons output a continuous value and do not operate over a temporal dimension. The most plausible way to connect these two models is to interpret artificial neurons as modelling the *rate* at which spikes are being fired (firing rate) in the biological neuron given some steady stimulus (Pfeiffer and Pfeil, 2018).

The equation governing the output of an artificial neuron is given by,

$$f(\mathbf{x}) = \psi\left(\sum_{i=1}^{H} w_i x_i + b\right) \tag{2.3}$$

$$= \psi(\mathbf{w}^\top \mathbf{x} + b) \tag{2.4}$$

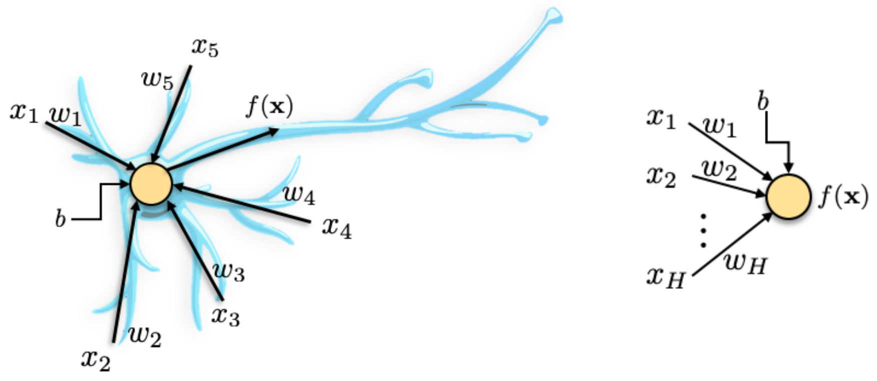


Figure 2.6 Diagram of a single neuron. Left: Overlaid on biological neuron. Right: Abstracted computational graph.

for some activation function, ψ, input $\mathbf{x} \in \mathbb{R}^H$, weight vector, $\mathbf{w} \in \mathbb{R}^H$, bias $b \in \mathbb{R}$, and output $f(\mathbf{x}) \in \mathbb{R}$. This thesis will tend to use vector and matrix notation in eq. 2.4.

It might be hard to see how eq. 2.4 will be of interest. Whilst each neuron carries out a simple operation, by connecting these neurons into large structured networks, powerful computational mappings can be built.

Single Layer Neural Networks

We now build a simple single layer NN composed of artificial neurons. Figure 2.7 shows a NN with a single input and output neuron and H hidden neurons.

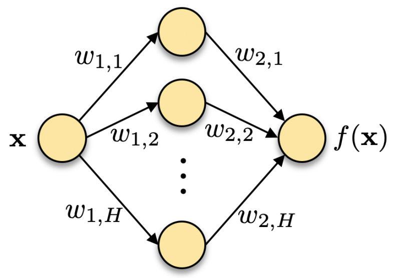


Figure 2.7 Single-layer NN (biases not drawn for clarity).

The output of the NN with a general number of inputs (d) and outputs (d_{out}) is computed,

$$f(\mathbf{x}) = \mathbf{b}_2 + \mathbf{W}_2\psi(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1), \tag{2.5}$$

where $\mathbf{W}_1 \in \mathbb{R}^{H\times d}$, and $\mathbf{b}_1 \in \mathbb{R}^H$, whilst $\mathbf{W}_2 \in \mathbb{R}^{d_{out}\times H}$ and $\mathbf{b}_2 \in \mathbb{R}^{d_{out}}$. The activation, ψ, is applied pointwise.

Deep Neural Networks

If wide enough, single layer NNs are able to approximate any function (Debao, 1993). However, it's been argued that adding layers, rather than width, produces models better suited to the types of task typically encountered (Bengio, 2009; Poggio et al., 2017). The arguments are non-trivial, describing depth as important to learn compositional, or highly-varying functions, which it is argued are properties of vision and language. The empirical success of deep models provides evidence for this, and it is from their effectiveness that the term 'deep learning' emerged (Goodfellow et al., 2016). Figure 2.8 shows a NN with three hidden layers. Computations are as before, but now must be performed recursively as in eq. 2.6.

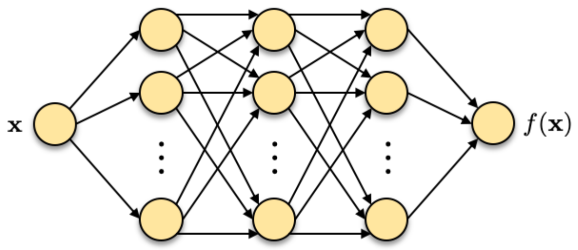


Figure 2.8 Deep NN with three hidden layers.

$$f^L(\mathbf{x}) = \mathbf{b}_{L-1} + \mathbf{W}_{L-1}\psi_{L-1}\big(f^{L-1}(\mathbf{x})\big) \tag{2.6}$$

Where the dimensions of $\mathbf{W}_{L-1}$ and $\mathbf{b}_{L-1}$ depend on the number of neurons in the adjacent layers.

Complex Architectures

So far we have considered fully-connected NNs, where neurons are organised into layers, and each neuron in a layer is connected to every neuron in the adjacent layers. This is a somewhat arbitrary choice of connectivity, and a variety of alternatives exist. Figure 2.9 contains illustrations of three: autoencoders, recurrent NNs, and convolutional NNs. Much of deep learning's success derives from using architectures suited to a task (often termed 'structural priors') (Goodfellow et al., 2016).

Amongst the most important of these architectures, and of most relevance to later experiments in this thesis, is the convolutional NN (CNN), which can be viewed as a sparsely-connected NN with weight sharing. This encourages translational equivariance - a feature produces similar output regardless of it's local position in the input - and makes it particularly suitable

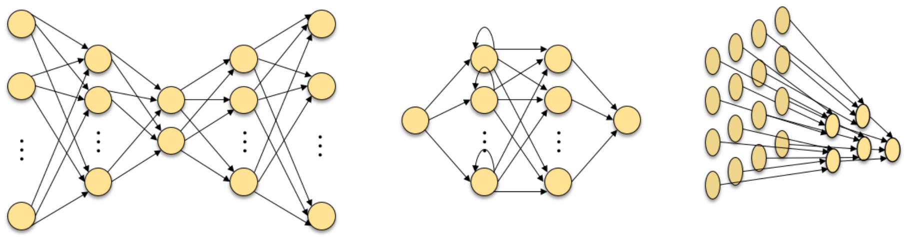


Figure 2.9 NNs with various connectivity patterns. Left: Autoencoder. Middle: Recurrent NN. Right: CNN.

for data with a spatial or temporal aspect, such as images. The CNN was originally designed to mimic the wiring pattern in human visual systems (Hassabis et al., 2017).

2.2.3 Learning in NNs

Learning in biological intelligence involves modifying synaptic weights of a biological NN. Until now, we have discussed only the structure and computations carried out by artificial NNs in order to get from input to output, and assumed the weights and biases (parameters) are given. Goodfellow et al. (2016) provides a reference for this section.

In order for a NN to be useful, it is necessary to specify parameter values that perform a desired input-output mapping. The process through which these parameters are specified is known as learning or training. One of the major challenges in NNs has been developing effective methods for doing this.

Generally, it is framed as an optimisation problem. A cost function (also 'loss function' or 'objective function') is defined based on the desired outcome of the task. In a supervised regression setting, this might be,

$$\mathcal{L}_\theta(\theta, \mathbf{x}, \mathbf{y}) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2, \tag{2.7}$$

where $\hat{y}_i = f(\mathbf{x}_i, \theta)$ is the output of the NN, θ are the NN parameters, and $\mathbf{x}_i$ and y_i refers to the i^{th} input and output respectively.

The optimisation problem is to minimise the cost function by adjusting the parameters. This is commonly done using gradient descent, which uses the partial derivative, $\frac{\partial \mathcal{L}}{\partial \theta_i}$, i.e. how the loss changes given a (infinitesimally) small increase in θ_i. The update rule is then,

$$\theta_{i,t+1} = \theta_{i,t} - \tau \frac{\partial \mathcal{L}}{\partial \theta_{i,t}}, \tag{2.8}$$

at learning step t, for some learning rate, τ, controlling the size of the update. Rather than computing the gradient over the whole dataset, in practice it is often computed over some subset ('mini-batch') of the data. This introduces some noise to the update step, hence the learning process is termed stochastic gradient descent (SGD). In order to compute the term $\frac{\partial \mathcal{L}}{\partial \theta_i}$ for all parameters in the NN, backpropogation is used, which applies the chain rule successively backwards through the NN. In order to calculate the derivatives, the loss function and NN operations must be differentiable.

Eq. 2.8 gives the vanilla gradient descent update rule. Modern NNs often use variants such as Adam (Kingma and Ba, 2015), which stores a moving average of the gradient mean and variance, and allows more effective weight updates to be made.

2.3 Modelling Uncertainty with Neural Networks

Having introduced uncertainty in section 2.1 and NNs in section 2.2, this section details the ability of standard NNs to model uncertainty. They are found lacking in certain areas, and the section proceeds by summarising existing methods and modifications that can improve their ability to model uncertainty.

A new method is contributed in chapter 5, and improvements to one of the most promising approaches are made in chapters 3 and 4.

2.3.1 Can Standard Neural Networks Model Uncertainty?

Can standard NNs (defined as in section 2.2) used for regression and classification tasks communicate an appropriate level of epistemic and aleatoric uncertainty relating to a particular prediction? Table 2.1 summarises the answer.

By using a NN to parameterise a probability distribution, aleatoric uncertainty can be captured. For a classification NN, outputs can be squashed through a softmax, creating a final output that can then be interpreted as a multinomial distribution. A regression NN with one output could represent the mean of a normal distribution, with constant variance, capable of capturing only homoskedastic noise. A second output could be added to represent an

Table 2.1 Ability of standard NNs to quantify uncertainty types. For regression, it's assumed prediction is of a single scalar value.

	Classification NN	Regression NN (one output)	Regression NN (two outputs)
Aleatoric (Homoskedastic)	✓	✓	✓
Aleatoric (Heteroskedastic)	✓	✗	✓
Epistemic	✗	✗	✗

input-dependent variance, which would capture heteroskedastic noise. (Note that the loss function must be matched to the architecture.) These set ups are further detailed in section 2.4.1.

When asked to predict on a data point unlike the training data, the NN should increase its uncertainty. There is no mechanism built into standard NNs to do this. Even in the case of a two-output regression NN, there is no reason for the estimated variance to increase for a new data point. As such, standard NNs cannot estimate epistemic uncertainty. This is unfortunate since epistemic uncertainty is often the more useful component of the two; it is required for many use cases described in chapter 1, such as active learning, exploration in RL, and knowing when an unusual input is received.

2.3.2 Overview of Methods

This section briefly introduces some common methods used to improve uncertainty estimates of NNs.

Quantile Regression

Eq. 2.7 details a loss function that trains a NN to estimate the mean of a scalar target. Using instead, $\sum |y_i - \hat{y}_i|$, encourages output of the median. Further modifying to, $\tau \sum_{\forall y_i < \hat{y}_i}(y_i - \hat{y}_i) + (\tau - 1)\sum_{\forall y_i \geq \hat{y}_i}(y_i - \hat{y}_i)$, encourages return of some quantile (the τ percentile) of the predictive distribution (Koenker and Hallock, 2001; Taylor, 2000). Note that no distributional form has been assumed. By estimating, say the 0.025 and 0.975 quantiles, a 95% PI could be constructed. Quantile regression is only able to capture aleatoric uncertainty.

Conformal Prediction

This method uses previously seen data to determine prediction regions (e.g. a prediction interval) that provide coverage with some pre-set probability (Shafer and Vovk, 2008). The size of the region indicates the uncertainty. Usefully, it is compatible with any predictive model, including NNs. It is difficult to determine the link with aleatoric and epistemic uncertainty (Hüllermeier and Waegeman, 2019).

Ensembling

A simple approach to estimating uncertainty is to train a small number of NNs (an ensemble), beginning from different initialisations and sometimes on noisy versions of the training data. When asked to make predictions on new data, there will be some diversity in the ensemble's predictions, which can be interpreted as the uncertainty. The intuition is simple: if the new data point is similar to the training data, the NNs should all produce similar estimates, but if it is very different, there should be higher variance in the predictions.

This is generally presented as a non-Bayesian alternative to epistemic uncertainty estimation (Lakshminarayanan et al., 2017). Chapter 4 discusses this claim more thoroughly, and presents a modification that does lead to strong connections to Bayesian methodology.

Prior Networks

This recently proposed approach explicitly trains a NN to be uncertain on OOD data (Malinin and Gales, 2018). These OOD samples are generated either through adversarial techniques, or by using alternative datasets (e.g. train on CIFAR and use SVHN as OOD examples). This is most useful when these samples are representative of OOD data that will be encountered in the real world. The result is a single NN with good resistance to OOD examples. On the other hand, it may not be straightforward to extend the method to use cases of BNNs such as exploration and active learning, where the training distribution is incrementally growing. Prior networks can capture both aleatoric and epistemic uncertainty.

Bayesian Neural Networks

The Bayesian framework provides a principled framework for handling uncertainty, and has been combined with numerous models and applications. At its simplest, it specifies how to update beliefs (e.g. about functions) in light of evidence (e.g. data points), in an optimal fashion. It was first applied to NNs in 1992 (MacKay, 1992b), motivated as a principled way to do hyperparameter selection as well as for its ability to capture epistemic uncertainty. A large body of work has followed, more recently under the name ‘Bayesian deep learning’. It is arguably the most prominent approach to handling uncertainty in NNs (as an example, the Baysian Deep Learning workshop has attracted around 100 abstracts in recent years at the NeurIPS conference, bayesiandeeplearning.org). The contributions of this thesis are primarily in this area, as such the following section fully introduces these models.

2.4 Making Neural Networks Bayesian

2.4.1 Distributions Over Parameters

Earlier we introduced weights of a NN as scalar numerical values, that represent the strength of a connection between two neurons, and can be learnt using gradient descent.

A Bayesian NN requires a conceptually different set up. Rather than assigning each parameter a specific numerical value, we will instead model them as random variables with a probability distribution. This could either be a parametric probability distribution (such as normal or uniform), or an empirical probability distribution (by storing a set of parameter samples). Figure 2.10 illustrates this. Note that probability distributions are shown as though parameters are independent of each other, though this need not be the case.

Probability distributions in general are used to describe the uncertainty over the value of some variable, and this is no exception: we use them to represent the uncertainty in parameters.

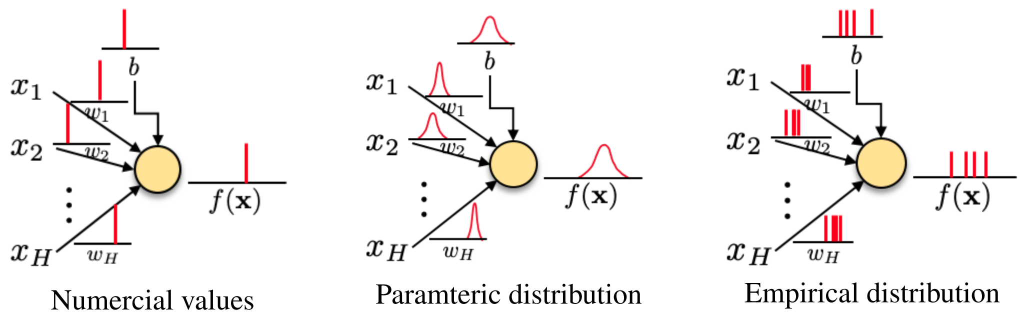


Figure 2.10 A neuron with point estimates over parameters produces a point estimate at its output. If parameters are modelled with distributions (parametric or empirical), the output will also be a distribution.

Although relatively straightforward conceptually, this new set up has implications both for computing the feedforward output (how should the output of the BNN now be computed?), and more challenging, the learning process (how can gradient descent be applied to distributions?).

Bayesian Posterior Distribution

Assigning arbitrary distributions to parameters doesn't make a NN Bayesian. Bayesian NNs mean a NN with a specific distribution over the parameters, this distribution is known as the 'Bayesian posterior', denoted $P(\boldsymbol{\theta}|\mathcal{D})$. Here, $\mathcal{D}$, stands for some dataset, which in supervised learning is a set of input/output pairs, $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), \dots, (\mathbf{x}_n, \mathbf{y}_n)\}$. It is simple to write an equation to calculate this posterior distribution in terms of several other distributions.

$$P(\boldsymbol{\theta}|\mathcal{D}) = \frac{P(\mathcal{D}|\boldsymbol{\theta})P(\boldsymbol{\theta})}{P(\mathcal{D})} \tag{2.9}$$

Note that we have slightly abused notation here, and more precisely can be written, $P(\mathcal{D}|\boldsymbol{\theta}) := P(\{\mathbf{y}_i\}_{i=1}^n|\boldsymbol{\theta}, \{\mathbf{x}_i\}_{i=1}^n)$, and, $P(\mathcal{D}) := P(\{\mathbf{y}_i\}_{i=1}^n|\{\mathbf{x}_i\}_{i=1}^n)$.

Likelihood

On the RHS of eq. 2.9 is, $P(\mathcal{D}|\boldsymbol{\theta})$, called the 'likelihood', which represents how likely the data is given the parameters. It returns a single scalar representing the probability of drawing that dataset, given the model parameters.

In order to formulate the likelihood function some probability distribution is specified that is parameterised by the output of a NN. For a regression task a NN with a single output might specify the mean of a normal distribution, with constant variance. If data points are treated independent and identically distributed (iid), the likelihood is then given by,

$$P(\mathcal{D}|\boldsymbol{\theta}) = \prod_{i=1}^{N} \mathcal{N}(\mathbf{y}_i|f(\mathbf{x}_i, \boldsymbol{\theta}), \sigma_\epsilon^2), \tag{2.10}$$

where the NN prediction has been written, $f(\mathbf{x}, \boldsymbol{\theta})$, to explicitly depend on the parameters, and some amount of noise has been assumed in the data, of variance σ_ϵ^2. This is appropriate if the data noise is homoskedastic (section 2.1.1).

If there is reason to believe the data noise is heteroskedastic (section 2.1.1), one might use a NN with two outputs, one specifying the mean and one the variance (or log variance to ensure it's positive). The likelihood is then,

$$P(\mathcal{D}|\boldsymbol{\theta}) = \prod_{i=1}^{N} \mathcal{N}(\mathbf{y}_i | f_1(\mathbf{x}_i,\boldsymbol{\theta}), f_2(\mathbf{x}_i,\boldsymbol{\theta})). \tag{2.11}$$

For classification tasks, with C classes, and one-hot-encoded labels $y_{i,j} \in \{0,1\}$, it's usual to have a NN with C outputs, which are passed through a softmax function in order to convert into a normalised probability (each output between 0 and 1, together sum to 1). Denoting $f_j(\mathbf{x}_i,\boldsymbol{\theta})$ as output j after the softmax, the likelihood is given by a multinomial distribution,

$$P(\mathcal{D}|\boldsymbol{\theta}) \propto \prod_{i=1}^{N} \prod_{j=1}^{C} f_j(\mathbf{x}_i,\boldsymbol{\theta})^{y_{i,j}}. \tag{2.12}$$

This covers the likelihood functions most commonly implemented with BNNs.

Prior

Also in eq. 2.9, we have, $P(\boldsymbol{\theta})$, which is termed the 'prior'. Broadly speaking, a modeller sets this distribution based on prior knowledge available about the task.

Textbook introductions to Bayes might give examples about coin flipping, where a parameter represents the probability of a coin being biased. The prior encodes an initial belief about this parameter, which is updated by observed data. NNs introduce a complication; it is not obvious what a prior distribution over a NN parameter means about the implied prediction, which is what a modeller might have knowledge about.

It is common (though by no means necessary, e.g. (Nalisnick, 2018)) for BNNs to have iid Gaussian priors, of constant mean and variance across a layer, which leads to interesting connections with GPs. This connection forms an important foundation for later contributions, and is described in section 2.6. This will allow an interpretation of parameter priors as functional priors.

A prior is generally set before observing evidence, however empirical Bayes describes a paradigm whereby hyperparameters controlling the prior are set to be their most likely values according to the data. For example, in a GP this could be the kernel length scale, or in BNNs, the variance of the weight priors for a particular layer. Note that this philosophy faces some criticism (Blundell et al., 2015; Gelman, 2008).

Marginal Likelihood

The final term on the RHS of eq. 2.9 is $P(\mathcal{D})$, known as the 'marginal likelihood' or 'evidence' or 'prior predictive distribution', calculated by,

$$P(\mathcal{D}) = \int P(\mathcal{D}|\boldsymbol{\theta})P(\boldsymbol{\theta})d\boldsymbol{\theta}. \tag{2.13}$$

A simple interpretation is that it is a normalising constant, ensuring that $P(\mathcal{D}|\boldsymbol{\theta})P(\boldsymbol{\theta})$ sums to one over the domain of $\boldsymbol{\theta}$, which is a requirement of a probability distribution.

It is also useful as a measure of how well a model fits the data (Rasmussen and Williams, 2006). This can be valuable in empirical Bayes procedures where hyperparameters of the prior are to be estimated. Examples of this are provided for a GP in figure 2.14.

2.4.2 Posterior Predictive Distribution

In models where parameters themselves are of interest (as in the coin flipping example), it might be that generating their posterior probability distribution, $P(\boldsymbol{\theta}|\mathcal{D})$, is the goal of Bayesian inference. By contrast, in NNs we are less interested in the parameter distributions themselves, and more interested in making predictions using the posterior distribution $P(\boldsymbol{\theta}|\mathcal{D})$ for some new data point, $\mathbf{x}^*$. This leads to the posterior predictive distribution, $P(\mathbf{y}^*|\mathbf{x}^*,\mathcal{D})$.

$$P(\mathbf{y}^*|\mathbf{x}^*,\mathcal{D}) = \int P(\mathbf{y}^*|\mathbf{x}^*,\boldsymbol{\theta})P(\boldsymbol{\theta}|\mathcal{D})d\boldsymbol{\theta} \tag{2.14}$$

One insight used in later contributions (chapter 4) is that provided samples can be generated from the posterior, it may not be necessary to fully capture the parameter posterior itself.

2.4.3 Learning in BNNs

In section 2.4.3, we described a learning mechanism, SGD, which is widely used for NNs with point estimates of the parameters and a defined loss function. One of the major challenges of BNNs is that we can no longer use this mechanism directly. The objective of BNNs is to find the posterior distribution; but there is no obvious loss function that can be minimised, nor is it clear how the parameter update rule in eq. 2.8 would apply to distributions.

Computing the posterior analytically is possible only in special models where the prior and likelihood distributions are conjugate. This occurs, for example, with a beta prior and

Bernoulli likelihood, which apply in the coin flipping example. For most models, including BNNs, eq. 2.9 is not tractable.

We now briefly introduce the main classes of techniques used to perform Bayesian inference in NNs. Note that this is an vast, active area of research and we attempt to summarise only the main approaches.

One of the disadvantages of many of these methods is that they fall outside common frameworks used for training NNs used by the majority of the community. Indeed one method that doesn't, MC Dropout, is (arguably) the most widely used because of this. Our proposal in chapter 4 provides a similarly easy-to-implement method.

Sampling Methods - MCMC, HMC

Markov chain Monte Carlo (MCMC) methods provide a procedure for sampling from a probability distribution. Stochastic chains explore the parameter space. They are designed so that the stationary distribution of the chain matches the target probability distribution. Hence one can obtain a set of parameter samples forming an empirical distribution.

MCMC methods can be implemented with a variety of algorithms. Metropolis–Hastings works well in smalls-scale settings but can be slow to converge for more complex scenarios. Hamiltonian Monte Carlo (HMC) uses gradient information to move around the parameter space more efficiently. It was shown to be better suited to use in BNNs (Neal, 1997), and more recently has been made more practical (Chen et al., 2014).

Variational Inference

Variational inference (VI) provides an alternative paradigm to sampling methods. A parametric approximating distribution is defined, $q_\nu(\boldsymbol{\theta})$, with the goal of finding distribution parameters, ν, to minimise the Kullback–Leibler divergence between the approximating distribution and the true posterior, $\text{argmin}_\nu\, D_{\text{KL}}(q_\nu(\boldsymbol{\theta})||p(\boldsymbol{\theta}|\mathcal{D}))$. This optimisation is done by maximising a surrogate quantity called the ELBO (evidence lower bound), given by,

$$\text{ELBO} = \int q_\nu(\boldsymbol{\theta}) \log p(\mathcal{D}|\boldsymbol{\theta}) d\boldsymbol{\theta} + \text{KL}\left(q_\nu(\boldsymbol{\theta})||p(\boldsymbol{\theta})\right) \tag{2.15}$$

The major attraction of the method is precisely that it *can* be framed as an optimisation problem, which opens the door to widely used tools already available. A disadvantage is that the the quality of approximation of the posterior is limited by both how restrictive the

q distribution is, and how successful the optimisation is. Both can be difficult to check in practice (Yao et al., 2018).

VI for BNNs has generally been proposed with a 'mean-field' assumption to simplify the optimisation problem. That is, parameters in the posterior are assumed uncorrelated. The suitability of this assumption in BNNs is discussed in chapter 4.

Many methods proposed for BNNs leverage the VI framework (Barber et al., 1998; Blundell et al., 2015; Khan et al., 2018; MacKay, 1995; **?**; **?**).

One technique worth elaboration due to its popularity is MC Dropout. Dropout (Srivastava et al., 2014) is a stochastic regularisation technique where the outputs of hidden nodes are randomly set to zero with some probability. This can be an effective way to reduce overfitting. Originally it was proposed to be used only in the training phase, with the randomness turned off during prediction. However if the randomness is left on during prediction, a distribution is produced by running multiple forward passes. It was proposed that the variance of this distribution could be interpreted as epistemic uncertainty, and this was connected to VI with Bernoulli approximating q distributions (Gal and Ghahramani, 2015), though the connection has been debated (Hron et al., 2018).

Other Inference Methods

There are many other approximation methods. Stein Variational methods (Liu and Wang, 2016) offer a hybrid sampling/optimisation method, similar in spirit to our proposal in chapter 4. The Laplace method (Bishop, 2006; Ritter et al., 2018) first trains a NN to a MAP solution, then fits a multivariate Gaussian as the approximate posterior, with covariance matched to the curvature of the Hessian. Another proposal does analytical inference over the final layer weights only (Azizzadenesheli et al., 2018), questioning whether it's necessary to do Bayesian inference over the entire model.

Whilst not an inference method itself, a final area worthy of mention is probabilistic programming. Several popular frameworks have recently been released, e.g. Pyro, PyMC3, ZhuSuan, providing convenient implementations of the inference methods discussed (and more).

2.5 Gaussian Processes

Having provided an overview of how NNs can be modified to output uncertainty estimates, this section now introduces the Gaussian Process (GP), a Bayesian nonparametric model that

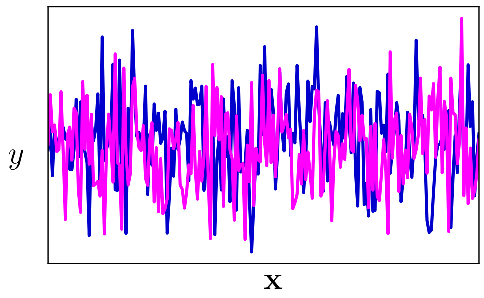


Figure 2.11 Two realisations of a Gaussian white noise process. The x-axis and y-axis labels are dropped in subsequent plots.

naturally handles uncertainty. Whilst a direct comparison is of mild interest, understanding GPs is prerequisite for section 2.6, which will reiterate a profound connection between BNNs and GPs, and provides a unique perspective to study both NNs and BNNs, important to later contributions in chapter 3.

Following is an intuitive introduction to GPs. First we consider how to sample stochastic functions using a Gaussian distribution. This requires specifying functions for the mean and covariance of a multivariate Gaussian. We describe how the covariance function determines properties of the functions being generated, and how multiple covariance functions can be combined. This draws from sources Rasmussen and Williams (2006), and Duvenaud (2014).

2.5.1 Sampling Functions Using a Gaussian Distribution

Consider sampling a function using a Gaussian distribution in the simplest way possible. Choose some value of interest, $x_1 \in \mathbb{R}$, then sample a corresponding value, $y \in \mathbb{R}$, from a Gaussian distribution, $y_1 \sim \mathcal{N}(0, \sigma^2)$. To sample a second data point, at x_2, we could again sample, $y_2 \sim \mathcal{N}(0, \sigma^2)$. Note that y_2 is independent of y_1. This is known as Gaussian white noise. Figure 2.11 visualises two examples of this process, over a grid of x values.

Instead of every point being independent, a more interesting stochastic function might introduce some relationship between them. One way in which two random variables are related can be described by their covariance, $\mathbb{E}[(y_1 - \mathbb{E}(y_1))(y_2 - \mathbb{E}(y_2))]$. Consider a function that takes in two arbitrary data points, x, x', and returns a value for how they covary. Such a function is called a 'covariance function' (or less precisely 'kernel'), denoted $K(x, x')$, where $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. The squared exponential (SE) kernel is a popular choice,

$$K_{\text{SE}}(x, x') = \sigma^2 \exp\left(- \frac{(x - x')^2}{l^2} \right). \tag{2.16}$$

When the two points are very close, the covariance is high, $(x-x')^2 \approx 0 \implies K(x,x') \approx \sigma^2$. And when they are far apart the covariance is weak, $(x-x')^2 \approx \infty \implies K(x,x') \approx 0$. The hyperparameter l, the 'length scale', controls how rapidly this transition happens. σ^2 controls the overall amplitude of the functions.

These stochastic functions are sampled from a subset of the input domain from a multivariate Gaussian distribution, $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, with mean vector, $\boldsymbol{\mu} \in \mathbb{R}^n$, and covariance matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{n \times n}$ (termed 'Gram matrix') of form,

$$\boldsymbol{\Sigma} = \begin{bmatrix} K(x_1,x_1) & K(x_1,x_2) & \dots & K(x_1,x_n) \\ K(x_2,x_1) & K(x_2,x_2) & \dots & K(x_2,x_n) \\ \vdots & \vdots & \ddots & \vdots \\ K(x_1,x_n) & K(x_2,x_n) & \dots & K(x_n,x_n) \end{bmatrix}. \tag{2.17}$$

Often, the mean is set equal to zero, $\boldsymbol{\mu} = \mathbf{0} = \{0,\dots,0\}$. Letting $\mathbf{x} := \{x_1, x_2, \dots, x_n\}$, and $\mathbf{y} := \{y_1, y_2, \dots, y_n\}$, stochastic functions can be sampled according to,

$$\mathbf{y} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}). \tag{2.18}$$

Example prior draws are visualised in figure 2.12 for various values of l.

This corresponds to the formal definition of a GP as a collection of random variables, where every finite subset of random variables has a consistent multivariate Gaussian distribution. Formally, for some constant noise variance, σ_ϵ^2, and kernel function , K,

$$f \sim \text{GP}(\mathbf{0}, K) \tag{2.19}$$

$$\mathbf{y}_i | \mathbf{x}_i \sim \mathcal{N}\left(f(\mathbf{x}_i), \sigma_\epsilon^2\right). \tag{2.20}$$

The SE kernel bakes in a particular assumption about the functions - the covariance value monotonically decreases with the distance between two data points. An alternative assump-

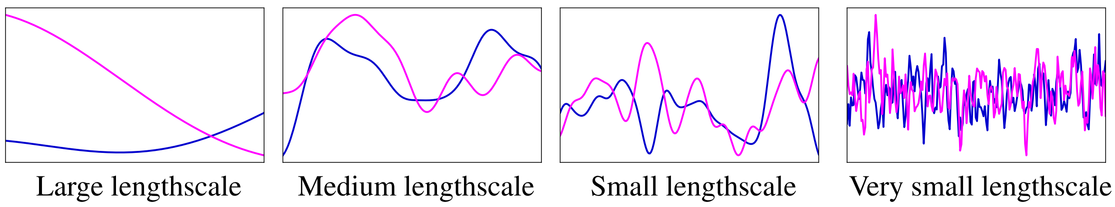


Figure 2.12 Squared exponential kernel, with varying lengthscale.

tion might be that functions repeat over some period, p. One kernel producing this effect is the cosine kernel, $K(x,x') = \sigma^2 \cos\left((x-x')\frac{\pi}{p}\right)$, which leads to pure cosine waves as in figure 2.13. A slightly more interesting kernel also shown in figure 2.13 is given by the exponential sine squared (ESS) (or sometimes simply 'periodic') kernel,

$$K_{\text{ESS}}(x,x') = \sigma^2 \exp\left(-\frac{2\sin^2\left(\frac{\pi}{p}(x-x')\right)}{l^2}\right). \tag{2.21}$$

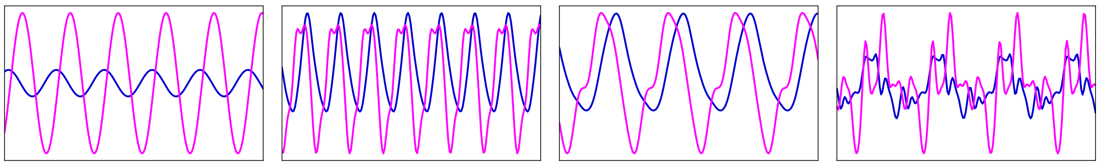


Figure 2.13 Periodic kernels.

2.5.2 Fitting GPs to Data

The previous section showed how to sample arbitrary functions from a GP. Machine learning is often concerned with fitting a function to some dataset, $\mathbf{x}$, $\mathbf{y}$, and making predictions for that function at some new point, $\hat{\mathbf{y}}^*$ at $\mathbf{x}^*$. GPs are valuable in this context, although the mechanism by which they do this differs from that of a parametric model.

A typical deterministic NN makes a single estimate of the function by learning a set of model parameters, $\hat{\boldsymbol{\theta}}$, that fit the training data. The new data point, $\mathbf{x}^*$, is then passed through the NN to produce a prediction, $\hat{\mathbf{y}}^* = f(\mathbf{x}^*, \hat{\boldsymbol{\theta}})$.

A GP differs from this in two ways. Firstly, multiple functions consistent with the data are learnt - more exactly, a probability distribution over all functions is learnt - functions that better fit the training data have higher probability. Secondly, being a non-parametric model, rather than learning parameter values as an intermediate step, prediction is done directly in function space.

GPs naturally combine with the Bayesian framework. The distribution in eq. 2.18 can be set as a prior (here the distribution is over functions rather than parameters).

The joint distribution over training and test points can be written in terms of blocks of training and predicted data points,

$$\begin{bmatrix} \hat{\mathbf{y}}^* \\ \mathbf{y} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix}, \begin{bmatrix} \boldsymbol{\Sigma}_{\mathbf{x}^*,\mathbf{x}^*} & \boldsymbol{\Sigma}_{\mathbf{x},\mathbf{x}^*} \\ \boldsymbol{\Sigma}_{\mathbf{x}^*,\mathbf{x}} & \boldsymbol{\Sigma}_{\mathbf{x},\mathbf{x}} + \sigma_\epsilon^2 \mathbf{I} \end{bmatrix} \right), \tag{2.22}$$

where $\boldsymbol{\Sigma}_{\mathbf{x},\mathbf{x}}$ is the covariance matrix for each data point pair in the training data, as in eq. 2.17, and other matrices are similar (with differing resulting dimensions). A constant level of noise variance has additionally been assumed, σ_ϵ^2.

The prediction goal is to compute $P(\hat{\mathbf{y}}^*|\mathbf{x},\mathbf{y},\mathbf{x}^*)$. For a regression task, if the likelihood is assumed Gaussian (with variance σ_ϵ^2 as above), the predictions are available in closed form, as this simply requires specifying the conditional distribution of a multivariate Gaussian.

$$\hat{\mathbf{y}}^*|\mathbf{y},\mathbf{x},\mathbf{x}^* \sim \mathcal{N}\left(\boldsymbol{\Sigma}_{\mathbf{x}^*,\mathbf{x}} \left(\boldsymbol{\Sigma}_{\mathbf{x},\mathbf{x}} + \sigma_\epsilon^2 \mathbf{I} \right)^{-1} \mathbf{y}, \boldsymbol{\Sigma}_{\mathbf{x}^*,\mathbf{x}^*} - \boldsymbol{\Sigma}_{\mathbf{x},\mathbf{x}^*} \left(\boldsymbol{\Sigma}_{\mathbf{x},\mathbf{x}} + \sigma_\epsilon^2 \mathbf{I} \right)^{-1} \boldsymbol{\Sigma}_{\mathbf{x},\mathbf{x}^*}^\top \right) \tag{2.23}$$

Significantly, this requires the matrix inversion, $\left(\boldsymbol{\Sigma}_{\mathbf{x},\mathbf{x}} + \sigma_\epsilon^2 \mathbf{I} \right)^{-1}$. Given this matrix is of dimension $n \times n$, this inversion takes $\mathcal{O}(n^3)$, which limits the scalability of GPs. Addressing this is an active research area, e.g. (Burt et al., 2019; Csato and Opper, 2002; Quiñonero-Candela and Rasmussen, 2005; Snelson and Ghahramani, 2006; Titsias, 2009).

In this thesis, we only use GPs for regression tasks with Gaussian likelihoods. Approximate methods must be employed to fit to datasets of other sorts, for example classification tasks.

Marginal Likelihood

Eq. 2.13 introduced the marginal likelihood. This quantity can be computed in closed form for GPs (again for regression tasks with a Gaussian likelihood),

$$\log P(\mathbf{y}) = -\frac{1}{2}\mathbf{y}^\top \left(\boldsymbol{\Sigma}_{\mathbf{x},\mathbf{x}} + \sigma_\epsilon^2 \mathbf{I} \right)^{-1} \mathbf{y} - \frac{1}{2}\log|\boldsymbol{\Sigma}_{\mathbf{x},\mathbf{x}} + \sigma_\epsilon^2 \mathbf{I}| - \frac{n}{2}\log 2\pi, \tag{2.24}$$

which provides a useful way for comparing kernels and hyperparameters, discussed in section 2.5.1. Figure 2.14 illustrates this.

2.5.3 Combining Kernels

Whilst there are restrictions on what makes a valid covariance functions (it must be positive semidefinite), one can be surprisingly creative in combining existing kernels to create a new

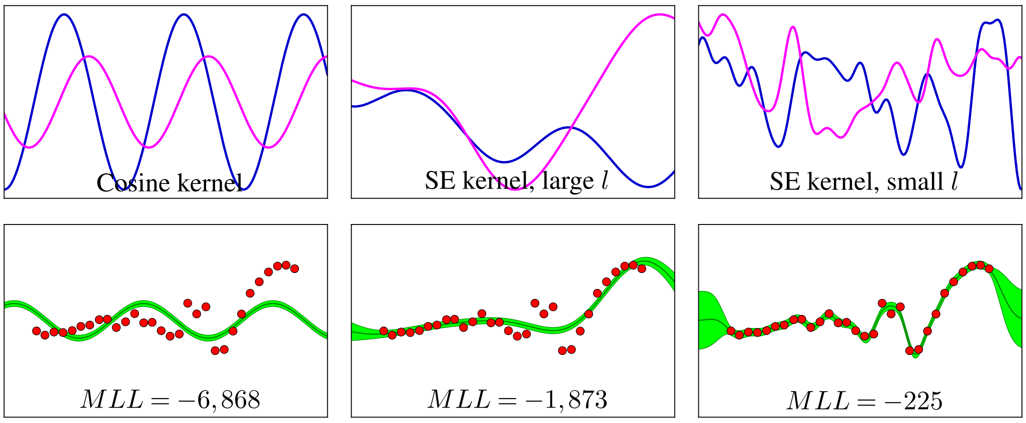


Figure 2.14 Above: Prior draws for three different kernels. Below: Marginal log likelihood (MLL) scores for these GP kernels fitted to the same dataset. The SE kernel with small l produces priors that are visually more similar to the data, which is quantified by the larger MLL score.

kernel. This offers flexibility to design a prior over functions that is suited to a particular dataset of interest. Of interest in this thesis, are addition and multiplication of kernels, as well as applying different kernels to different dimensions of the input.

Adding Kernels

Consider a dataset generated by $f(x) = \sin(x) + x$. The kernels introduced in section 2.5.1 are not a good fit for this, since they either enforce perfect periodicity, or ignore it entirely. Yet a new kernel can be defined as an additive combination of a SE and periodic kernel, $K(x,x') = K_{\text{SE}}(x,x') + K_{\text{ESS}}(x,x')$, which generates functions that do fit the data.

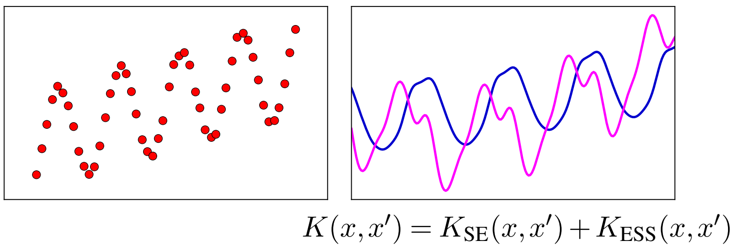


Figure 2.15 Left: Dataset suiting an additive kernel combination. Right: Prior draws of a suitable additive kernel combination.

Multiplying Kernels

Certain datasets suite multiplicative kernels. For example $f(x) = \sin(x) \times x$ is well captured by a new kernel, $K(x,x') = K_{\text{SE}}(x,x') \times K_{\text{ESS}}(x,x')$.

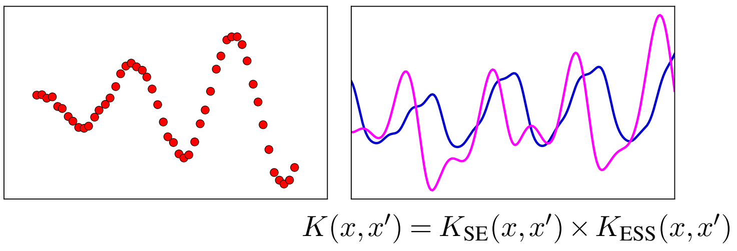


Figure 2.16 Left: Dataset suiting a multiplicative kernel combination. Right: Prior draws of a suitable multiplicative kernel combination.

Different Kernels for Different Dimensions

A further useful technique is to apply different kernels to different input dimensions. For a 2-D input, one could use, $K(\mathbf{x},\mathbf{x}') = K_{\text{SE}}(x_1,x_1') + K_{\text{ESS}}(x_2,x_2')$, if a periodic kernel suited the second dimension, but not the first. Alternatively, one could use the same parametric kernel for each dimension, but with different lengthscales. Figure 2.17 illustrates this.

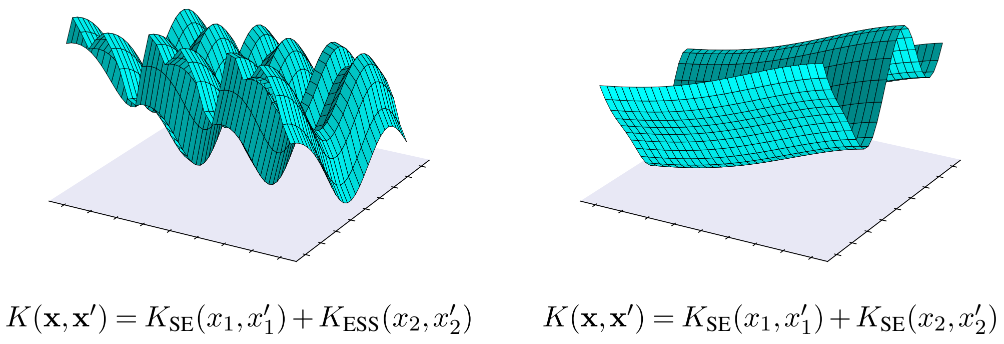


Figure 2.17 Prior draws when different kernels are applied to different dimensions of the input. The right-hand plot uses two SE kernels with differing lengthscales.

2.6 Connections Between BNNs and GPs

On the surface, there is little connecting BNNs (section 2.4) with GPs (section 2.5). BNNs place distributions over parameters, whilst GPs specify kernels which place distributions over functions. This section shows that under certain conditions these two models are actually equivalent. This connection is important from several angles, but most significant to this thesis (in chapter 3), is that it enables one to study how a distribution over the parameters of a BNN translates into a distribution over functions. This is of practical use as it provides a more interpretable space in which to choose priors for a BNN.

This section begins by formally outlining the connection between BNNs and GPs, first observed by Radford Neal (Neal, 1997). Analytical kernels that relate to popular choices of single-layer NN architectures are then listed. Recent work extending the results to other architectures is presented. Finally we consider practical implications of this link.

2.6.1 BNNs Converge to GPs

Here we reproduce the derivation of that an infinitely wide single-layer BNN is a GP (Neal, 1997). Consider a single-layer NN, $f(\mathbf{x}) : \mathbb{R}^d \to \mathbb{R}$, with input, $\mathbf{x}$, weights, $\mathbf{w}_1$ & $\mathbf{w}_2$, biases $\mathbf{b}_1$, activation function, ψ, and hidden units H. To unclutter analysis, the final bias is added later.

$$f(\mathbf{x}) = \sum_{i=1}^{H} w_{2i}\psi(\mathbf{w}_{1i}\mathbf{x} + b_{1i}). \tag{2.25}$$

Begin by considering the mean of this model. Assume an iid prior over the final layer weights centred at zero, $\mathbb{E}[w_{2i}] = 0$, and with variance scaled by width, $\mathbb{V}\text{ar}[w_{2i}] = \sigma_{w2}^2/H$,

$$\begin{aligned}
\mathbb{E}[f(\mathbf{x})] &= \mathbb{E}[\sum_{i=1}^{H} w_{2i}\psi(\mathbf{w}_{1i}\mathbf{x} + b_{1i})] && (2.26)\\
&= \sum_{i=1}^{H} \mathbb{E}[w_{2i}]\mathbb{E}[\psi(\mathbf{w}_{1i}\mathbf{x} + b_{1i})] && (2.27)\\
&= 0. && (2.28)
\end{aligned}$$

Hence the mean function of the resulting GP will be zero.

Consider now the covariance of outputs corresponding to two arbitrary inputs, $\mathbf{x}$ & $\mathbf{x}'$. Denoting for convenience $\psi_i(\mathbf{x}) := \psi(\mathbf{w}_{1i}\mathbf{x} + b_{1i})$,

$$
\begin{aligned}
K(\mathbf{x},\mathbf{x}') &= \mathbb{E}[(f(\mathbf{x}) - \mathbb{E}[f(\mathbf{x})])(f(\mathbf{x}') - \mathbb{E}[f(\mathbf{x}')])] && (2.29)\\
&= \mathbb{E}[f(\mathbf{x})f(\mathbf{x}')] && (2.30)\\
&= \mathbb{E}\left[\left(\sum_{i=1}^{H} w_{2i}\psi_i(\mathbf{x})\right)\left(\sum_{j=1}^{H} w_{2j}\psi_j(\mathbf{x}')\right)\right] && (2.31)
\end{aligned}
$$

$$
\begin{aligned}
= \mathbb{E}\big[w_{2,1}\psi_1(\mathbf{x})w_{2,1}\psi_1(\mathbf{x}') + w_{2,1}\psi_1(\mathbf{x})w_{2,2}\psi_2(\mathbf{x}') + \cdots \\
w_{2,2}\psi_2(\mathbf{x})w_{2,1}\psi_1(\mathbf{x}') + w_{2,2}\psi_2(\mathbf{x})w_{2,1}\psi_1(\mathbf{x}') + \cdots \\
\cdots + w_{2,H}\psi_H(\mathbf{x})w_{2,H}\psi_H(\mathbf{x}')\big]
\end{aligned} \qquad (2.32)
$$

if parameter priors are independent, the terms between separate hidden units are zero, e.g. $\mathbb{E}[w_{2,1}\psi_1(\mathbf{x})w_{2,2}\psi_2(\mathbf{x}')] = \mathbb{E}[w_{2,1}]\mathbb{E}[\psi_1(\mathbf{x})]\mathbb{E}[w_{2,2}]\mathbb{E}[\psi_2(\mathbf{x}')] = 0$, so,

$$
\begin{aligned}
= \mathbb{E}\big[w_{2,1}\psi_1(\mathbf{x})w_{2,1}\psi_1(\mathbf{x}') + w_{2,2}\psi_2(\mathbf{x})w_{2,2}\psi_2(\mathbf{x}') + \cdots \\
\cdots + w_{2,H}\psi_H(\mathbf{x})w_{2,H}\psi_H(\mathbf{x}')\big]
\end{aligned} \qquad (2.33)
$$

and if priors are identically distributed,

$$
\begin{aligned}
&= H\mathbb{E}\big[w_2\psi(\mathbf{x})w_2\psi(\mathbf{x}')\big] && (2.34)\\
&= \sigma_{w2}^2\mathbb{E}\big[\psi(\mathbf{x})\psi(\mathbf{x}')\big], && (2.35)
\end{aligned}
$$

since w_2 variance was earlier defined to be scaled by width.

Having derived expressions for mean and covariance, it remains to show that the joint distribution is Gaussian. Eq. 2.35 is a sum (mean) of H iid random variables, hence, under mild conditions, the CLT states that the distribution over functions is normally distributed as $H \to \infty$.

An alternative, intuitive way to motivate the requirement of infinite hidden units is to consider zooming in on a section of a BNN's function. Observe that the function should not degenerate to some piece-wise constant - this is illustrated in (Neal, 1997) [figure 2.1].

Adding a final bias to eq. 2.25 of mean zero and variance σ_{b2}^2 does not affect $\mathbb{E}[f(\mathbf{x})]$, with the resulting kernel,

$$
K(\mathbf{x},\mathbf{x}') = \sigma_{w2}^2\mathbb{E}\big[\psi(\mathbf{x})\psi(\mathbf{x}')\big] + \sigma_{b2}^2. \qquad (2.36)
$$

2.6.2 Analytical BNN Kernels - Single-Layer NNs

The previous section showed a widely known result - that an infinitely wide single hidden layer BNN converges to a GP. Less well known is that the *specific* kernel of this GP is available, in closed form, for a variety of popular activation functions (Williams, 1996).

To derive analytical kernels for specific activations, ψ, and priors, $p(\mathbf{w}_1)$ & $p(b_1)$, eq. 2.36 must be evaluated.

$$K(\mathbf{x},\mathbf{x}') = \sigma_{b2}^2 + \sigma_{w2}^2 \int\int \psi(\mathbf{x})\psi(\mathbf{x}')p(\mathbf{w}_1)p(b_1)d\mathbf{w}_1 db_1 \tag{2.37}$$

Commonly the bias is appended to the weight vector, $\tilde{\mathbf{w}}_1 = [\mathbf{w}_{1,1}, \mathbf{w}_{1,2}..., 0]$, with $p(\tilde{\mathbf{w}}_1) = \mathcal{N}(\mathbf{0}, \mathbf{\Sigma})$, and $\mathbf{\Sigma}$ is a diagonal matrix.

$$= \sigma_{b2}^2 + \sigma_{w2}^2 \int \psi(\mathbf{x})\psi(\mathbf{x}')p(\tilde{\mathbf{w}}_1)d\tilde{\mathbf{w}}_1 \tag{2.38}$$

The integral is generally not trivial, and several papers have focused on deriving analytical forms for popular activation functions which are listed shortly. Naturally eq. 2.37 can be computed numerically where analytical forms do not exist.

This thesis has contributed to this list, for cosine activations, and also for more modern activation functions - ELU, GELU, and PReLU - through collaboration with Russell Tsuchida (Tsuchida et al., 2020). These are discussed in chapter 3. Chapter 1 disentangles personal contributions.

The analytical forms for these kernels are now listed. It is assumed all parameters are normally distributed, with mean zero, and variance shared by parameter layer and type, e.g. $\mathbf{w}_{1i} \sim \mathcal{N}(0, \sigma_{w1}^2), \ \forall i.$

ERF/probit Activation (Williams, 1996) , $\psi(x) = \frac{2}{\sqrt{\pi}} \int_0^x \exp -t^2 dt$. Note that this is a proxy for the more commonly used TanH function, which doesn't have an analytical solution.

$$K_{\text{ERF}}(\mathbf{x},\mathbf{x}') = \sigma_{b2}^2 + \frac{2\sigma_{w2}^2}{\pi} \sin^{-1} \frac{2\sigma_{b1}^2 + 2\sigma_{w1}^2 \mathbf{x}\cdot\mathbf{x}'}{\sqrt{\left(1+2\sigma_{b1}^2+2\sigma_{w1}^2||\mathbf{x}'||^2\right)\left(1+2\sigma_{b1}^2+2\sigma_{w1}^2||\mathbf{x}||^2\right)}} \tag{2.39}$$

RBF Activation (Williams, 1996), This requires a slight reformulation of the NN - instead of, $\psi(\mathbf{w}_{1i}\mathbf{x}+b_{1i})$, it uses, $\psi(\mathbf{x}-\mathbf{w}_{1i}) = \exp\left(-\frac{1}{2\sigma_g^2}||\mathbf{x}-\mathbf{w}_{1i}||^2\right)$, for some constant σ_g^2. Define for convenience, $1/\sigma_e^2 = 2/\sigma_g^2 + 1/\sigma_{w1}^2$, $\sigma_s^2 = 2\sigma_g^2 + \sigma_g^4/\sigma_{w1}^2$, $\sigma_m^2 = 2\sigma_{w1}^2 + \sigma_g^2$.

$$K_{\text{RBF}}(\mathbf{x},\mathbf{x}') = \sigma_{b2}^2 + \sigma_{w2}^2 \left(\frac{\sigma_e}{\sigma_{w1}}\right)^d \exp\left(-\frac{||\mathbf{x}||^2}{2\sigma_m^2}\right) \exp\left(-\frac{||\mathbf{x}-\mathbf{x}'||^2}{2\sigma_s^2}\right) \exp\left(-\frac{||\mathbf{x}'||^2}{2\sigma_m^2}\right) \tag{2.40}$$

As $\sigma_{w1}^2 \to \infty$, the SE kernel in eq. 2.16 is recovered.

RBF with cos/sin warping (chapter 3), consider a warping, $x \to (\cos(\frac{2\pi x}{p}), \sin(\frac{2\pi x}{p}))$, followed by a single-layer RBF BNN taking this warped 2-D input.

$$K_{\text{RBF warp}}(\mathbf{x},\mathbf{x}') = \sigma_{b2}^2 + \sigma_{w2}^2 \left(\frac{\sigma_e}{\sigma_{w1}}\right)^2 \exp\left(-\frac{1}{\sigma_m^2}\right) \exp\left(-\frac{2\sin^2(\frac{\pi}{p}(x-x'))}{\sigma_s^2}\right) \tag{2.41}$$

This is the same form as the popular periodic kernel in eq. 2.21.

ReLU Activation (Cho and Saul, 2009), $\psi(x) = \max(0,x)$.

$$K_{\text{ReLU}}(\mathbf{x},\mathbf{x}') = \sigma_{b2}^2 + \frac{\sigma_{w2}^2}{2\pi}\sqrt{\left(\sigma_{b1}^2 + \sigma_{w1}^2||\mathbf{x}||^2\right)\left(\sigma_{b1}^2 + \sigma_{w1}^2||\mathbf{x}'||^2\right)}\left(\sin\theta^0 + \left(\pi - \theta^0\right)\cos\theta^0\right) \tag{2.42}$$

Where θ^0 is the angle between the input vectors (also used in subsequent kernels).

$$\theta^0 = \cos^{-1}\left(\frac{\sigma_{b1}^2 + \sigma_{w1}^2 \mathbf{x}\cdot\mathbf{x}'}{\sqrt{\left(\sigma_{b1}^2 + \sigma_{w1}^2||\mathbf{x}||^2\right)\left(\sigma_{b1}^2 + \sigma_{w1}^2||\mathbf{x}'||^2\right)}}\right) \tag{2.43}$$

Leaky ReLU Activation (Tsuchida et al., 2018), $\psi(x) = \max(ax,x), a \in [0,1]$.

$$K_{\text{LReLU}}(\mathbf{x},\mathbf{x}') = \\ \sigma_{b2}^2 + \sigma_{w2}^2\sqrt{\left(\sigma_{b1}^2 + \sigma_{w1}^2||\mathbf{x}||^2\right)\left(\sigma_{b1}^2 + \sigma_{w1}^2||\mathbf{x}'||^2\right)}\left(\frac{(1-a)^2}{2\pi}\left(\sin\theta^0 + (\pi - \theta^0)\cos\theta^0\right) + a\cos\theta^0\right) \tag{2.44}$$

PReLU Activation is similar to Leaky ReLU, but treats a as a learnable parameter (hence the name ‘parametric ReLU’). To compute the kernel, a prior is placed over this parameter.

For the choice of a uniform distribution, $p(a) = \mathcal{U}(0,1)$, the result is straightforward,

$$K_{\text{PReLU}}(\mathbf{x},\mathbf{x}') =$$
$$\sigma_{b2}^2 + \frac{\sigma_{w2}^2}{2\pi}\sqrt{\left(\sigma_{b1}^2 + \sigma_{w1}^2||\mathbf{x}||^2\right)\left(\sigma_{b1}^2 + \sigma_{w1}^2||\mathbf{x}'||^2\right)}\left(\frac{1}{6\pi}\left(\sin\theta_0 + (\pi-\theta_0)\cos\theta_0\right) + \frac{1}{2}\cos\theta_0\right). \tag{2.45}$$

GELU and ELU Activation (Tsuchida et al., 2020), are given by $\psi(x) = x/2(1+\text{erf}(x))$ and $\psi(x) = \Theta(x)x + \Theta(-x)(\exp(z)-1)$ respectively, where $\Theta(\cdot)$ is the heaviside step function.

These forms are rather lengthy so are not included here, see (Tsuchida et al., 2020).

Cosine Activation (chapter 3), $\psi(x) = \cos(x)$.

$$K_{\text{cos}}(\mathbf{x},\mathbf{x}') = \sigma_{b2}^2 + \frac{\sigma_{w2}^2}{2}\left(\exp\left(-\frac{||\mathbf{x}-\mathbf{x}'||_2^2}{2/\sigma_{w1}^2}\right) + \exp\left(-\frac{||\mathbf{x}+\mathbf{x}'||_2^2}{2/\sigma_{w1}^2} + 2\sigma_{b1}^2\right)\right) \tag{2.46}$$

Note the kernel is not periodic, it is the sum of the SE kernel (eq. 2.16), and another term.

Step Function Activation (Cho and Saul, 2009; Williams, 1996) , $\psi(x) = \begin{cases} 1, & \text{for } x > 0 \\ -1, & \text{else} \end{cases}$.

$$K_{\text{step}}(\mathbf{x},\mathbf{x}') = \sigma_{b2}^2 + \sigma_{w2}^2\left(1 - \frac{2\theta^0}{\pi}\right) \tag{2.47}$$

(This can be derived from the ERF kernel, taking variances to infinity.)

2.6.3 Analytical BNN Kernels - Deep NNs

The previous section listed analytical kernels for single-layer fully-connected BNNs with specific activations. Deeper BNNs also converge to GPs as hidden node width is taken to infinity (Lee et al., 2018; Matthews et al., 2018).

Their kernels can be computed analytically if activations as in the previous section are used, however these must be found through recurrent equations. Below, the structure of the recursion is shown for the case of ReLU activations, other activations follow similarly.

$$K^L(\mathbf{x},\mathbf{x}') = \\ \sigma_{bL}^2 + \frac{\sigma_{wL}^2}{2\pi}\sqrt{K^{L-1}(\mathbf{x},\mathbf{x})K^{L-1}(\mathbf{x}',\mathbf{x}')}\Big(\sin\theta^{L-1}(\mathbf{x},\mathbf{x}') + \big(\pi - \theta^{L-1}(\mathbf{x},\mathbf{x}')\big)\cos\theta^{L-1}(\mathbf{x},\mathbf{x}')\Big) \tag{2.48}$$

$$\theta^L(\mathbf{x},\mathbf{x}') = \cos^{-1}\left(\frac{K^L(\mathbf{x},\mathbf{x}')}{\sqrt{K^L(\mathbf{x},\mathbf{x})K^L(\mathbf{x}',\mathbf{x}')}}\right) \tag{2.49}$$

The initial layer, $L=0$, covariance is defined as,

$$K^0(\mathbf{x},\mathbf{x}') = \sigma_{b1}^2 + \sigma_{w1}^2\mathbf{x}\cdot\mathbf{x}' \tag{2.50}$$

One consequence of this recurrence is that infinitely wide deep NNs produce ‘flat’ GP kernels (as opposed to deep GPs (Damianou and Lawrence, 2013)). This is something of a contradiction - earlier we motivated deep NNs through the need for hierarchy, yet this is absent from the corresponding GPs (MacKay, 1998).

A further issue arises from the recurrence; it can also be shown that as depth increases, the kernels degenerate into undesirable functions like input independent constants or white noise (Duvenaud et al., 2014; Schoenholz et al., 2016; Yang and Schoenholz, 2017).

As well as full-connected NNs, similar results have been shown for convolutional NNs (Garriga-Alonso et al., 2019; Novak et al., 2019), and recurrent architectures (Yang, 2019).

2.6.4 Implications of the Connection

The connection between NNs and GPs is more than a mathematical curiosity. It has so far been studied with two primary objectives in mind. Firstly it offers a rare opportunity to study NNs in an analytical way, for example it has provided insight into initialisation schemes for NNs (Yang and Schoenholz, 2017), and dynamics of SGD (Jacot et al., 2018; Lee et al., 2019). Secondly, it has motivated the use of equivalent GPs as replacements for NNs (Arora et al., 2020, 2019).

This thesis follows a third line that has not been deeply explored; taking inspiration from GPs to help specify better priors for BNNs. This is the first major contribution of the thesis, presented in chapter 3.

The connection also brings certain downsides. Kernel methods have long been criticised for their inability to learn 'highly varying functions' (Bengio, 2009; Bengio and Lecun, 2007), that is, a small number of causal variables such as lighting angle, colour, and orientation, can lead to a large variety of data examples. It's questionable whether kernel methods such as GPs have the ability to learn such functions efficiently, without a training set covering all possible permutations. This has motivated hierarchical models such as deep NNs, which *are* effective at learning locally compositional functions (Poggio et al., 2017). An interesting question remains; if deep NNs correspond to flat GPs in the infinite-width limit, is this a sensible limit to take (MacKay, 1998)?

2.7 Summary: Model vs. Inference

This chapter has provided the required foundational knowledge for later contributions of the thesis. This section briefly draws attention to a useful distinction; that of 'model' and 'inference', which have so far been presented in an entwined way.

Model here refers to the algorithmic steps taken to transform the input to the output, and the assumptions embedded within. This chapter introduced two models; the NN in section 2.2.2, and the GP in eq. 2.19 & 2.20. Each model encodes certain assumptions about the input to output mapping. For example a convolutional NN has finite capacity and applies translational equivariance.

Inference refers to the process by which learning is done in a model. For NNs, one inference technique is SGD, another is Bayesian inference that itself can be implemented in a variety of ways as discussed in section 2.4.3.

The distinction between model and inference is sometimes blurred in deep learning - for example, popular modifications to the inference process such as data augmentation, could be interpreted as indirect modelling assumptions.

This thesis will contribute to improving the NN model (albeit assuming the inference process will be Bayesian) in chapter 3, and then to the (Bayesian) inference process itself (assuming the model is a NN) in chapter 4. Chapter 5 contributes to both areas, proposing both a modified model and modified loss function.

Chapter 3

Expressive Priors in BNNs

TLDR: How to design a BNN that incorporates prior knowledge about a function, using GPs as a lens.

Key background sections 2.4, 2.5, 2.6.

This chapter presents the first technical contribution of the thesis. Bayes' rule provides a principled framework for handling uncertainty, with a central step being specification of an appropriate model. However, applied to NNs, one problematic question arises; how can a prior belief about a task be converted to a prior over NN parameters?

This chapter begins by outlining why the link between BNNs and GPs provides an effective lens to reason about this question (section 3.1). It proceeds in section 3.2 by offering an intuitive interpretation of the effect of parameter priors in fully-connected NNs for various activation functions. This represents a minor contribution over previous work which commented on broad behaviour empirically (Gal, 2016; Neal, 1997).

This interpretation provides the foundations of a practical toolkit helping modellers to build BNNs with priors suited to a task. The major contribution of the chapter is then adding to this toolkit in several ways.

1. Section 3.3 shows how a proxy GP can be used as a vehicle to rapidly optimise priors, which can then be converted to an equivalent BNN architecture.
2. Section 3.4 - BNNs are designed that mirror kernel combinations in GPs.
3. Section 3.5 - BNNs are designed that recover periodic GP kernels, which are often useful in the context of kernel combinations.

Work in sections 3.2, 3.4 & 3.5 was presented in (Pearce et al., 2019b). The GELU kernel was derived in (Tsuchida et al., 2020).

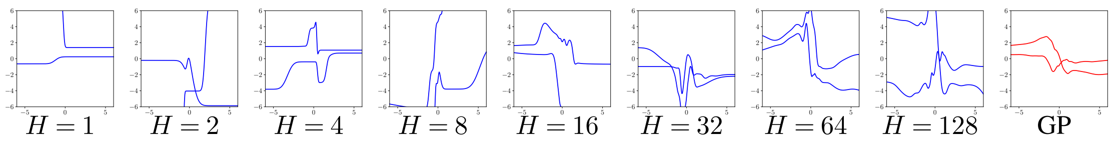


Figure 3.1 Priors from finite-width BNNs provide an increasingly close approximation to priors drawn from the equivalent GP kernel when BNN width, H, is increased - here for ERF activation and kernel.

3.1 GPs as a Lens to Study BNN Priors

Section 2.6 showed that BNNs converge to GPs as their width is taken to infinity. Analytical kernels were listed that correspond to BNNs of particular architectures. Whilst correspondence is only exact in the limit of infinite width, wide (but finite) BNNs offer a reasonable approximation to these GPs. This was confirmed in several studies which stated the rate of convergence as $\mathcal{O}(1/\sqrt{H})$ (Lee et al., 2019; Matthews et al., 2018; Williams, 1996). Figure 3.1 provides illustrative evidence of this.

Our key observation is that this relationship can be reversed; **the GP kernels describe, approximately, the behaviour of wide BNNs**. This observation is valuable in the context of priors. Rather than attempting to analyse the meaning of a high-dimensional distribution over a BNN parameters, instead we can examine the equivalent GP kernels, which contain only a handful of hyperparameters.

This lens imposes some restrictions on the type of BNN we can analyse - parameters are enforced as iid across layers and of finite variance. Since it is common practice to use these types of priors in BNNs, this is not particularly problematic. Also, analysis is only possible for architectures where analytical kernels are available. This is again not too problematic - kernels have been derived for many popular activation functions, and where they do not exist, kernels exist for functionally similar activations - e.g. though TanH is not available, the qualitatively similar ERF is.

3.2 Interpreting BNN Priors

This section discusses, in intuitive terms, how activation functions, number of layers, and parameter prior variances affect the prior over functions. It assumes normally distributed independent parameter priors, of zero mean and variance shared across layers. Weight prior variances are scaled by the number of hidden nodes in the preceding layer, $\bar{\sigma}_{wi}^2 := \sigma_{wi}^2/H_{i-1}$.

Swapping Activations Swaps Priors Over Functions

In deep learning, activation functions are often motivated by their compatibility with gradient descent (e.g. managing the vanishing gradient problem), or computational simplicity (Goodfellow et al., 2016). There is an additional impact of activation function that is less often discussed - by changing activation functions one effectively is changing the prior over functions. This was shown in section 2.6; different activations lead to GPs with different kernels.

Figure 3.2 shows example prior draws as well as a predictive distribution on toy data for single-layer BNNs with varying activations. The differences are non-trivial; for example, ReLU activations lead to extrapolations that are approximately linear well outside the data range, while the ERF produces flat extrapolations.

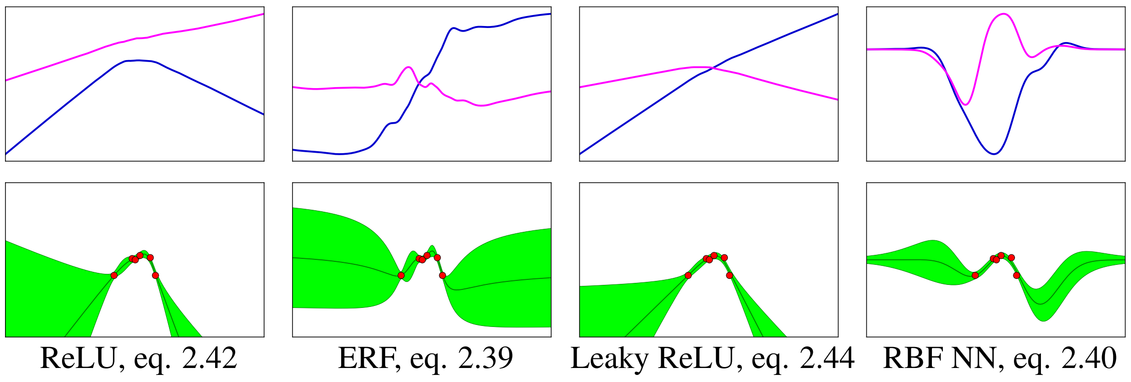


Figure 3.2 Top: Prior draws for infinitely wide single-layer BNNs of various architectures. Bottom: Predictive distributions on toy data.

Final Layer Weight and Bias Variance

For all activation functions, the weight and bias variance of the final layer have a consistent and interpretable effect on the prior over functions. From section 2.6.2, it can be seen that for all kernels, σ_{b2}^2 simply adds some constant value to the covariance. This has the effect of increasing the 'spread' of the prior functions. On the other hand, σ_{w2}^2 multiplies the kernel by a constant value, which controls the amplitude of the prior functions. Figure 3.3 illustrates that the effect is to apply a linear translation to the functions.

When tackling regression tasks with NNs, it is common practice to standardise inputs and outputs to zero mean and unit variance. Final layer variance parameters should therefore be set to produce a prior over functions in this range. For classification tasks the relationship is less clear since the NN outputs (logits) are squashed through a softmax. In this case σ_{w2}^2 controls the amplitude of the logits, which affects both how confident a prediction can be, and how rapidly the transition from one class to another - shown in figure 3.4.

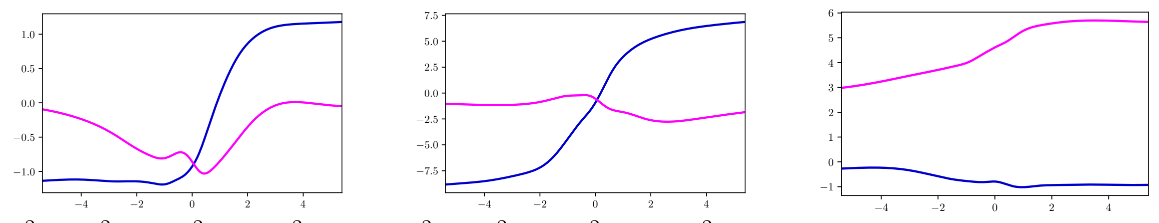


Figure 3.3 Final layer variances: σ_{w2}^2 controls the amplitude of the whole function, whilst σ_{b2}^2 controls the spread. ERF activations shown.

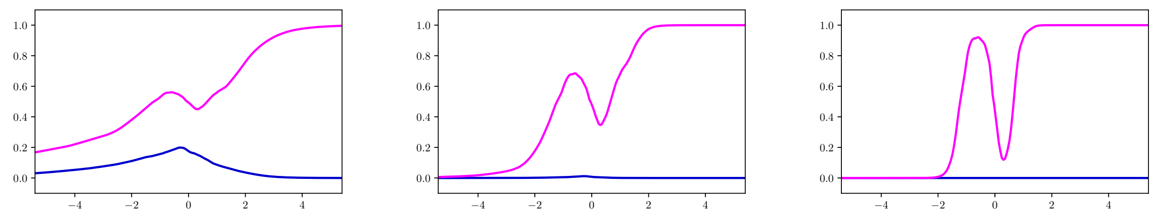


Figure 3.4 Final layer variances for classification: σ_{w2}^2 controls the both absolute confidence and the speed at which class predictions can switch. Single-layer NN with ReLU activations shown.

First Layer Weight and Bias Variance

Related work (Gal, 2016) [chapter 4] argued that weight and bias variances were the reciprocal of lengthscale, but here we show the effect is more nuanced, and varies depending on the activation function. The proceeding discussion constrains $\sigma_{w1}^2 = \sigma_{b1}^2$ to simplify analysis.

For ERF, increasing $\sigma_{w1}^2, \sigma_{b1}^2$ has the effect of creating noisier, more wiggly functions (figure 3.5). This is similar to the role of lengthscale in the traditional SE GP kernel, though ERF activations do not degenerate to white noise as in the SE kernel, maintaining some lower frequencies. An analysis of the effect of $\sigma_{w1}^2, \sigma_{b1}^2$ on input angles in the GP kernel shows this in figure 3.6.

For ReLU activations, one can see from eq. 2.42 & 2.43 that $\sigma_{w1}^2, \sigma_{b1}^2$ affect the kernel in two places. Firstly, they have a trivial effect on the magnitude of the kernel which, as with the final layer variance, simply acts to increase the amplitude of the functions. Secondly, they are present in θ^0 - however, when $\sigma^2_{w1} = \sigma^2_{b1}$, these terms simply cancel out. As such increasing $\sigma^2_{w1}, \sigma^2_{b1}$ has no effect on the 'wiggliness' of functions, and only increases the amplitude[1]. This can be seen in figure 3.6.

For RBF, inspecting eq. 2.40 shows σ^2_s to play a role similar to lengthscale, which is primarily driven by σ^2_g. Weight variance, σ^2_{w1}, primarily controls the span across the input space before functions decay to zero - through its influence on σ^2_m. This decay can be observed in figure 3.2.

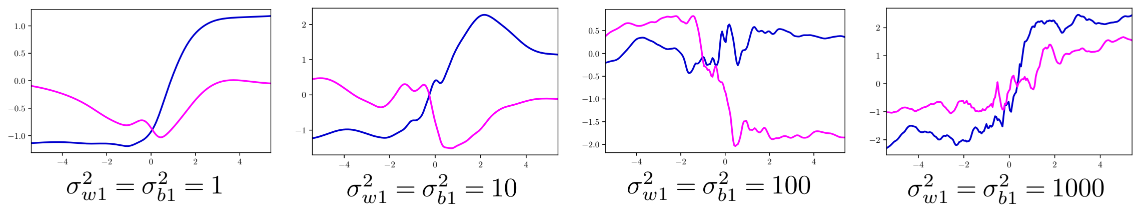


Figure 3.5 Prior draws for a single-layer BNN with ERF activations. Increasing $\sigma^2_{w1}, \sigma^2_{b1}$ creates noisier functions. Here, $\sigma^2_{w2} = 1, \sigma^2_{b2} = 0$.

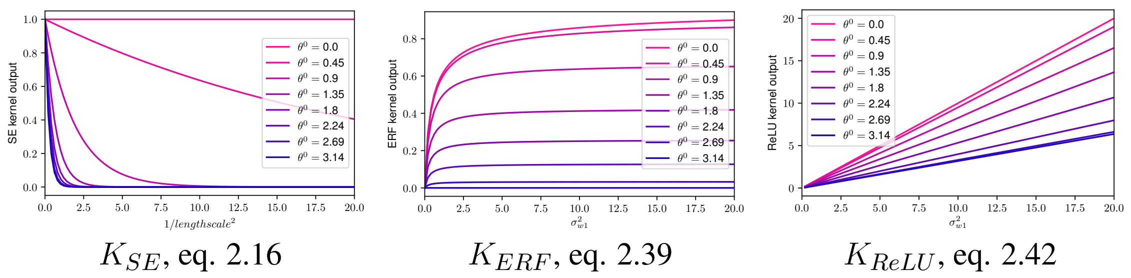


Figure 3.6 Kernel analysis, where $\sigma^2_{w2} = \sigma^2_{b2}$ and $\sigma^2_{w2} = 1, \sigma^2_{b2} = 0$. Input angle, θ^0 between two inputs is varied whilst the norm, $||\mathbf{x}||||\mathbf{x}'|| = 1$ is held constant.

Middle Layer Weight and Bias Variance

In general, the effect of parameter priors in the middle layers follow the roles played by first layer priors (again dependent on the activation function). Of more pressing concern in deeper models, however, is to maintain constant variance through the BNN. A practical approach to specifying priors is therefore to set them so the variance through the BNN remains constant. For example, with ReLU this requires setting $\sigma^2_{wi} = 2/H$. This corresponds to the He initialisation scheme (He et al., 2015), which is derived with a similar motivation.

[1]Concurrent work independently made a similar observation (Wilson and Izmailov, 2020).

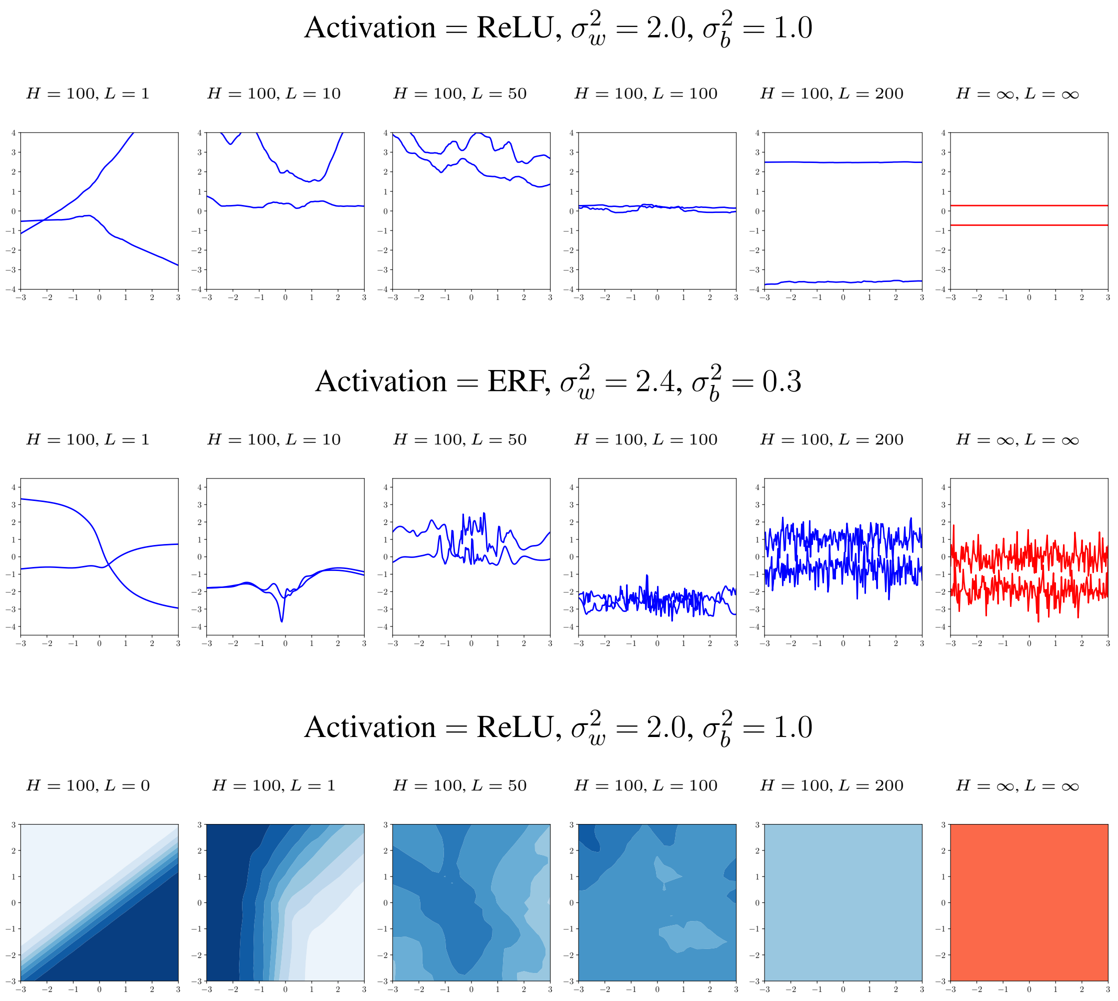


Figure 3.7 Prior draws (top & middle: regression, bottom: classification) for finite BNNs of increasing depth. These degenerate either into input-independent constants or white noise. In red are prior draws from the 'infinite' kernel; that is, by running the recurrence in section 2.6.3 to infinity.

Number of Layers

One of the limitations of studying BNNs via GPs, is that the recursive GPs corresponding to deep BNNs are still 'flat' models (section 2.6.3). There is therefore limited insight that can be gained by studying GPs in terms of the advantages of depth. However, it does turn out to be a useful way to study the *challenges* of depth.

Even when priors are set to maintain variance through a BNN, it transpires that as depth is increased, the prior over functions degenerates, producing either input-independent constants or white noise, depending on the activation function (Duvenaud et al., 2014; Poole et al., 2016; Schoenholz et al., 2016). Figure 3.7 illustrates this in regression and classification for increasingly deep finite-width NNs, alongside the GP kernel corresponding to an infinitely deep, infinitely wide BNN. This degeneration is slowed, but not avoided, with residual connections.

3.3 Specifying Priors for BNNs via Proxy GPs

Having used GPs as a lens to describe the meaning of BNN architecture and parameter priors, a method exploiting the connection for practical gain is now proposed.

GP kernels generally have a small number of hyperparameters, which can be conveniently tuned by maximising marginal likelihood (section 2.4.1) for example via gradient descent. Modern GP libraries such as GPflow (De et al., 2017) handle this automatically.

Conversely, priors for BNNs are generally tuned along with other model hyperparameters, through a relatively inefficient cross-validation process e.g. (Blundell et al., 2015; Khan et al., 2018), else ignored and fixed to some default value e.g. (Chua et al., 2018; Gal and Ghahramani, 2015). Certain variational methods allow direct optimisation of the prior and posterior distributions simultaneously (Hernández-Lobato and Adams, 2015), which has been reported to be effective by some authors (Wu et al., 2019), though challenging by others (Blundell et al., 2015).

Given that GPs and BNNs of certain kernels and architectures are interchangeable, we propose to perform optimisation of the BNN prior hyperparameters using the equivalent (‘proxy’) GP, which is efficiently done for smaller datasets. The optimised GP hyperparameters can then be transferred to build a BNN with suitable prior.

This scheme is well suited to scenarios where data grows incrementally, such as in model-based RL or active learning tasks. Here, a GP kernel can be quickly optimised on the initial small dataset. Training then proceeds using the equivalent BNN beginning with suitable prior hyperparameters, enabling learning to proceed in a scalable manner. Figure 3.8 shows a toy example of this on a 1D regression problem with ERF activations and kernel.

3.3.1 Illustrative Experiment: Model-Based RL

This experiment demonstrates the value of optimising BNN priors via proxy GPs in a model-based RL task with an incrementally growing dataset. The cartpole environment was used, where rewards, r_t, are maximised by holding a pole (starting from a hanging position) at a

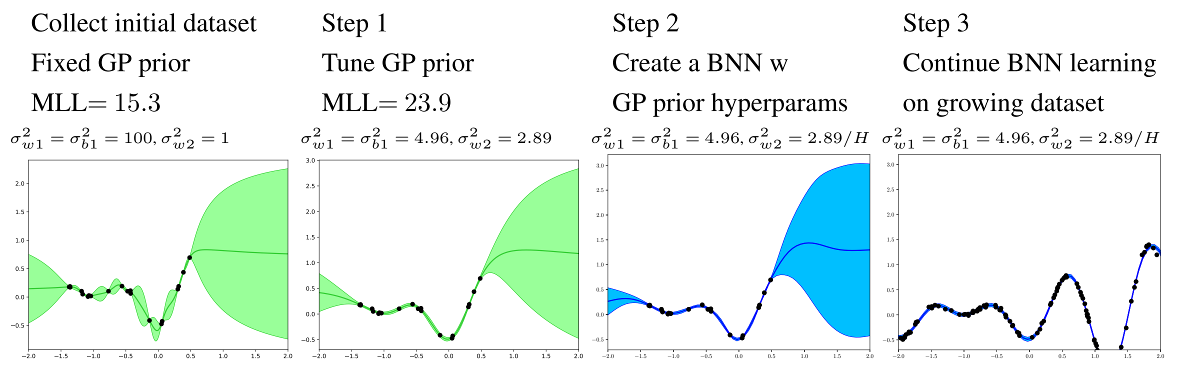


Figure 3.8 Using a proxy GP to specify a prior for a single-layer BNN with ERF activations. In step 1, the parameters of the GP ERF kernel (eq. 2.39) are optimised. Note the tuned variance in the GP model is normalised by H when building the BNN.

target location (central and upright). The action space is continuous, and applies a force to the cart in the range $a_t \in [-1, 1]$. Episodes ran for 200 time steps, $t = 0, 1...199$. The state vector, s_t, consists of four variables; cart position, cart velocity, pole angle, pole angular velocity. The reward function is assumed given.

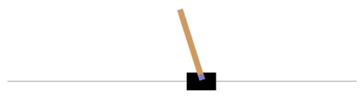


Figure 3.9 The cartpole control task is a classic RL problem. Rewards are maximised by balancing the pole upright. The cart is moved left and right to achieve this, forming the action space for the environment.

In model-based RL, the agent learns a model of the dynamics of the environment, i.e. $p(s_{t+1}|s_t, a_t)$. Given this model, a planning algorithm is used to select actions. Here random shooting was used with model-based predictive control (MPC) (Nagabandi et al., 2017), with a horizon length of 60 timesteps, and 1000 random tracjectories.

It has been shown that accounting for uncertainty is an important property of the dynamics prediction models (Chua et al., 2018), making this a good use case for BNNs. Four BNNs were used, one for the prediction of each state variable. This allowed full tailoring of each BNN prior to its prediction task. Bayesian inference was performed using anchored ensembling (section 4) with 5 NNs per ensemble. BNNs had one hidden layer of width $H = 100$ and used TanH activations.

To generate an initial dataset, a random policy was rolled out for three episodes. This resulted in a small dataset, on which a GP with ERF kernel (eq. 2.39) was optimised. The tuned prior hyperparameters were then converted to a BNN. After each further episode, the BNN was retrained over the full dataset.

Figure 3.10 shows the results of the prior tuned via proxy GP (red), against BNNs that were not tuned at all (blue). The tuned prior provides a clear performance boost in earlier trials, though as the dataset grows, the likelihood begins to dominate the posterior, and the prior has a decreasing effect.

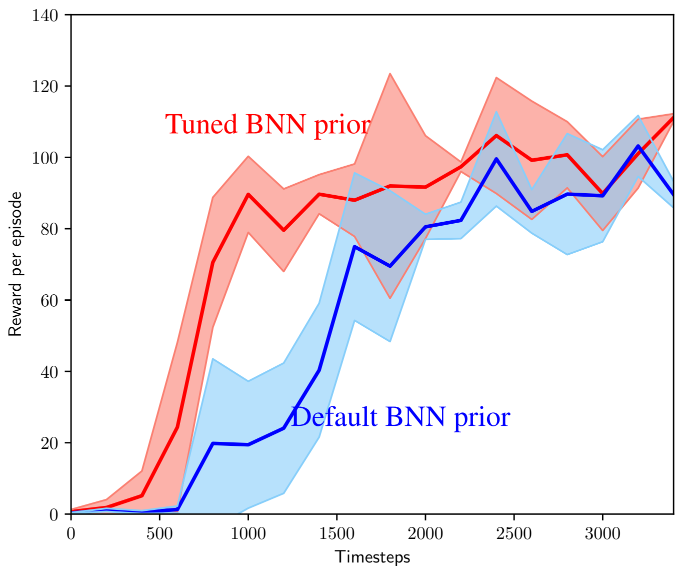


Figure 3.10 Performance on a model-based RL task, cartpole, comparing a BNN with prior tuned using the proposed method (via proxy GP) vs. fixed default prior. Mean ± two standard errors, repeated for three runs.

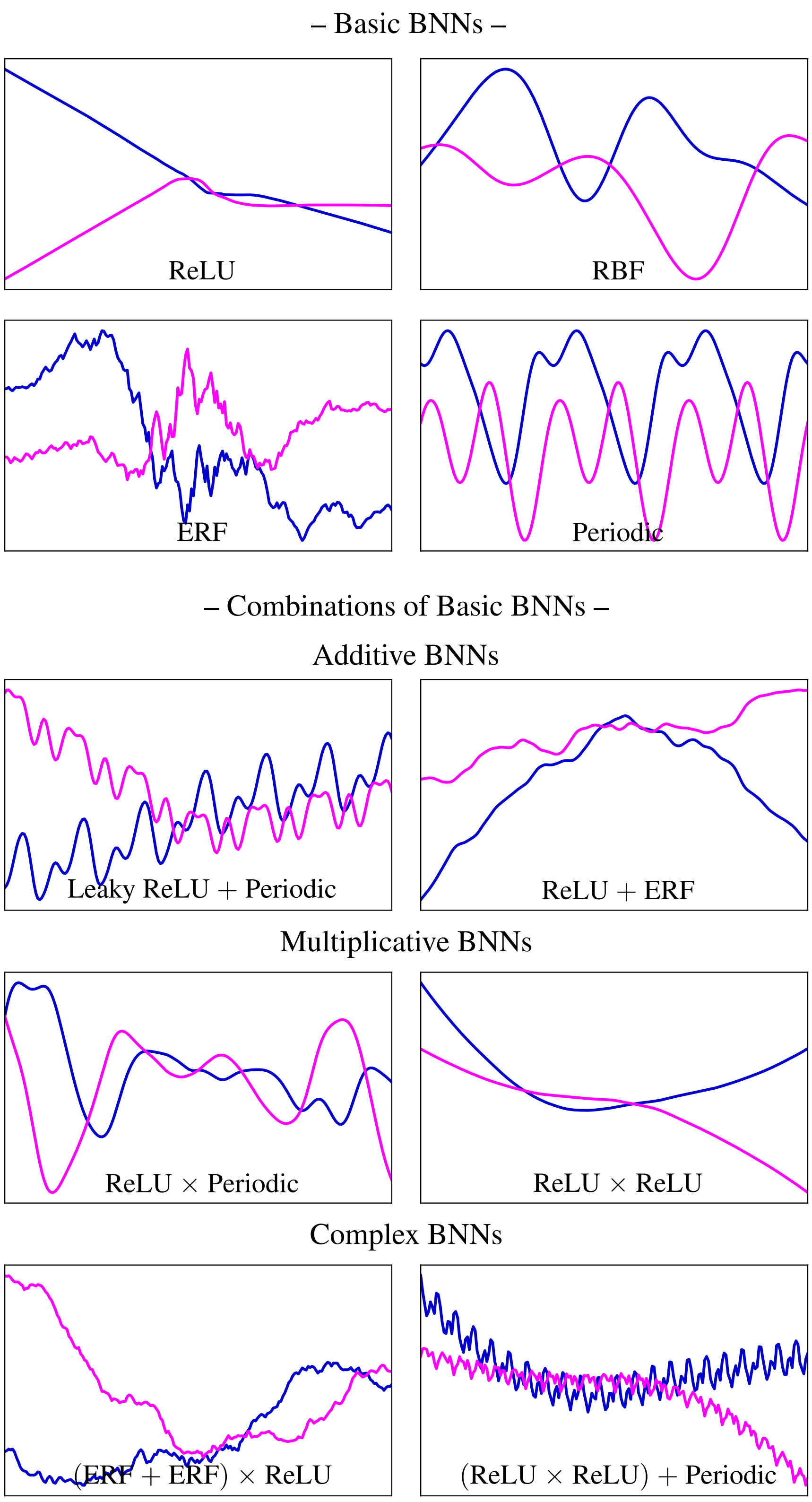


Figure 3.11 BNN architecture determines our prior belief about a function's properties. In general BNNs provide little flexibility in this regard - modifying only the activation function and length scale ('Basic BNNs'). This chapter explores how to design BNNs to produce more expressive prior functions ('Combinations of Basic BNNs'). Two prior draws are shown for each BNN architecture.

3.4 Kernel Combinations in BNNs

The previous section introduced the idea that GPs can be used as an effective vehicle to tune BNN priors. This section considers taking a powerful technique used in prior design for GPs - that of creating new kernels by combining existing kernels (section 2.5.3) - and showing how the same idea can be applied to BNNs. Specifically, we consider how to design BNN architectures such that, in the infinite width limit, they give rise to the equivalent GP kernel combinations. Figure 3.11 illustrates how this technique allows moving from relatively inexpressive ‘basic BNN’ priors to more expressive prior functions.

The kernel combination operations we consider are;

- **Addition**: $K(\mathbf{x},\mathbf{x}') = K_A(\mathbf{x},\mathbf{x}') + K_B(\mathbf{x},\mathbf{x}')$
- **Multiplication**: $K(\mathbf{x},\mathbf{x}') = K_A(\mathbf{x},\mathbf{x}')K_B(\mathbf{x},\mathbf{x}')$
- **Polynomial**: e.g. $K(\mathbf{x},\mathbf{x}') = K_A(\mathbf{x},\mathbf{x}')^2$ (A subset of multiplication.)
- **Warping**: $K(\mathbf{x},\mathbf{x}') = K_A(u(\mathbf{x}),u(\mathbf{x}'))$ for a function, $u : \mathbb{R}^d \to \mathbb{R}^m$

We begin by considering architectures that combine the output of two BNNs. This turns out to be a valid way to add kernels, but not to multiply kernels. We then consider architectures that combine BNNs, point wise, at the final hidden layer. This is valid for multiplicative kernels, but produces a small artefact for additive kernels.

Having derived architectures mirroring additive and multiplicative kernels, section 3.4.3 examines using these in more advanced ways.

3.4.1 Combining BNNs at Output

A straightforward way to combine BNNs is to consider some operation combining their outputs.

Additive

Consider two independent GPs denoted $f_A(\mathbf{x})$ & $f_B(\mathbf{x})$, summed,

$$f_{add}(\mathbf{x}) = f_A(\mathbf{x}) + f_B(\mathbf{x}). \tag{3.1}$$

In general, it is known that $f_{add}(\mathbf{x})$ will also be a GP with kernel, $K_{add}(\mathbf{x},\mathbf{x}') = K_A(\mathbf{x},\mathbf{x}') + K_B(\mathbf{x},\mathbf{x}')$, (Saul et al., 2016).

For two single-layer BNNs, this is recovered by a BNN of architecture,

$$= \sum_{i=1}^{H} w_{A,2,i}\psi_{A,i}(\mathbf{x}) + \sum_{j=1}^{H} w_{B,2,j}\psi_{B,j}(\mathbf{x}), \tag{3.2}$$

where $w_{A,2,i}$ refers to the weight in sub-BNN A, in layer 2, of the i^{th} hidden neuron, and $\psi_{Ai}(\mathbf{x})$ refers to the i^{th} hidden unit of sub-BNN A, after applying the non-linearity.

Since this converges to the sum of two independent GPs, regardless of depth (section 2.6.1), the general GP result applies, and suffices to show that independent BNNs (of infinite width) summed at outputs reproduce a GP with additive kernel.

Multiplicative

Two GPs multiplied together,

$$f_{mult}(\mathbf{x}) = f_A(\mathbf{x}) f_B(\mathbf{x}), \tag{3.3}$$

do *not* generally produce a GP (Rasmussen and Williams, 2006) [4.2.4], even though there does exist a GP with kernel, $K_{mult}(\mathbf{x},\mathbf{x}') = K_A(\mathbf{x},\mathbf{x}')K_B(\mathbf{x},\mathbf{x}')$. (Analogously, the product of two normally distributed random variables is not normally distributed.)

This means that independent BNNs multipled at outputs (shown for the single layer case),

$$= \sum_{i=1}^{H} w_{A,2,i}\psi_{A,i}(\mathbf{x}) \sum_{j=1}^{H} w_{B,2,j}\psi_{B,j}(\mathbf{x}), \tag{3.4}$$

do *not* produce a GP with multiplied kernel.

3.4.2 Combining BNNs at Hidden Layers

Consider now combining BNNs by point wise operations at their (final) hidden layers.

Additive

Taking two single-layer BNNs, the additive case is,

$$f(\mathbf{x}) = \sum_{i=1}^{H} w_{2,i}\big(\psi_{A,i}(\mathbf{x}) + \psi_{B,i}(\mathbf{x})\big), \tag{3.5}$$

where ψ_A and ψ_B are hidden units for each sub-BNN. As in section 3.4.1, neither hyperparameters nor activation function need be shared, e.g. one could take a RBF and ReLU BNN, $\psi_A(\mathbf{x}) = \exp(-||\mathbf{x} - \mathbf{w}_{A1}^T||_2^2/\sigma_g^2)$, and, $\psi_B(\mathbf{x}) = \max(\mathbf{w}_{B1}\mathbf{x} + b_{B1}, 0)$. We now derive the equivalent GP for such an architecture.

Analysis precisely as in section 2.6.1 can be followed up to eq. 2.35, leaving,

$$\begin{aligned} K_{add}(\mathbf{x},\mathbf{x}') =& \sigma_{w2}^2 \mathbb{E}\big[\big(\psi_A(\mathbf{x}) + \psi_B(\mathbf{x})\big)\big(\psi_A(\mathbf{x}') + \psi_B(\mathbf{x}')\big)\big] \\ =& \sigma_{w2}^2 \mathbb{E}\big[\psi_A(\mathbf{x})\psi_A(\mathbf{x}') + \psi_A(\mathbf{x})\psi_B(\mathbf{x}') + \psi_B(\mathbf{x})\psi_A(\mathbf{x}') + \psi_B(\mathbf{x})\psi_B(\mathbf{x}')\big] \end{aligned} \tag{3.6}$$

by linearity of expectation, and noting ψ_A and ψ_B are independent,

$$\begin{aligned} =& \sigma_{w2}^2 \mathbb{E}\big[\psi_A(\mathbf{x})\psi_A(\mathbf{x}')\big] + \sigma_{w2}^2 \mathbb{E}\big[\psi_B(\mathbf{x})\psi_B(\mathbf{x}')\big] + \\ & \quad \sigma_{w2}^2 \mathbb{E}\big[\psi_A(\mathbf{x})\big]\mathbb{E}\big[\psi_B(\mathbf{x}')\big] + \sigma_{w2}^2 \mathbb{E}\big[\psi_A(\mathbf{x}')\big]\mathbb{E}\big[\psi_B(\mathbf{x})\big] \\ =& K_A(\mathbf{x},\mathbf{x}') + K_B(\mathbf{x},\mathbf{x}') + \\ & \quad \sigma_{w2}^2 \mathbb{E}\big[\psi_A(\mathbf{x})\big]\mathbb{E}\big[\psi_B(\mathbf{x}')\big] + \sigma_{w2}^2 \mathbb{E}\big[\psi_A(\mathbf{x}')\big]\mathbb{E}\big[\psi_B(\mathbf{x})\big]. \end{aligned} \tag{3.7}$$

This is the additive kernel plus two extra terms. The impact of these extra terms depends on the activation function, and could be compensated for. If either ψ is an odd function, $\mathbb{E}\big[\psi_{odd}(\cdot)\big] = 0$, the additive kernel is exactly recovered. Alternatively, if both ψs are sigmoids, $\mathbb{E}\big[\psi_{sig}(\cdot)\big] = 0.5$, this results in the kernel, $K_A(\mathbf{x},\mathbf{x}') + K_B(\mathbf{x},\mathbf{x}') + c$, for some constant c. If ψ is a ReLU, $\mathbb{E}\big[\psi_{ReLU}(\cdot)\big]$ is input dependent, making compensation trickier (though still possible).

In general, summing point wise after hidden nodes is not a valid way to reproduce an additive GP kernel, although effects of the artefact terms could be compensated for.

Multiplicative

Following the same procedure for multiplication after hidden nodes,

$$f(\mathbf{x}) = \sum_{i=1}^{H} w_{2,i} \big(\psi_{Ai}(\mathbf{x})\psi_{B,i}(\mathbf{x})\big) \tag{3.8}$$

$$K_{mult}(\mathbf{x},\mathbf{x}') = \sigma_{w2}^2 \mathbb{E}\big[\big(\psi_A(\mathbf{x})\psi_B(\mathbf{x})\big)\big(\psi_A(\mathbf{x}')\psi_B(\mathbf{x}')\big)\big] \tag{3.9}$$

BNN independence allows the rearrangement,

$$= \sigma_{w2}^2 \mathbb{E}\big[\psi_A(\mathbf{x})\psi_A(\mathbf{x}')\big]\mathbb{E}\big[\psi_B(\mathbf{x})\psi_B(\mathbf{x}')\big] \tag{3.10}$$

$$= K_A(\mathbf{x},\mathbf{x}')K_B(\mathbf{x},\mathbf{x}') \tag{3.11}$$

and hence multiplying point wise after hidden nodes is a valid way to reproduce a multiplicative GP kernel.

In order that eq. 3.8 be Gaussian distributed, note that the CLT demands finite variance, $\mathrm{Var}[\psi_{A,i}(\mathbf{x})\psi_{B,i}(\mathbf{x})] < \infty$, decomposed as,

$$\begin{aligned}&\mathrm{Var}[\psi_{A,i}(\mathbf{x})\psi_{B,i}(\mathbf{x})] = \\ &\quad \mathrm{Var}[\psi_{A,i}(\mathbf{x})]\mathrm{Var}[\psi_{B,i}(\mathbf{x})] + \mathbb{E}[\psi_{A,i}(\mathbf{x})]^2\mathrm{Var}[\psi_{B,i}(\mathbf{x})] + \mathbb{E}[\psi_{B,i}(\mathbf{x})]^2\mathrm{Var}[\psi_{A,i}(\mathbf{x})].\end{aligned} \tag{3.12}$$

As such, both mean and variance of each sub-BNN's activations must be finite. This is slightly more onerous than the usual requirement that only their variance be finite.

3.4.3 Extensions

Whilst the previous results were explicitly shown for two single-layer BNNs, it is straightforward to extend them to a variety of situations. Following, we provide examples of useful constructions.

Additive and Multiplicative

Kernel:

$$K(\mathbf{x},\mathbf{x}') = K_A(\mathbf{x},\mathbf{x}') + K_B(\mathbf{x},\mathbf{x}')K_C(\mathbf{x},\mathbf{x}')K_D(\mathbf{x},\mathbf{x}') \tag{3.13}$$

BNN architecture:

$$f(\mathbf{x}) = \sum_{i=1}^{H} w_{2,i}\psi_{A,i}(\mathbf{x}) + \sum_{j=1}^{H} w_{2,j}\psi_{B,j}(\mathbf{x})\psi_{C,j}(\mathbf{x})\psi_{D,j}(\mathbf{x}) \tag{3.14}$$

Polynomials

Kernel:

$$K(\mathbf{x},\mathbf{x}') = K_A(\mathbf{x},\mathbf{x}')^2 \tag{3.15}$$

BNN architecture:

$$f(\mathbf{x}) = \sum_{i=1}^{H} w_{2,i}\psi_{A,1,i}(\mathbf{x})\psi_{A,2,i}(\mathbf{x}) \tag{3.16}$$

Where $\psi_{A,1}$ and $\psi_{A,2}$ are separate nodes sharing common hyperparameters.

Warping

Kernel:

$$K(\mathbf{x},\mathbf{x}') = K(u(\mathbf{x}),u(\mathbf{x}')) \tag{3.17}$$

BNN architecture:

$$f(\mathbf{x}) = \sum_{i=1}^{H} w_{2,i}\psi_{i}(u(\mathbf{x})) \tag{3.18}$$

Separation of Inputs

It can be useful to consider multiple kernels taking subsets of inputs, combined through either addition or multiplication (Duvenaud, 2014) [2.3, 2.4], as visualised in figure 3.12.

Kernel:

$$K(\mathbf{x},\mathbf{x}') = K_A(x_1,x_1') + K_B(x_2,x_2') \tag{3.19}$$

BNN architecture:

$$f(\mathbf{x}) = \sum_{i=1}^{H} w_{2,i}\psi_{A,i}(x_1) + \sum_{j=1}^{H} w_{2,j}\psi_{B,j}(x_2) \tag{3.20}$$

Kernel:

$$K(\mathbf{x},\mathbf{x}') = K_A(x_1,x_1')K_B(x_2,x_2'). \tag{3.21}$$

BNN architecture:

$$f(\mathbf{x}) = \sum_{i=1}^{H} w_{2,i}\psi_{A,i}(x_1)\psi_{B,i}(x_2). \tag{3.22}$$

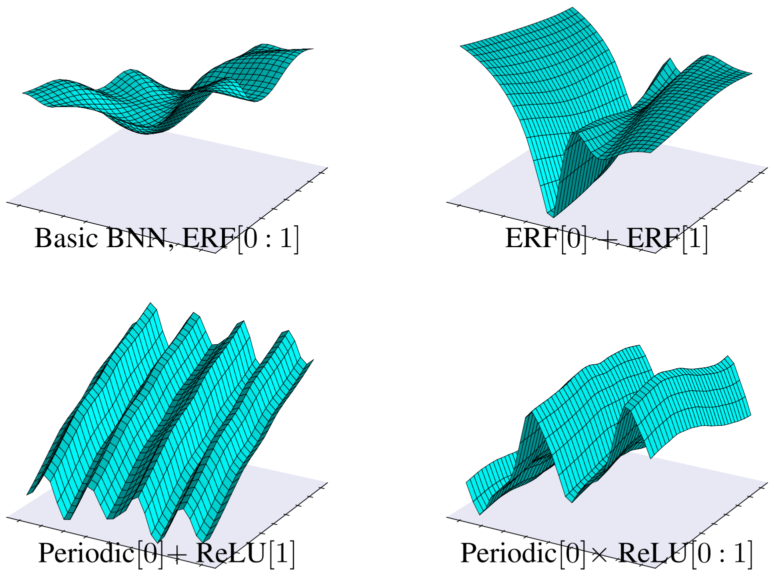


Figure 3.12 Prior draws for a 2-D input. Square brackets designate which dimension(s) each kernel is applied to.

Deeper BNNs

Out of convenience, constructions have been shown for single-layer BNNs. These could be replaced by deep BNNs, which equally correspond to GP kernels (section 2.6.3).

3.5 Periodic BNN Kernels

Having shown how to design BNNs that mirror GP kernel combinations, this section considers how to create BNNs whose priors are periodic functions, which are often useful in this context. Once again we consider how to build a BNN that gives rise to a periodic kernel in the limit of infinite width.

We define a periodic function as, $f(x) = f(x+p)$, for some scalar period, $p \in \mathbb{R}_+$. We will show that cosine activations do not produce a periodic kernel, but applying warping to inputs followed by standard activations functions, does.

3.5.1 Cosine Activations

Consider a single-layer BNN with cosine activation functions; intuition might suggest this leads to a periodic kernel. (Note such activations have been explored in other contexts (Parascandolo et al., 2017; Ramachandran et al., 2017).)

$$f(\mathbf{x}) = \sum_{i=1}^{H} w_{2,i} \cos(\mathbf{w}_{1,i}\mathbf{x} + b_{1,i}) \tag{3.23}$$

Following the usual GP kernel derivation in section 2.6.1,

$$K_{cos}(\mathbf{x},\mathbf{x}') = \sigma_{w2}^2 \iint \cos(\mathbf{w}_1\mathbf{x} + b_1)\cos(\mathbf{w}_1\mathbf{x}' + b_1) p(\mathbf{w}_1) p(b_1) d\mathbf{w}_1 db_1 \tag{3.24}$$

Assuming priors, $p(\mathbf{w}_1) \sim \mathcal{N}(0, \sigma_{w1}^2 I)$, and, $p(b_1) \sim \mathcal{N}(0, \sigma_{b1}^2)$, we find,[2]

$$= \frac{\sigma_{w2}^2}{2}\left(\exp\left(-\frac{||\mathbf{x}-\mathbf{x}'||_2^2}{2/\sigma_{w1}^2}\right) + \exp\left(-\frac{||\mathbf{x}+\mathbf{x}'||_2^2}{2/\sigma_{w1}^2} + 2\sigma_{b1}^2\right)\right). \tag{3.25}$$

Slightly counter-intuitively, the kernel is not periodic. Rather it is the sum of the SE kernel (eq. 2.16), and another term.

We further considered using Laplace and uniform distributions for priors, which did result in kernels containing trigonometric functions, but the forms were untidy and not apparently useful.

Note that our analysis is from the perspective of equivalent GP kernels. It is possible to consider narrow BNNs with cosine activations that *would* produce periodic predictive distributions. If initialised suitably, these may be of some use.

3.5.2 Input Warping

Whilst modifying the activation function failed to produce periodic kernels, applying a warping to inputs is more successful.

[2] Rewrite $\cos(A)\cos(B) = \frac{1}{2}[\cos(A-B) + \cos(A+B)]$, then use, $\mathbb{E}[\cos(\mathbf{x}\mathbf{w})] = \exp(-\frac{1}{2}\mathbf{x}^T\Sigma\mathbf{x})$, if $\mathbf{w} \sim \mathcal{N}(0,\Sigma)$.

The most common periodic kernel used in GP modelling is the ESS kernel (Duvenaud, 2014; MacKay, 1998), as shown in eq. 2.21.

$$K_{\text{ESS}}(x,x') = \sigma^2 \exp\left(- \frac{2\sin^2\left(\frac{\pi}{p}(x-x')\right)}{l^2} \right). \tag{3.26}$$

Having established its value in periodic modelling, we wanted to reproduce this as closely as possible with a BNN. Surprisingly, an exact recovery is possible as follows.

Apply a warping to a 1-D input, $x \to (\cos(\frac{2\pi x}{p}), \sin(\frac{2\pi x}{p}))$, followed by a single-layer RBF BNN taking this 2-D warping as input.

In general, an infinitely wide single-layer RBF BNN produces the GP kernel given in eq. 2.40 (Williams, 1996). If the discussed warping is applied to the inputs, for the 1-D case this becomes,

$$\begin{aligned} &K_{\text{RBF Per}_{\text{BNN}}}(\mathbf{x},\mathbf{x}') = \\ &\left(\frac{\sigma_e}{\sigma_{w1}}\right)^2 \exp\left(- \frac{\cos^2(\frac{2\pi x}{p}) + \sin^2(\frac{2\pi x}{p})}{2\sigma_m^2} \right) \\ &\qquad \exp\left(- \frac{\left(\cos(\frac{2\pi x}{p}) - \cos(\frac{2\pi x'}{p})\right)^2}{2\sigma_s^2} + \frac{\left(\sin(\frac{2\pi x}{p}) - \sin(\frac{2\pi x'}{p})\right)^2}{2\sigma_s^2} \right) \\ &\qquad \exp\left(- \frac{\cos^2(\frac{2\pi x'}{p}) + \sin^2(\frac{2\pi x'}{p})}{2\sigma_m^2} \right). \end{aligned} \tag{3.27}$$

Noting, $\left(\cos(\frac{2\pi x}{p}) - \cos(\frac{2\pi x'}{p})\right)^2 + \left(\sin(\frac{2\pi x}{p}) - \sin(\frac{2\pi x'}{p})\right)^2 = 4\sin^2(\frac{\pi}{p}(x-x'))$, and also, $\cos^2(\cdot) + \sin^2(\cdot) = 1$, this reduces to,

$$= \left(\frac{\sigma_e}{\sigma_{w1}}\right)^2 \exp\left(- \frac{1}{\sigma_m^2} \right) \exp\left(- \frac{2\sin^2(\frac{\pi}{p}(x-x'))}{\sigma_s^2} \right) \tag{3.28}$$

which is of the same form as the periodic ESS kernel. Indeed there is a connection to the derivation of the ESS kernel, which used the same warping followed by the SE kernel (MacKay, 1998).

It is equally plausible to apply the same warping followed by BNNs of other architectures. For example, the single-layer ReLU case results in,

$$K_{\text{ReLU Per}}(\mathbf{x},\mathbf{x}') = \frac{\sigma_{w_2}^2}{\pi}(\sin\omega + (\pi - \omega)\cos\omega) \tag{3.29}$$

where,

$$\omega = \cos^{-1}\left(\frac{\sigma_{b_1}^2 + \sigma_{w_1}^2 \cos(\frac{2\pi}{p}(x - x'))}{\sigma_{b_1}^2 + \sigma_{w_1}^2}\right). \tag{3.30}$$

This is equally suited to periodic modelling, and perhaps more convenient in BNNs given the prevalence of ReLUs.

3.6 Illustrative Experiments

This section provides experiments illustrating the value of techniques presented in sections 3.4 & 3.5. This shows, all things being equal, that BNNs designed to incorporate suitable prior knowledge with these techniques can deliver a performance boost over basic BNNs. These gains should be independent of learning algorithm or inference method, but are necessarily task specific. Hence, experiments are framed as illustrative rather than exhaustive.

We showcase situations benefiting simultaneously from both of the ideas introduced - combinations of BNNs *and* periodic function modelling, although either can also be used separately.

All experiments used BNN widths of 50 hidden nodes. Their success supports our claim that, despite the theory presented being exact only for infinite-width BNNs, it provides sound principles for building expressive BNN models of finite width.

3.6.1 Supervised Learning: Time Series

Time series data often have seasonal fluctuations combined with longer term trends. These experiments show that where basic BNNs struggle to capture such patterns, simple combinations of these basic BNNs produce appropriate priors.

We considered two prediction tasks: CO_2 levels recorded at a Hawaiian volcano (Mauna), and numbers of airline passengers flying internationally (Airline). For both datasets we used ten years of monthly recordings, then deleted data between years 3-5 to create a gap in the series. Below, we qualitatively assess the predictive distribution in both the interpolation region (3-5 years) and an extrapolation region (10-20 years).

In Mauna, seasonal variations appear to be of constant amplitude, suggesting an additive relationship between trend and period, whilst Airline shows increasing amplitudes, suggesting a multiplicative relationship.

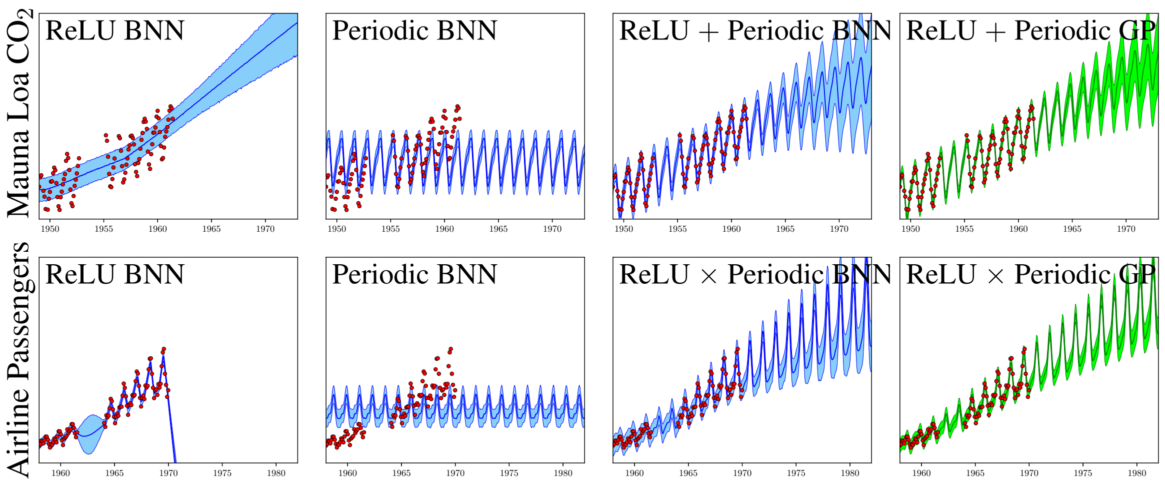


Figure 3.13 Two time series with seasonal fluctuations and long term trends. ReLU and periodic BNNs are separately unable to capture these patterns (first and second from left). However, they succeed when combined in BNN architectures as proposed (third from left), closely approximating the predictive distributions of exact GP inference with the equivalent kernel combinations (right). See main text for model details.

Figure 3.13 shows the two datasets and the predictive distributions produced by four types of model (shading gives ± 3 standard deviations). Inference was performed with HMC for BNNs (Neal, 1997), and analytically for GP.

1. **ReLU BNN** - single-layer BNN with ReLU activations and 100 hidden nodes. There are two possibilities with this model - a long length scale, as shown for Mauna, which captures the long term trend but does not fit the seasonal variations. Alternatively, a short length scale allows better fitting of the training data, but at the expense of extrapolations - in Airline this produces a nonsensical 10-20 year forecast.

2. **Periodic BNN** - single-layer BNN with cos/sin warping applied to input, followed by RBF activations, 100 hidden neurons. This is the structure derived earlier, with equivalent kernel in eq. 3.28. Since these BNNs output pure periodic functions they are unable to fit the data well.

3. **Combined BNN** - these models combined the ReLU & Periodic BNNs from 1. and 2. above, again with 100 hidden neurons. For Mauna, an addition operation at outputs was applied, whilst for Airline, hidden nodes were multiplied point wise. Note that the main characteristics of the datasets are captured. **This creates sensible interpolation and extrapolation predictions.** Importantly, uncertainty increases with the time horizon.

4. **Combined GP** - the GPs corresponding to the combined BNNs in 3. were implemented. For Mauna, the kernel is, $K(\mathbf{x},\mathbf{x}') = K_{\text{RBF Per}_{\text{BNN}}}(\mathbf{x},\mathbf{x}') + K_{\text{ReLU}}(\mathbf{x},\mathbf{x}')$ and for Airline, $K(\mathbf{x},\mathbf{x}') = K_{\text{RBF Per}_{\text{BNN}}}(\mathbf{x},\mathbf{x}') \times K_{\text{ReLU}}(\mathbf{x},\mathbf{x}')$. This enables verification that the BNN architectures produce a predictive distribution corresponding to the GP's (which could be thought of as the 'ground truth'). The slight differences could be put down to the finite width of the BNNs, and imperfect inference procedure.

3.6.2 Reinforcement Learning: Pendulum Swing up

We considered the pendulum swing up task; an agent applies torque to a bar on a pivot, maximising rewards by controlling the bar to be vertically upright. Observations consist of angle, θ, and angular velocity, $\dot{\theta}$.

We used a slightly modified version of the task. Actions were discretised so that torque $\in \{-1, 0, +1\}$. Dynamics were also modified - usually the update rule for θ is,

$$\theta_t = \theta_{t-1} + \dot{\theta}_{t-1} dt, \tag{3.31}$$

where t is time, and $\dot{\theta}$ is a function of the applied torque and gravity. We modified this to,

$$\theta_t = \theta_{t-1} + \frac{2}{1 - e^{-\theta_{t-1}/3}} \dot{\theta}_{t-1} dt. \tag{3.32}$$

This effectively introduces a frictional force that varies according to the absolute value of θ. Crucially this means that as the bar spins, slightly different dynamics are experienced - this could arise from the bar spinning along a thread.

A priori, we therefore know that the function is locally periodic. This makes the task challenging for basic BNN architectures - enforcing exact periodicity is just as inappropriate as ignoring it entirely.

We tested three BNN architectures on the task.

1. **ReLU**: a two-layer ReLU BNN with 50 hidden neurons per layer, with raw angle, θ, and angular velocity, $\dot{\theta}$, and torque as input. Priors: $\sigma^2_{w1} = \sigma^2_{b1} = 1$, $\sigma^2_{w2} = \sigma^2_{b2} = 1/50$, $\sigma^2_{w2} = \sigma^2_{b2} = 10.0$. The model, predicting Q value, $Q(\mathbf{x})$, and with, $\psi(\cdot) := \max(0, \cdot)$, is given by,

$$Q(\mathbf{x}) = \mathbf{W}_3 \psi(\mathbf{b}_2 + \mathbf{W}_2 \psi(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1)). \tag{3.33}$$

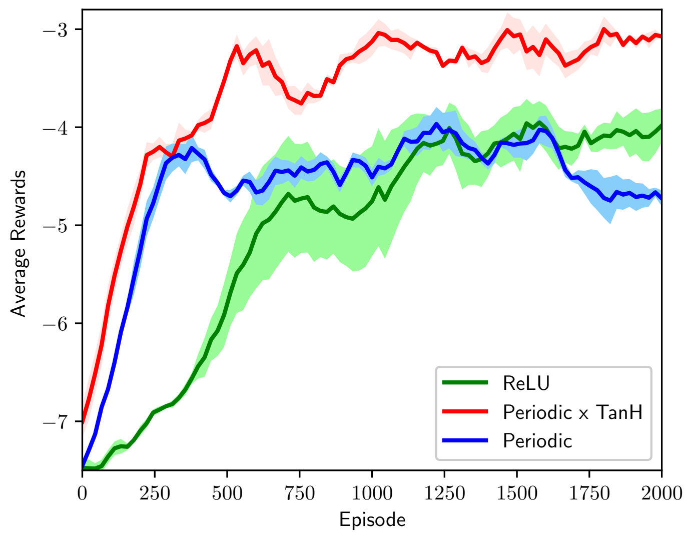


Figure 3.14 Learning for three different BNN architectures on the pendulum task. The BNN incorporating a suitable prior for task, 'Periodic×TanH', outperforms basic BNNs. Mean ± 1 standard error, three runs.

2. **Periodic**: cos/sin input warping applied to θ, raw angular velocity, $\dot{\theta}$, followed by a two-layer ReLU BNN with 50 hidden neurons per layer. Prior variances as for 1. Denoting the warped input as a concatenation, $\tilde{\mathbf{x}} := [\cos(\mathbf{x}), \sin(\mathbf{x})]$.

$$Q(\mathbf{x}) = \mathbf{W}_3\psi(\mathbf{b}_2 + \mathbf{W}_2\psi(\mathbf{W}_1\tilde{\mathbf{x}} + \mathbf{b}_1)) \tag{3.34}$$

3. **Periodic×TanH**: takes the Periodic BNN as in 2., multiplied by a single-layer TanH BNN (taking only θ as input) with long length scale, $\sigma_{w1}^2 = \sigma_{b1}^2 = 0.2$. This combines multiplication, warping and separation of inputs from section 3.4. For $\psi_A(\cdot) := \max(0,\cdot), \psi_B(\cdot) := \tanh(\cdot)$, and elementwise multiplication operator $\odot$,

$$Q(\mathbf{x}) = \mathbf{W}_3\left(\psi_A\left(\mathbf{b}_{A,2} + \mathbf{W}_{A,2}\psi\left(\mathbf{W}_{A,1}\tilde{\mathbf{x}} + \mathbf{b}_{A,1}\right)\right) \odot \psi_B(\mathbf{W}_{B,1}\theta + \mathbf{b}_{B,1})\right). \tag{3.35}$$

Note that the benefits of BNN architecture should be independent of the learning algorithm and inference method. Here we used Bayesian Q-learning (Dearden et al., 1998), similar to regular Q-learning, but with Q-values modelled as *distributions* rather than point estimates, with BNNs as the function approximators.

It was important that a scalable technique be used for inference. Q-learning is sample inefficient, and the experience buffer accumulates hundreds of thousands of data points

$(2,000$ episodes $\times$ 200 time steps). Both GPs and HMC struggle with data of this size. We used Bayesian ensembles for inference - this is the inference method proposed in chapter 4.

Figure 3.14 shows cumulative rewards for the three different architectures over 2,000 episodes. Periodic$\times$TanH clearly outperforms other models, both in terms of learning speed and quality of final policy. This is an example of the *blessing of abstraction* at work - the more structure we account for, the less data we need (Duvenaud, 2014) [p13]. The Periodic BNN has similar learning speed early on, but plateaus since it does not have the flexibility to fully capture system dynamics. ReLU, meanwhile, learns slowly, but has enough flexibility to capture closer to the true dynamics, and eventually surpasses the Periodic BNN.

Figure 3.15 provides evidence for these comments. It shows the dynamics learnt for three revolutions of the pendulum for each BNN. The Periodic and ReLU BNNs are only able to approximate the optimum dynamics found by Periodic$\times$TanH.

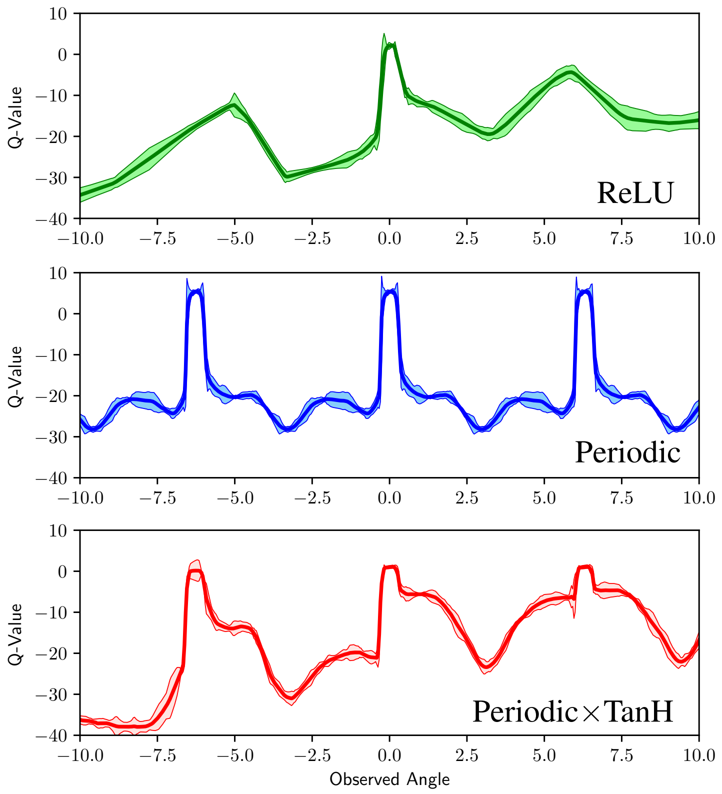


Figure 3.15 Q-values learnt for the action, torque $= 0$, conditioned on observations of $\dot{\theta} = 0$, and input angle, θ, varied on x-axis and measured in radians. The x-axis range allows observation of three complete rotations of the pendulum (each rotation given by 2π radians). Hence, maximum Q-value is predicted when the pendulum is orientated directly upwards. Periodic$\times$TanH captures the local periodicity of the function most accurately.

3.7 Related Work

This section summarises literature on the topic of specifying useful functional priors for BNNs. Note this does not cover explicit studies of properties of parameter priors, e.g. (Nalisnick, 2018), nor does is it cover work where ‘prior’ is used in a looser, not-strictly-Bayesian, sense (Du and Narasimhan, 2019; Dubey et al., 2018). As discussed in section 3.3, the majority of research in BNNs either fixes some default parameter prior (Chua et al., 2018; Gal and Ghahramani, 2015), or does a small level of tuning through cross-validation (Blundell et al., 2015; Khan et al., 2018), without explicitly considering the implied prior over functions.

Several recent works proposed methods to overcome the limited expressivity of BNN priors. Flam-Shepherd et al. (2017) trained a BNN to output GP priors before running inference on a task. Sun et al. (2019) had a similar approach that did not require pretraining. Both methods operate roughly in a supervised learning fashion, training BNNs to match the output of some GP (or any stochastic process in the case of Sun et al. (2019)), on training data augmented with extra OOD sampled data points. The advantage over the proposal in this chapter is that there is full flexibility to choose any GP kernel, without being restricted to the subset corresponding to BNNs. The drawback is that they add complexity to the training process, and the task of generating OOD samples for complex (e.g. image or text) datasets is non trivial. It was reported that sometimes the BNN architecture needed to reflect properies of the prior (Flam-Shepherd et al., 2017).

Hafner et al. (2018) considered a slightly more limited, though practically useful prior; training models to output high uncertainty through a dataset augmented with OOD samples.

Coker et al. (2019) propose a particular formulation of an RBF BNN and show that, for specific choices of weight priors over the central parameters, control over the lengthscale and amplitude in function space is provided, in a decoupled way. This is an interesting proposal, and shares the attractive property of our method; that the prior is directly built into the model. More work is required to develop the proposal beyond single-layer RBF BNNs.

Cui et al. (2020) considered ways to encode prior knowledge about feature sparsity and signal-to-noise ratio into BNNs. The motivating example is gene analysis; one may know a priori that only a small fraction of genes are relevant to a prediction task, and also that individually each gene only explains a very small proportion of total variance.

Krishnan et al. (2019) propose to first train a (non-Bayesian) NN via maximum likelihood. The NN is then converted to a BNN with priors and posteriors centred at their learnt values and (independently) normally distributed. Whilst improved empirical results are reported, a disadvantage is that priors are specified *after* rather than before learning.

Several other works are of relevance. Ma et al. (2019) proposes variational implicit processes for BNNs. Gaier and Ha (2019) could be interpreted as fixing a posterior over parameters, and using evolutionary search to find a BNN architecture producing suitable posterior functions. This flips the Bayesian inference process on its head. Neural processes (Garnelo et al., 2018) apply a meta-learning approach to learn a prior over functions. There is also ongoing discussion on the interplay between architecture and parameter priors; one line of reasoning states that parameter priors may be unimportant if combined with strict architectural priors (Wilson and Izmailov, 2020). Our recent work provided an initial investigation into this (Pearce et al., 2020b).

An orthogonal line of work to ours considers how to improve the scalability of GPs over the default $\mathcal{O}(N^3)$, e.g. (Snelson and Ghahramani, 2006). There also exist a variety of techniques for creating expressive priors in GPs, e.g. (Sun et al., 2018; Wilson and Adams, 2013).

3.8 Discussion & Conclusion

This chapter considered a long-standing challenge in BNNs; how can one specify a prior distribution over weights that incorporates knowledge a modeller might have about the task of interest? We have argued that GPs corresponding to infinitely-wide BNNs provide an effective, interpretable lens to study this, and we provided an intuitive explanation of the meaning of parameter priors for this set up.

Our explanation in itself provides a basic toolbox for prior design in BNNs. We added to it in three ways. 1) We proposed a convenient method to optimise priors using proxy GPs, which can then be translated to BNNs with suitable parameter priors. This was shown to be effective for incrementally growing datasets. 2) Expressive priors can be created in GPs by combining basic kernels into a new kernel. We ported this idea to BNNs, deriving architectures that mirror such kernel combinations. 3) We advanced the modelling of periodic functions with BNNs, which are often useful in this context. Experiments showcased scenarios for which all three were useful; a model-based RL task, time series modelling, and an RL task involving a locally periodic function. Note that this is only one possible way to approach this problem, an alternative might be to simply visualise samples from models in low dimensions.

One question in response to our proposal might be; why not use the GPs directly rather than their BNN equivalents? There are several advantages of these BNNs, 1) they scale to high dimensional input spaces more easily than GPs, 2) they do not strictly enforce Gaussian statistics, which may be unsuitable for some settings, 3) they are able to benefit from hierarchical learning, unlike flat GPs (Aitchison, 2020). Future work could illustrate these advantages more explicitly - one could argue that scalable GPs might perform well on many of the experiments run in this chapter.

One attractive property of our proposal over related work is that building priors directly into the model has limited impact on usability, which was one of the goals of this thesis. This is in contrast to other related BNN approaches, which require some sort of pretraining or data augmentation step, else are restrictive in the architectures they support.

Recent progress in this sub-field now allows a BNN to encode priors about granular properties of a function being modelled; e.g. lengthscale, amplitude, sparsity, periodicity. Whilst this is an important capability and a natural first step, these granular properties are of limited use in complex tasks that motivate deep learning, e.g. it's difficult to see how to incorporate such priors into a BNN learning to play an Atari game from pixels. In practice a modeller might have rich prior knowledge about more abstract properties of the task - we believe the long-term goal of work in this area should be to develop tools enabling this abstract knowledge to be incorporated as a BNN prior.

Chapter 4

Bayesian Ensembling

TLDR: A simple modification to make ensembling 'more Bayesian'.

Key background sections 2.1, 2.2, 2.3, 2.4.

Chapter 2 introduced BNNs as a principled approach to handling epistemic uncertainty in NNs. The previous chapter then considered how priors can be specified for these models, which is the first step of Bayesian methods. The next challenge is to compute the Bayesian posterior. The large number of parameters and data points used with modern NNs make this a formidable task - many Bayesian inference techniques that work well in small-scale settings, e.g. MCMC methods, don't scale to the extent required.

This chapter explores an alternative inference method, that adopts a procedure similar to ensembling. Ensembling is generally described as a non-Bayesian way to estimate uncertainty: it aggregates the estimates of multiple individual NNs, trained from different initialisations and sometimes on noisy versions of the training data. The variance of the ensemble's predictions may be interpreted as its uncertainty. The chief attraction is that the method scales well to large parameter and data settings, with each individual NN implemented in precisely the usual way.

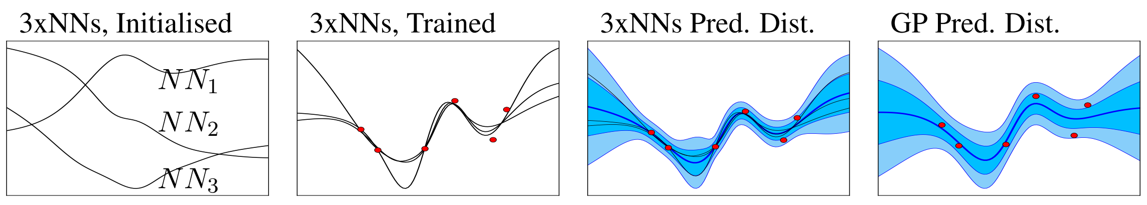


Figure 4.1 An ensemble of NNs, starting from different initialisations and trained with the proposed modification, produce a predictive distribution approximating that of a GP. This improves with number and width of NNs.

This chapter proposes, analyses and tests one modification to the usual NN ensembling process, with the purpose of examining how closely the resulting procedure *does* align with Bayesian inference - regularising parameters about values drawn from a distribution which can be set equal to the prior. Our method falls into a family of methods known as *randomised MAP sampling* (RMS) (section 4.1), similar in spirit to Stein variational gradient descent (Liu and Wang, 2016). We name this procedure *anchored ensembling* - figure 4.1 provides an illustration.

This chapter largely draws from the corresponding paper (Pearce et al., 2019a). A workshop paper further applied anchored ensembling to exploration in RL (Pearce et al., 2018a). Organisation of the chapter and its contributions are as follows.

1. Our first contributions do not specifically consider NNs; we derive an abstracted version of RMS in parameter space rather than output space. This abstraction later allows us to propose RMS for classification tasks for the first time. Under the assumption that the joint parameter likelihood and prior obey a multivariate normal distribution, we show that it is always possible to design an RMS procedure to recover the true posterior.

2. This design requires knowing the parameter likelihood covariance a priori, which is infeasible except in the simplest models. We propose a workaround that results in an approximation of the posterior. In general this approximation has correct mean but underestimated variance and overestimated correlation. However, two conditions lead to an exact recovery: 1) perfectly correlated parameters, 2) parameters whose marginal likelihood variance is infinite ('extrapolation parameters').

3. We proceed by considering the applicability of RMS to NNs. We discuss the appropriateness of assumptions used in the theoretical analysis, and argue that the two conditions leading to exact recovery of the posterior are partially present in NNs. We postulate this as the reason that predictive posteriors produced by anchored ensembling appear very similar to those by exact Bayesian methods in figures 4.4, 4.7, 4.9 & 4.8.

4. The performance of anchored ensembling is assessed experimentally on regression, image classification, sentiment analysis and RL tasks. It provides an advantage over standard ensembling procedures, and is competitive with variational methods.

4.1 Randomised MAP Sampling

Recent work in the Bayesian community, and independently in the RL community, has begun to explore a novel approach to Bayesian inference. Roughly speaking, it exploits the fact that adding a regularisation term to a loss function returns a maximum a posteriori (MAP) parameter estimate. Injecting noise into this loss, either to targets or regularisation term, and sampling repeatedly (i.e. ensembling), produces a *distribution* of MAP solutions mimicking that of the true posterior. This can be an efficient method to sample from high-dimensional posteriors (Bardsley et al., 2014; Chen and Oliver, 2012; Gu et al., 2007).

Whilst it is possible to specify a noise injection that produces exact inference in linear regression, there is difficulty in transferring this idea to more complex settings, such as NNs or classification. Directly applying the noise distribution from linear regression to NNs has had some empirical success, despite not reproducing the true posterior (Lu and Van Roy, 2017; Osband et al., 2018) (section 4.2.2). A more accurate, though more computationally demanding solution, is to wrap the optimisation step in an MCMC procedure (Bardsley, 2012; Bardsley et al., 2014).

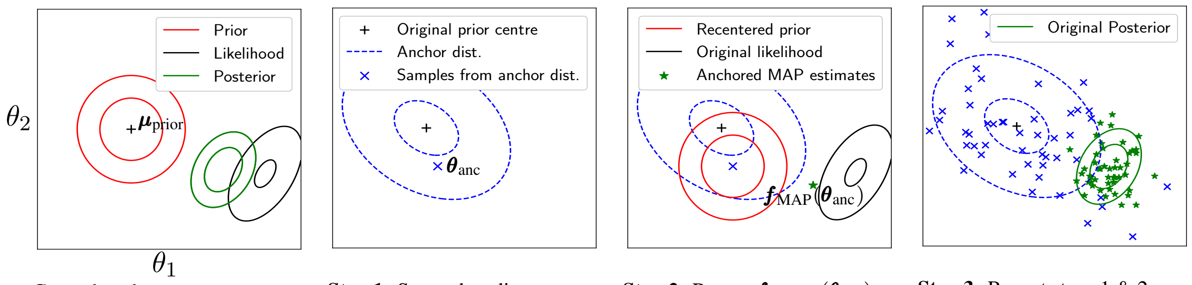
These works have been proposed under several names including randomise-then-optimise, randomised prior functions, and ensemble sampling. We refer to this family of procedures as *randomised MAP sampling* (RMS).

4.2 RMS Theoretical Results

This section presents several novel results. We first derive a general form of RMS by analysing the procedure in parameter space, using the simplifying assumption that both prior and parameter likelihood are multivariate normal distributions. This is an abstraction compared to previous works. Appendix A.1 contains definitions and proofs in full.

If the parameter likelihood covariance is known a priori, we show how RMS can be designed to recover the true posterior. In general, this will not be known, and we propose a practical workaround requiring knowledge only of the prior distribution.

This workaround no longer guarantees exact recovery of the posterior. We derive results specifying in what ways the estimated RMS posterior is in general biased, including under-estimated marginal variance, and overestimated correlation coefficient. We discover two special conditions that lead to an exact recovery.


Prior
Likelihood
Posterior
θ_2
θ_1
$\boldsymbol{\mu}_{\text{prior}}$
Original prior centre
Anchor dist.
Samples from anchor dist.
$\boldsymbol{\theta}_{\text{anc}}$
Recentered prior
Original likelihood
Anchored MAP estimates
$\boldsymbol{f}_{\text{MAP}}(\boldsymbol{\theta}_{\text{anc}})$
Original Posterior

Ground truth: Analytical Bayesian Inference

Step 1: Set anchor dist. as $\mathcal{N}(\boldsymbol{\mu}_{\text{anc}}, \boldsymbol{\Sigma}_{\text{anc}})$ from eq. 4.4 & 4.5. Sample $\boldsymbol{\theta}_{\text{anc}}$ once.

Step 2: Return $\boldsymbol{f}_{MAP}(\boldsymbol{\theta}_{\text{anc}})$ from eq. 4.3 with $\boldsymbol{\theta}_{\text{anc}}$ sampled from step one.

Step 3: Repeat steps 1 & 2. The distribution of $\boldsymbol{f}_{\text{MAP}}(\boldsymbol{\theta}_{\text{anc}})$ recovers original posterior.

Figure 4.2 Demonstration of (exact) RMS in a 2D parameter space.

The appropriateness of the normal assumption in non-linear models for general data likelihoods will be discussed in section 4.3, when we consider applying this RMS scheme with workaround to NNs.

4.2.1 Parameter-Space Derivation

Consider multivariate normal prior and parameter likelihood distributions, $\boldsymbol{\theta} \sim \mathcal{N}(\boldsymbol{\mu}_{\text{prior}}, \boldsymbol{\Sigma}_{\text{prior}})$, $P_{\boldsymbol{\theta}}(\mathcal{D}|\boldsymbol{\theta}) \propto \mathcal{N}(\boldsymbol{\theta}; \boldsymbol{\mu}_{\text{like}}, \boldsymbol{\Sigma}_{\text{like}})$. (We subsequently drop the $\boldsymbol{\theta}$ within the normal distribution to lighten notation.) We make a distinction between two forms of likelihood: *data likelihood*, which is defined on the domain of the target variable (the probability of the data given the parameters), and *parameter likelihood*, which is specified in parameter space (the likelihood of the parameters). (See definition 1.)

From Bayes' rule the posterior, also normal, is,

$$\mathcal{N}(\boldsymbol{\mu}_{\text{post}}, \boldsymbol{\Sigma}_{\text{post}}) \propto \mathcal{N}(\boldsymbol{\mu}_{\text{prior}}, \boldsymbol{\Sigma}_{\text{prior}})\mathcal{N}(\boldsymbol{\mu}_{\text{like}}, \boldsymbol{\Sigma}_{\text{like}}). \tag{4.1}$$

The MAP solution is simply the mean, $\boldsymbol{\theta}_{\text{MAP}} = \boldsymbol{\mu}_{\text{post}}$, available in closed form,

$$\boldsymbol{\theta}_{\text{MAP}} = \boldsymbol{\Sigma}_{\text{post}}\boldsymbol{\Sigma}_{\text{like}}^{-1}\boldsymbol{\mu}_{\text{like}} + \boldsymbol{\Sigma}_{\text{post}}\boldsymbol{\Sigma}_{\text{prior}}^{-1}\boldsymbol{\mu}_{\text{prior}}, \tag{4.2}$$

where $\boldsymbol{\Sigma}_{\text{post}} = (\boldsymbol{\Sigma}_{\text{like}}^{-1} + \boldsymbol{\Sigma}_{\text{prior}}^{-1})^{-1}$. In RMS we assume availability of a mechanism for returning $\boldsymbol{\theta}_{\text{MAP}}$, and are interested in injecting noise into eq. 4.2 so that a *distribution* of $\boldsymbol{\theta}_{\text{MAP}}$ solutions are produced, matching the true posterior distribution.

A practical choice of noise source is the mean of the prior, $\boldsymbol{\mu}_{\text{prior}}$, since a modeller has full control over this value. Let us replace $\boldsymbol{\mu}_{\text{prior}}$ with some noisy random variable, $\boldsymbol{\theta}_{\text{anc}}$. This is the same place as a hyperprior over $\boldsymbol{\mu}_{\text{prior}}$, though with a subtly different role. Denote

$\boldsymbol{f}_{\text{MAP}}(\boldsymbol{\theta}_{\text{anc}})$ a function that takes as input $\boldsymbol{\theta}_{\text{anc}}$ and returns the resulting MAP estimate,

$$\boldsymbol{f}_{\text{MAP}}(\boldsymbol{\theta}_{\text{anc}}) = \boldsymbol{\Sigma}_{\text{post}}\boldsymbol{\Sigma}_{\text{like}}^{-1}\boldsymbol{\mu}_{\text{like}} + \boldsymbol{\Sigma}_{\text{post}}\boldsymbol{\Sigma}_{\text{prior}}^{-1}\boldsymbol{\theta}_{\text{anc}}. \tag{4.3}$$

Accuracy of this procedure hinges on selection of an appropriate distribution for $\boldsymbol{\theta}_{\text{anc}}$, which we term the *anchor distribution*. The distribution that will produce the true posterior can be found by setting $\mathbb{E}[\boldsymbol{f}_{\text{MAP}}(\boldsymbol{\theta}_{\text{anc}})] = \boldsymbol{\mu}_{\text{post}}$ and $\mathbb{V}\text{ar}[\boldsymbol{f}_{\text{MAP}}(\boldsymbol{\theta}_{\text{anc}})] = \boldsymbol{\Sigma}_{\text{post}}$.

Theorem 1. *In order that, $P(\boldsymbol{f}_{MAP}(\boldsymbol{\theta}_{anc})) = P(\boldsymbol{\theta}|\mathcal{D})$, the required distribution of $\boldsymbol{\theta}_{anc}$ is also multivariate normal, $P(\boldsymbol{\theta}_{anc}) = \mathcal{N}(\boldsymbol{\mu}_{anc}, \boldsymbol{\Sigma}_{anc})$, where,*

$$\boldsymbol{\mu}_{\text{anc}} = \boldsymbol{\mu}_{\text{prior}} \tag{4.4}$$

$$\boldsymbol{\Sigma}_{\text{anc}} = \boldsymbol{\Sigma}_{\text{prior}} + \boldsymbol{\Sigma}_{\text{prior}}\boldsymbol{\Sigma}_{\text{like}}^{-1}\boldsymbol{\Sigma}_{\text{prior}}. \tag{4.5}$$

Figure 4.2 provides a demonstration of the RMS algorithm in 2D parameter space.

4.2.2 Comparison to Prior Work

Previous work on RMS (Lu and Van Roy, 2017; Osband et al., 2019) was motivated via linear regression. Noting that the MAP solution is given by,

$$\boldsymbol{\theta}_{\text{MAP}} = (\frac{1}{\sigma_\epsilon^2}\mathbf{X}^T\mathbf{X} + \boldsymbol{\Sigma}_{\text{prior}}^{-1})^{-1}(\frac{1}{\sigma_\epsilon^2}\mathbf{X}^T\mathbf{y} + \boldsymbol{\Sigma}_{\text{prior}}^{-1}\boldsymbol{\mu}_{\text{prior}}), \tag{4.6}$$

these works added Gaussian noise to $\boldsymbol{\mu}_{\text{prior}}$, *in addition to* adding noise to $\mathbf{y}$, either by additive Gaussian noise or bootstrapping. Eq. 4.6 is a special case of our own derivation, substituting $\boldsymbol{\Sigma}_{\text{like}}^{-1} = 1/\sigma_\epsilon^2\mathbf{X}^T\mathbf{X}$ into eq. 4.2.

4.2.3 Practical Workaround: General Case

The previous section showed how to design an RMS procedure that will precisely recover the true Bayesian posterior. Unfortunately, in eq. 4.5 one must specify the parameter likelihood covariance in order to set the anchor distribution. For most models, this is infeasible.

A practical workaround is to simply ignore the second term in eq. 4.5 and set $\boldsymbol{\Sigma}_{\text{anc}} := \boldsymbol{\Sigma}_{\text{prior}}$. Using RMS with this anchor distribution will not generally lead to an exact recovery of the true posterior, however the resulting distribution can be considered an approximation of it. Corollary 1.1 derives this RMS approximate posterior in terms of the true posterior.

Corollary 1.1. *Set $\boldsymbol{\mu}_{anc} := \boldsymbol{\mu}_{prior}$ and $\Sigma_{anc} := \Sigma_{prior}$. The RMS approximate posterior is $P(\boldsymbol{f}_{MAP}(\boldsymbol{\theta}_{anc})) = \mathcal{N}(\boldsymbol{\mu}_{post}, \Sigma_{post}\Sigma_{prior}^{-1}\Sigma_{post})$.*

Proof sketch. This follows similar working to theorem 1, but instead of enforcing $\mathbb{E}[\boldsymbol{f}_{\text{MAP}}(\boldsymbol{\theta}_{\text{anc}})] = \boldsymbol{\mu}_{\text{post}}, \mathbb{V}\text{ar}[\boldsymbol{f}_{\text{MAP}}(\boldsymbol{\theta}_{\text{anc}})] = \Sigma_{\text{post}}$ and solving for $\boldsymbol{\mu}_{\text{anc}}, \Sigma_{\text{anc}}$, we now enforce $\boldsymbol{\mu}_{\text{anc}} := \boldsymbol{\mu}_{\text{prior}}$, $\Sigma_{\text{anc}} := \Sigma_{\text{prior}}$ and solve for $\mathbb{E}[\boldsymbol{f}_{\text{MAP}}(\boldsymbol{\theta}_{\text{anc}})], \mathbb{V}\text{ar}[\boldsymbol{f}_{\text{MAP}}(\boldsymbol{\theta}_{\text{anc}})]$.

This corollary shows that the means of the two distributions are aligned, although the covariances are not. Next we state several results quantifying how the RMS approximate posterior covariance differs compared to the true posterior covariance. All results assume multivariate normal prior and parameter likelihood. They can be observed in figure 4.3 (A).

Lemma 1.1. *When $\boldsymbol{\mu}_{anc} := \boldsymbol{\mu}_{prior}$, $\Sigma_{anc} := \Sigma_{prior}$ the RMS approximate posterior will in general underestimate the marginal variance compared to the true posterior, $\mathbb{V}ar[\boldsymbol{f}_{MAP}(\theta_{anc})] < \mathbb{V}ar[\theta|\mathcal{D}]$.*

Proof sketch. $\Sigma_{\text{post}}\Sigma_{\text{prior}}^{-1}\Sigma_{\text{post}}$ can be rearranged as $\Sigma_{\text{post}} - \Sigma_{\text{post}}\Sigma_{\text{like}}^{-1}\Sigma_{\text{post}}$. The second term will be positive definite, so the diagonal entry is positive, and hence, $\text{diag}(\Sigma_{\text{post}} - \Sigma_{\text{post}}\Sigma_{\text{like}}^{-1}\Sigma_{\text{post}})_i < \text{diag}(\Sigma_{\text{post}})_i$.

Lemma 1.2. *Additionally assume the prior is isotropic. When $\boldsymbol{\mu}_{anc} := \boldsymbol{\mu}_{prior}$, $\Sigma_{anc} := \Sigma_{prior}$ the eigenvectors (or 'orientation') of the RMS approximate posterior equal those of the true posterior.*

Proof sketch. $\Sigma_{\text{post}}\Sigma_{\text{prior}}^{-1}\Sigma_{\text{post}} = 1/\sigma_{prior}^2 \Sigma_{\text{post}}^2$. Squaring a matrix only modifies eigenvalues not eigenvectors. As does multiplying by a constant.

Theorem 2. *Additionally assume the prior is isotropic. For a two parameter model, when $\boldsymbol{\mu}_{anc} := \boldsymbol{\mu}_{prior}, \Sigma_{anc} := \Sigma_{prior}$, the RMS approximate posterior will in general overestimate the magnitude of the true posterior parameter correlation coefficient, $|\rho|$. If $|\rho| = 1$, then it will recover it precisely.*

Proof sketch. We compute the individual entries resulting from the required 2×2 matrix multiplications.

We were unable to generalise theorem 2 beyond a two parameter model, but numerical examples (appendix A.3) suggest that it holds for higher dimensionality.

4.2.4 Practical Workaround: Special Cases

Having described the covariance bias that in general will be present in the RMS approximate posterior, we now give two special conditions under which there is no bias, and the true posterior is exactly recovered. Illustrations of these cases are shown in figure 4.3 (B, C).

Theorem 3. *For extrapolation parameters (def. 2 - parameters which do not affect data likelihood but may affect new predictions) of a model, setting $\boldsymbol{\mu}_{anc} := \boldsymbol{\mu}_{prior}$, $\Sigma_{anc} := \Sigma_{prior}$, means the marginal RMS approximate posterior equals that of the marginal true posterior.*

Proof sketch. We show that the required matrix multiplications, $\Sigma_{\text{post}}\Sigma_{\text{prior}}^{-1}\Sigma_{\text{post}}$, do not affect rows corresponding to extrapolation parameters. Note these parameters remain at their prior.

Theorem 4. *Set $\boldsymbol{\mu}_{anc} := \boldsymbol{\mu}_{prior}, \Sigma_{anc} := \Sigma_{prior}$. The RMS approximate posterior will exactly equal the true posterior, Σ_{post}, when all eigenvalues of a scaled version of Σ_{post} (scaled such that the prior equals the identity matrix) are equal to either* 0 *or* 1. *This corresponds to posteriors that are a mixture of perfectly correlated and perfectly uncorrelated parameters.*

Proof sketch. We are searching for solutions to $\Sigma_{\text{post}} = \Sigma_{\text{post}}\Sigma_{\text{prior}}^{-1}\Sigma_{\text{post}}$. Applying a scaling, $\Sigma'_{\text{post}} = \Sigma_{\text{prior}}^{-1/2}\Sigma_{\text{post}}\Sigma_{\text{prior}}^{-1/2}$, results in a slightly simpler equation to find a solution to, $\Sigma'_{\text{post}} = \Sigma'^{2}_{\text{post}}$. Results for idempotent matrices tell us that if Σ'_{post} is singular with all eigenvalues equal to 0 or 1, this will be a solution.

To provide intuition behind theorem 4 consider a two parameter model. If parameters are perfectly correlated, the effect on the data likelihood of an increase in the first can be exactly compensated for by a change in the second. If the region over which this applies is large relative to the prior, the likelihood is a line of negligible width. This leads to a posterior of negligible width spanning the prior. Figure 4.3 [B] illustrates this. Examples in appendix A.3 show what combinations of parameters this holds for.

This section's proofs show that if these two conditions exist, RMS makes a precise recovery. In practice, one would expect to see an increasingly accurate RMS approximation as these conditions are approached.

4.3 RMS for Neural Networks

We now apply RMS with practical workaround to NNs. We will refer to this as 'anchored ensembling'.

First, we define the NN loss function to be optimised that corresponds to RMS. We then discuss the validity of the RMS procedure in the context of NNs, given the assumptions made. Finally we consider some matters arising in implementation of the scheme. Algorithm 1 details the full procedure.

4.3.1 Loss Function

Consider a NN containing parameters, $\boldsymbol{\theta}$, making predictions, $\hat{\mathbf{y}}$, with H hidden nodes and N data points. If the prior is given by $P(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\mu}_{\text{prior}}, \boldsymbol{\Sigma}_{\text{prior}})$, maximising the following returns MAP parameter estimates. (See appendix A.2 for the standard derivation.)

$$\boldsymbol{\theta}_{\text{MAP}} = \text{argmax}_{\boldsymbol{\theta}} \log(P_{\mathcal{D}}(\mathcal{D}|\boldsymbol{\theta})) - \frac{1}{2}||\boldsymbol{\Sigma}_{\text{prior}}^{-1/2} \cdot (\boldsymbol{\theta} - \boldsymbol{\mu}_{\text{prior}})||_2^2 \tag{4.7}$$

Some readers may be more familiar with the form when $\boldsymbol{\mu}_{\text{prior}} = \mathbf{0}$, and there is a single regularisation constant, λ, equal for all layers.

$$\boldsymbol{\theta}_{\text{MAP}} = \text{argmax}_{\boldsymbol{\theta}} \log(P_{\mathcal{D}}(\mathcal{D}|\boldsymbol{\theta})) - \lambda||\boldsymbol{\theta}||_2^2 \tag{4.8}$$

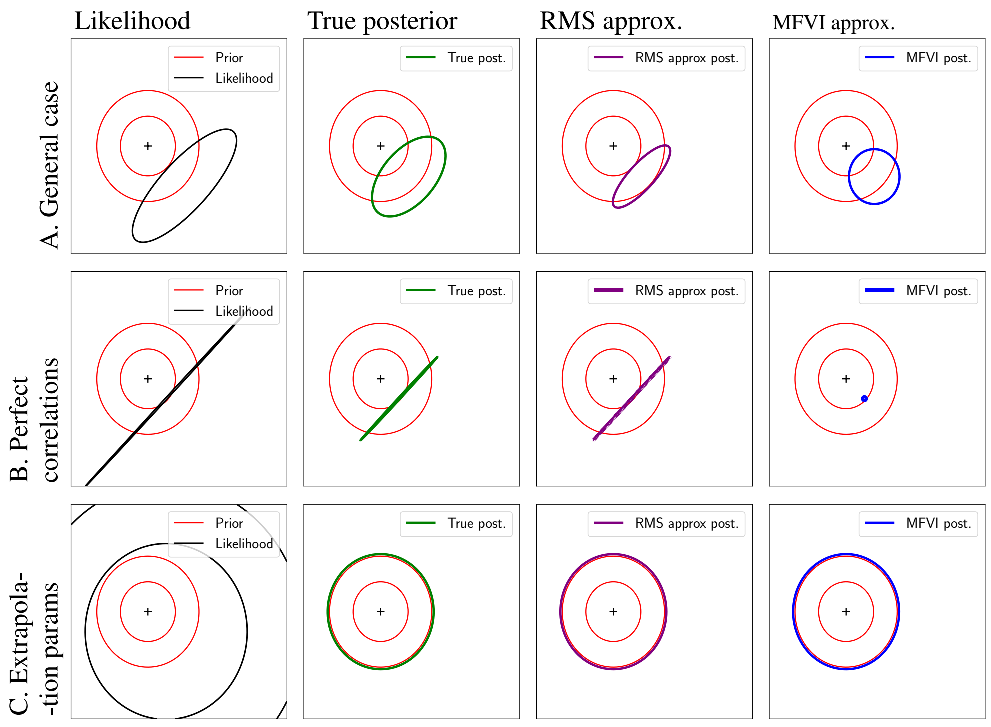


Figure 4.3 Examples of the RMS approximate posterior when $\boldsymbol{\mu}_{\text{anc}} := \boldsymbol{\mu}_{\text{prior}}, \boldsymbol{\Sigma}_{\text{anc}} := \boldsymbol{\Sigma}_{\text{prior}}$. MFVI also shown.

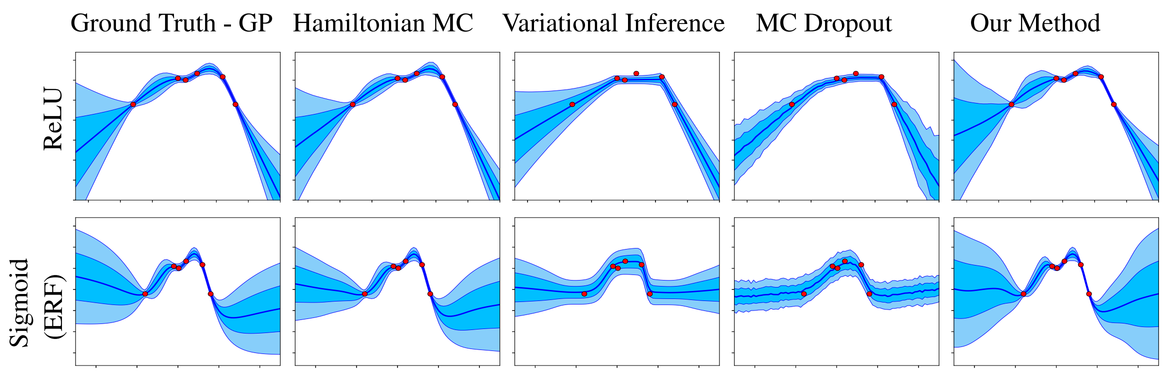


Figure 4.4 Predictive distributions produced by various inference methods (columns) with varying activation functions (rows) in single hidden layer NNs on a toy regression task.

Eq. 4.8 is standard L2 regularisation, however, in order to apply RMS, we need the more flexible form in eq. 4.7. We then replace $\boldsymbol{\mu}_{\text{prior}}$ with some random variable $\boldsymbol{\theta}_{\text{anc}}$. To use the practical form of RMS, we will draw $\boldsymbol{\theta}_{\text{anc}} \sim \mathcal{N}(\boldsymbol{\mu}_{\text{prior}}, \boldsymbol{\Sigma}_{\text{prior}})$.

Conveniently, no parametric form of data likelihood has yet been specified. For a regression task assuming homoskedastic Gaussian noise of variance σ_ϵ^2, MAP estimates are found by minimising,

$$\text{Loss}_j = \frac{1}{N}||\mathbf{y} - \hat{\mathbf{y}}_j||_2^2 + \frac{1}{N}||\boldsymbol{\Gamma}^{1/2} \cdot (\boldsymbol{\theta}_j - \boldsymbol{\theta}_{anc,j})||_2^2. \tag{4.9}$$

We have defined a diagonal regularisation matrix, $\boldsymbol{\Gamma}$, where the i^{th} diagonal element is the ratio of data noise of the target variable to prior variance for parameter θ_i, $\text{diag}(\boldsymbol{\Gamma})_i = \sigma_\epsilon^2/\sigma_{prior_i}^2$. Note a subscript has been introduced, $j \in \{1...M\}$, with the view of an ensemble of M NNs, each with a distinct draw of $\boldsymbol{\theta}_{\text{anc}}$.

For classification tasks, cross entropy is normally maximised, which assumes a multinomial data likelihood,

$$\text{Loss}_j = -\frac{1}{N}\sum_{n=1}^{N}\sum_{c=1}^{C} y_{n,c}\log \hat{y}_{n,c,j} + \frac{1}{N}||\boldsymbol{\Gamma}^{1/2} \cdot (\boldsymbol{\theta}_j - \boldsymbol{\theta}_{anc,j})||_2^2, \tag{4.10}$$

where y_c is the label for class $c \in \{1...C\}$. Here, $\text{diag}(\boldsymbol{\Gamma})_i = 1/2\sigma_{prior_i}^2$.

4.3.2 Validity of RMS in NNs

Theory derived to motivate and analyse RMS assumed a simplified setting of multivariate normal parameter likelihoods. This section discusses this assumption, then considers the prevalence of special conditions (section 4.2.4) that would lead to a close approximation of the true posterior.

Normal Distribution

Earlier proofs assumed parameter likelihoods follow a multivariate normal distribution. We provide two justifications for using this assumption in NNs.

1) Other approximate Bayesian methods incorporate similar assumptions into their methodologies. MFVI commonly fits a factorised normal distribution to the posterior. The Laplace approximation fits a multivariate normal distribution to the mode of a MAP solution.

2) In figure 4.5 we visualise conditional parameter likelihoods for actual NNs trained on regression and classification tasks. After training, a parameter is randomly selected, and all others are frozen. The chosen parameter is varied over a small range and the data likelihood calculated at each point. Hence conditional distributions may be plotted. The plots suggest that thinking of local modes as approximately normally distributed is not unreasonable for the purpose of analysis.

This justifies modelling a single mode of the parameter likelihood as multivariate normal. However, the parameter space of a NN is likely to contain many such modes, with each member of an anchored ensemble ending up at a different one. We believe that many of these modes would be exchangeable, for example arising from parameter symmetries. In this case we believe that MAP solutions would also be exchangeable.

Empirically we did not observe this multimodality being problematic - plots such as figure 4.8 show predictive posteriors with low bias compared to the true posterior.

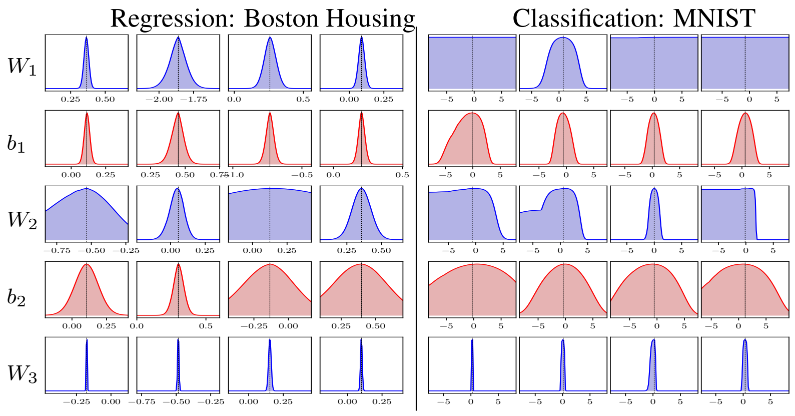


Figure 4.5 Empirical plots of conditional likelihoods for 4 randomly sampled parameters in two-layer NNs.

Presence of Special Cases

Setting the anchor distribution equal to the prior leads to an RMS approximate posterior that, in general, has underestimated variance and overestimated correlation.

Figures 4.4, 4.7 & 4.8 show predictive distributions for anchored ensembles that very closely approximate the true Bayesian predictive posterior, with little sign of bias. This demands an answer to why, rather than if, anchored ensembling performs such accurate inference in these examples. We believe the reason is the presence of the two special conditions that can lead to exact recovery.

It should be straightforward to see that extrapolation parameters (definition 2) exist in the figures. Many hidden nodes will be dead across the range which contains data. Their corresponding final layer weight then has no effect on the data likelihood, but they do affect predictions outside of the training data.

It is more difficult to see that perfect correlations also exist, and we provide a numerical example illustrating this shortly. Essentially it relies on two hidden nodes becoming live in between the same two data points. The associated final layer weights are then perfectly correlated. Whether these special conditions exist beyond fully-connected NNs is something tested indirectly in later experiments with CNNs.

One obvious way to further encourage these conditions is to increase the width of the NN, creating more parameters and an increasing probability of strong correlations. See also a study of multicollinearity in NNs (Cheng et al., 2018) [7.1].

Example of Perfectly Correlated Parameters

Here, we provide a concrete example of perfectly correlated parameters in a NN, and show that anchored ensembling *does* recover the true posterior for these parameters when $\Sigma_{\text{anc}} = \Sigma_{\text{prior}}$.

We consider finding the posterior of final layer weights in a small single layer ReLU NN of two hidden nodes for a regression problem with two data points. (Choosing the final layer weights for our analysis allows analytical equations associated with linear regression to be used, simplifying our analysis, though these results would also apply to the first layer weights and biases.)

We design the problem such that the point where both hidden nodes become greater than zero (the elbow points of ReLU units) falls in between the two data points, and the active half of the output is also shared, so that the final layer weights are perfectly correlated.

Figure 4.6 illustrates our set up. Data points and NN parameters are as follows,

$\mathcal{D} = \{x_1 = -5, y_1 = 0; x_2 = 5, y_2 = 0\}$, $\sigma_\epsilon^2 = 0.01$,

$\mathbf{W}_1 = [-0.8, -0.4], \mathbf{b}_1 = [-1, 0.1], \hat{y} = \mathbf{W}_2^T \max(x\mathbf{W}_1 + \mathbf{b}_1, 0)$

We set prior means to zero, with isotropic covariance according to $1/H$,

$$\boldsymbol{\Sigma}_{\text{prior}} = \begin{bmatrix} 0.5 & 0.0 \\ 0.0 & 0.5 \end{bmatrix}$$

We print out matrices of interest,

$$\boldsymbol{\Sigma}_{\text{like}} = \begin{bmatrix} -6.89e+13 & 9.85e+13 \\ 9.85e+13 & -1.40e+14 \end{bmatrix}$$

$$\boldsymbol{\Sigma}_{\text{like}}^{-1} = \begin{bmatrix} 90.0 & 63.0 \\ 63.0 & 44.1 \end{bmatrix}$$

$$\boldsymbol{\Sigma}_{\text{post}} = \begin{bmatrix} 0.169 & -0.231 \\ -0.231 & 0.338 \end{bmatrix}$$

$$\boldsymbol{\Sigma}_{\text{post}} \boldsymbol{\Sigma}_{\text{prior}}^{-1} \boldsymbol{\Sigma}_{\text{post}} = \begin{bmatrix} 0.166 & -0.235 \\ -0.235 & 0.338 \end{bmatrix}$$

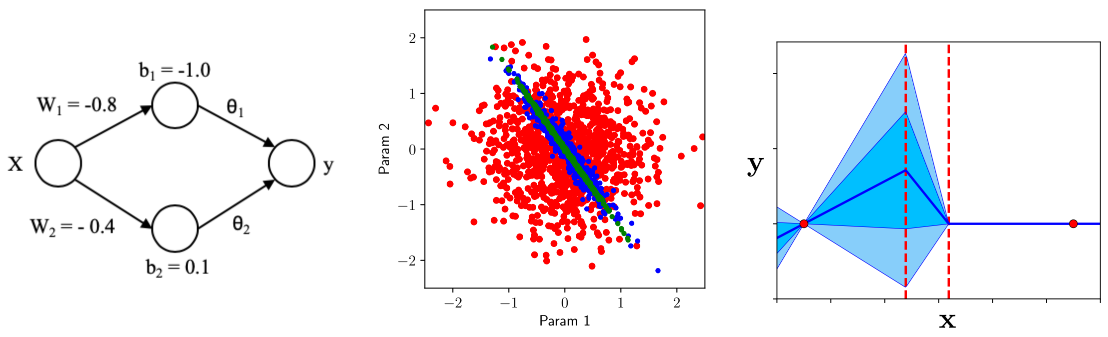


Figure 4.6 Left: Single layer NN of two hidden nodes. Middle: Draws in parameter space for prior (red), analytical posterior (blue) and anchored posterior (green). Right: Posterior predictive distribution - dashed red lines are elbows of ReLU units.

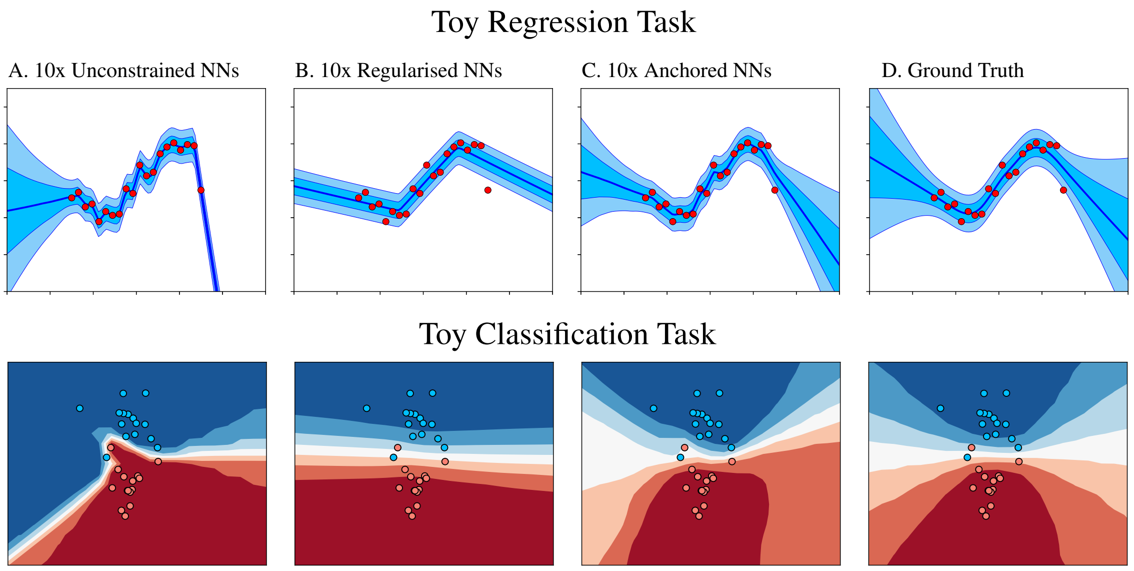


Figure 4.7 Comparison of NN ensemble loss choices.

The anchored ensembling posterior, $\Sigma_{\text{post}}\Sigma_{\text{prior}}^{-1}\Sigma_{\text{post}}$, provides a close approximation of the true posterior covariance (and would be exact discounting numerical issues). Note the similarity of the posterior in figure 4.6 (middle) with the perfect correlations example shown in figure 4.3 (B).

4.3.3 Implementation Practicalities

This section considers some decisions faced during implementation. Algorithm 1 details the full anchored ensembling procedure.

How many NNs to use in an RMS ensemble? A large number of samples (and therefore NNs) would be required to fully capture the posterior parameter distributions. By contrast, if one thinks of each NN as an iid sample from a posterior *predictive* distribution, a much smaller number are required, given output dimensionality is typically small. Note this is unaffected by input dimension. Our experiments in section 4.4 used 5-10 NNs per ensemble, delivering good performance on tasks ranging from 1-10 outputs. See also figure 4.8. This results in anchored ensembles scaling by $\mathcal{O}(MN)$.

Should the NNs be initialised at anchor points? It is convenient to draw parameter initialisations from the anchor distribution, and regularise directly around these initialised values, however, we found decoupling initialisations from anchor points benefited experiments.

4.4 Experiments

This section shares high-level findings from experiments. Further details and hyperparameter settings are given in appendix A.5. Code is available online (github/TeaPearce). Also see our interactive demo (teapearce.github.io).

4.4.1 Qualitative Tests

We first examine anchored ensembles on toy problems to gain intuition about its behaviour compared to popular approximate inference and ensembling methods.

Figure 4.4 compares popular Bayesian inference methods in single-layer NNs (100 hidden nodes) for ReLU and sigmoidal non-linearities. GP and HMC produce 'gold standard' Bayesian inference, and we judge the remaining methods, which are scalable approximations, to them. Both MFVI (with a factorised normal distribution) and MC dropout do a poor job of capturing interpolated uncertainty. This is a symptom of the posterior approximation ignoring parameter correlations - see also figure 4.3 which shows MFVI failing to capture correlations in the posterior. This was explored in Foong et al. (2019).

Figure 4.7 contrasts anchored ensembles trained on eq. 4.9 & 4.10, with NN ensembles using standard loss functions, either with no regularisation term ('unconstrained', $\mathbf{\Gamma} = \mathbf{0}$), or regularised around zero ('regularised', $\boldsymbol{\theta}_{anc,j} = \mathbf{0}$). Regularised produces poor results since it encourages all NNs to the same single solution and diversity is reduced – note that the randomisation of initialisation and optimiser does not help in this toy example. Unconstrained (most similar to deep ensembles) is also inappropriate - although it produces diversity, no notion of prior is maintained and it overfits the data.

Figure 4.8 shows the predictive distribution improving with number of NNs compared to a ReLU GP, however it appears a small residual difference remains. Single-layer NNs were used with 100 hidden nodes.

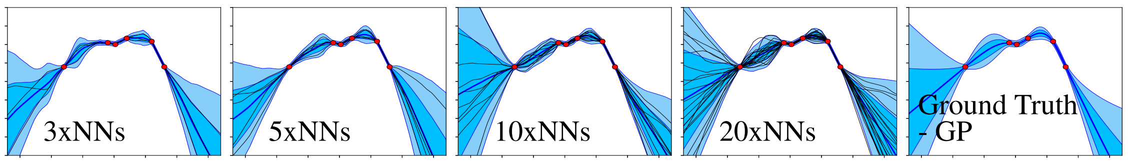


Figure 4.8 The predictive distribution of an anchored ensemble approaches that of a ReLU GP.

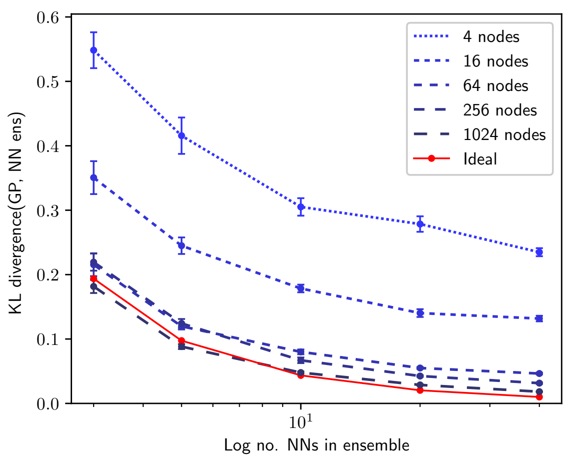


Figure 4.9 Difference in predictive distributions of an anchored ensemble and a ReLU GP as a function of width and number of NNs. Mean ± 1 standard error.

4.4.2 Convergence Behaviour

To assess how precisely anchored ensembling performs Bayesian inference on a real dataset, we compared its predictive distribution with that of an exact method (ReLU GP) on the Boston housing dataset. KL divergence between the predictive distribution of the ensemble and GP was measured - zero represents identical distributions. Both models used 50% of the data for training and the other 50% for testing. Results were averaged over 10 runs, randomly shuffling the test/train split for each run. The data noise variance was fixed for both models, $\sigma_\epsilon^2 = 0.1$. Data preprocessing was done similarly to the UCI regression protocol.

Figure 4.9 quantifies the difference when varying both the width of the NN, and number of NNs in the ensemble. The 'ideal' line shows the metric when posterior samples from the GP itself, rather than anchored NNs, were used. KL divergence between the two predictive distributions decreases as both NN width and number of NNs is increased. A small amount of residual difference remains even for 40xNNs of $1,024$ nodes.

4.4.3 UCI Regression Benchmarks

In order to compare anchored ensembles against popular approximate inference methods, we used a standard BNN benchmark. This assesses uncertainty quality for UCI regression tasks on data drawn from the same distribution as the training data (Hernández-Lobato and Adams, 2015). We also implemented the ReLU GP to assess the performance limit on these datasets.

The goal of the benchmark is to minimise negative log likelihood (NLL) and RMSE for each dataset. Models are trained on 90% of data, with RMSE and NLL recorded on the remaining

Table 4.1 NLL regression benchmark results. See appendix A.4 for RMSE (table A.1) and variants of our method. Mean ± 1 standard error.

	$\hat{\sigma}_\epsilon^2$	Deep Ens. *State-Of-Art*	Anch. Ens. *Our Method*	ReLU GP[1] *Gold Standard*
High Epistemic Uncertainty				
Energy	1e-7	1.38 ± 0.22	**0.96 ± 0.13**	0.86 ± 0.02
Naval	1e-7	-5.63 ± 0.05	**-7.17 ± 0.03**	-10.05 ± 0.02
Yacht	1e-7	1.18 ± 0.21	**0.37 ± 0.08**	0.49 ± 0.07
Equal Epistemic & Aleatoric Uncertainty				
Kin8nm	0.02	-1.20 ± 0.02	**-1.09 ± 0.01**	-1.22 ± 0.01
Power	0.05	**2.79 ± 0.04**	2.83 ± 0.01	2.80 ± 0.01
Concrete	0.05	3.06 ± 0.18	**2.97 ± 0.02**	2.96 ± 0.02
Boston	0.08	**2.41 ± 0.25**	2.52 ± 0.05	2.45 ± 0.05
High Aleatoric Uncertainty				
Protein	0.5	**2.83 ± 0.02**	2.89 ± 0.01	*2.88 ± 0.00
Wine	0.5	**0.94 ± 0.12**	**0.95 ± 0.01**	0.92 ± 0.01
Song	0.7	**$3.35 \pm$ NA**	$3.60 \pm$ NA	**$3.62 \pm$ NA

[1] For comparison only (not a scalable method). * Trained on $10,000$ rows of data. ** Trained on $20,000$ rows of data, tested on $5,000$ data points.

10%. Experiments are repeated 20 times with random test/train splits, using single-layer NNs of 50 hidden nodes. The larger datasets, Protein and Song, allow 100 hidden nodes, and are repeated only five and one time respectively.

The full hyperparameter tuning process is given in the appendix. Briefly, a single split train/validation split of 80%/20% was used. We used a grid search for both prior variance and data noise. The anchored ensembles were implemented as homoskedastic, i.e. with a single output per NN.

Table 4.1 lists our results. We include results reported for Deep Ensembles (Lakshminarayanan et al., 2017), which is considered the state-of-the-art ensemble method. Not that their implementation used two outputs per NN allowing heteroskedastic noise to be modelled. Appendix A.4 provides a full comparison with other approximate Bayesian methods including Probabilistic Backpropagation, MC Dropout, and Stochastic Gradient HMC.

Ordering results according to the level of estimated data noise, $\hat{\sigma}_\epsilon^2$, shows a clear pattern - anchored ensembles perform best in datasets with low data noise, surpassing both Deep Ensembles and many approximate inference methods listed in appendix A.4. This may be due to an increased importance of interpolation uncertainty when data noise is low, which anchored ensembles models well. On other datasets, the method is also competitive (the Deep Ensemble implementation used additional complexity to capture heteroskedastic uncertainty and has an advantage on higher data noise datasets).

Algorithm 1 Implementing anchored ensembles of NNs

```
Input: Training data, $\mathbf{X}$ & $\mathbf{Y}$, test data point, $\mathbf{x}^*$, prior mean and covariance, $\boldsymbol{\mu}_{\text{prior}}$, $\boldsymbol{\Sigma}_{\text{prior}}$, ensemble size, $M$, data noise variance estimate, $\hat{\sigma}_\epsilon^2$ (regression only).
Output: Estimate of mean and variance, $\hat{\mathbf{y}}$, $\hat{\sigma}_y^2$ for regression, or class probabilities, $\hat{\mathbf{y}}$ for classification.

# Set regularisation matrix
$\boldsymbol{\Gamma} \Leftarrow \hat{\sigma}_\epsilon^2 \boldsymbol{\Sigma}_{\text{prior}}^{-1}$ (regression)   OR   $\boldsymbol{\Gamma} \Leftarrow \frac{1}{2}\boldsymbol{\Sigma}_{\text{prior}}^{-1}$ (classification)

# Create ensemble
$\boldsymbol{\mu}_{\text{anc}} \Leftarrow \boldsymbol{\mu}_{\text{prior}}, \boldsymbol{\Sigma}_{\text{anc}} \Leftarrow \boldsymbol{\Sigma}_{\text{prior}}$
for $j = 1$ to $M$
    $\boldsymbol{\theta}_{anc,j} \sim \mathcal{N}(\boldsymbol{\mu}_{\text{anc}}, \boldsymbol{\Sigma}_{\text{anc}})$ # Sample anchor points
    $NN_j$.create($\boldsymbol{\Gamma}, \boldsymbol{\theta}_{anc,j}$) # Create custom regulariser
    $NN_j$.initialise() # Initialisations independent of $\boldsymbol{\theta}_{anc,j}$

# Train ensemble
for $j = 1$ to $M$
    $NN_j$.train($\mathbf{X}, \mathbf{Y}$), loss in eq. 4.9 (regression) or eq. 4.10 (classification) or eq. 4.7 (custom)

# Predict with ensemble
for $j = 1$ to $M$
    $\hat{\mathbf{y}}_j \Leftarrow NN_j$.predict($\mathbf{x}^*$)

# Regression - combine ensemble estimates
$\hat{\mathbf{y}} = \frac{1}{M}\sum_{j=1}^{M} \hat{\mathbf{y}}_j$, # Mean prediction
$\hat{\sigma}_{model}^2 = \frac{1}{M-1}\sum_{j=1}^{M}(\hat{y}_j - \hat{y})^2$ # Epistemic var.
$\hat{\sigma}_y^2 = \hat{\sigma}_{model}^2 + \hat{\sigma}_\epsilon^2$ # Total var. = epistemic + data noise

# Classification - combine ensemble estimates
$\hat{\mathbf{y}} = \frac{1}{M}\sum_{j=1}^{M} \hat{\mathbf{y}}_j$, # Average softmax output
$\hat{\sigma}_y^2$ = None # N/A for classification

return $\hat{\mathbf{y}}$, $\hat{\sigma}_y^2$
```

4.4.4 Out-of-Distribution Classification

We now test on classification tasks, for out-of-distribution (OOD) data, with complex NN architectures, and compare against other ensemble methods.

An uncertainty-aware NN should make predictions of decreasing confidence as it is asked to predict on data further from the distribution seen during training. To test this, we report the proportion of high confidence predictions (defined as a softmax output class being $\geq 90\%$) made by various ensemble systems - unconstrained (similar to deep ensembles), regularised, and our proposed anchored scheme (as in section 4.4.1). We consider unconstrained and regularised to be baselines we are comparing the anchored method with.

We trained on three different datasets, using a NN architecture appropriate to each: 1) Fashion MNIST image classification; 3 fully-connected layers of 100 hidden nodes. 2) IMDb movie review sentiment classification; embedding (20 dimensions) + 1D convolution (50 filters, kernel size of 3) + fully-connected layer (200 hidden nodes). 3) CIFAR-10 image classification; convolutional NN (CNN) similar to VGG-13 (9 million parameters, 64-64-maxpool-128-128-maxpool-256-256-256-maxpool-512-512-512-maxpool-flatten-2048-softmax). Experiments were repeated five times for Fashion MNIST and IMDb, and three times for CIFAR-10.

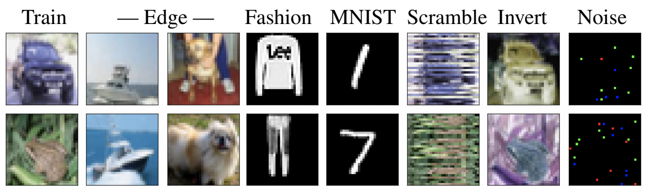
Note our accuracies remain below that of state-of-the-art since we do not use tricks such as data augmentation and batch normalisation.

The confidence of predictions on novel data categories not seen during training was assessed. Table 4.2 shows example OOD inputs shown to the NNs. Edge refers to two classes held out during training (for CIFAR ships, dogs; for Fashion MNIST trousers, sneakers). Further details are deferred to the appendix.

The three tables show similar patterns. Whilst all methods predict with similar confidence on the training data, confidence differs greatly for other data categories, with anchored ensembles generally producing the most conservative predictions. This gap increases for data drawn further from the training distribution. Note that it should not be expected that a perfect method score zero on *all* OOD data categories - e.g. sneakers can look forgivably like boots. Encouragingly, we observe similar (though less extreme) behaviour to that in the toy examples of figure 4.7.

Table 4.2 Proportion of predictions that were high confidence on out-of-distribution data, e.g. a single regularised VGG-13 CNN trained on CIFAR-10 made high confidence predictions 54% of the time when asked to predict on MNIST. Mean over five runs (three for CIFAR). 'Reg.' refers to standard L2 regularisation, 'Uncons.' refers to no regularisation, 'Anch.' refers to the proposed anchored regularisation, 'nxNNs' refers to an ensemble of n NNs.

CIFAR-10 Image Classification, CNN



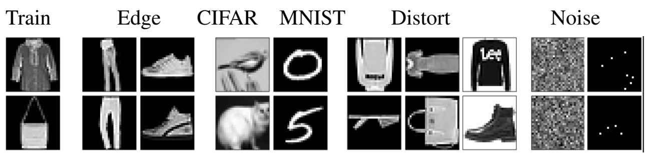
	Accuracy	Train	Edge	Fashion	MNIST	Scramble	Invert	Noise
6 layer CNN, 0.3 mil params								
1xNNs Reg.	80.1%	0.502	0.196	0.388	0.506	0.228	0.121	0.853
5xNNs Uncons.	81.7%	0.563	0.159	0.239	**0.105**	0.096	0.071	0.307
5xNNs Reg.	81.9%	0.461	0.176	0.326	0.303	0.179	0.084	0.223
5xNNs Anch.	81.8%	0.442	**0.132**	**0.192**	0.138	**0.062**	**0.059**	**0.182**
10xNNs Anch.	82.6%	0.43	0.103	0.194	0.115	0.049	0.046	0.082
VGG-13 CNN, 8.7 mil params								
1xNNs Reg.	81.6%	0.671	0.466	0.440	0.540	0.459	0.324	0.948
5xNNs Uncons.	85.0%	0.607	0.330	0.208	0.275	0.175	0.209	0.380
5xNNs Reg.	86.1%	0.594	0.296	0.219	0.188	**0.106**	0.153	0.598
5xNNs Anch.	85.6%	0.567	**0.258**	**0.184**	**0.149**	0.134	**0.136**	**0.118**
10xNNs Anch.	86.0%	0.549	0.256	0.119	0.145	0.122	0.124	0.161

IMDb Text Sentiment Classification, Embedding+CNN

Train: "this movie is the best horror movie bar none i love how stanley just . . . "
Reuters: "said it has started talks on the possible ... of the company with various..."
Rand. 1: "member member member member . . . "
Rand. 2: "twists mentally superb finest will dinosaur variety models stands knew"
Rand. 3: "computers towers bondage braveheart threatened rear triangle"

	Accuracy	Train	Reuters	Rand. 1	Rand. 2	Rand. 3
1xNNs Reg.	85.3%	0.637	0.119	0.153	0.211	0.326
5xNNs Uncons.	89.1%	0.670	0.102	0.141	0.100	0.075
5xNNs Reg.	87.1%	0.612	0.051	0.091	0.076	0.055
5xNNs Anch.	87.7%	0.603	**0.049**	**0.075**	**0.061**	**0.009**

Fashion MNIST Image Classification, Fully-Connected NN



	Accuracy	Train	Edge	CIFAR	MNIST	Distort	Noise
1xNN Reg.	86.8 %	0.660	0.584	0.143	0.160	0.429	0.364
5xNNs Uncons.	89.0 %	0.733	0.581	0.301	0.104	0.364	0.045
5xNNs Reg.	87.8 %	0.634	**0.429**	0.115	0.072	0.342	0.143
5xNNs Anch.	88.0 %	0.631	0.452	**0.065**	**0.041**	**0.246**	**0.006**

4.4.5 Reinforcement Learning

Uncertainty-Aware Model-Free RL

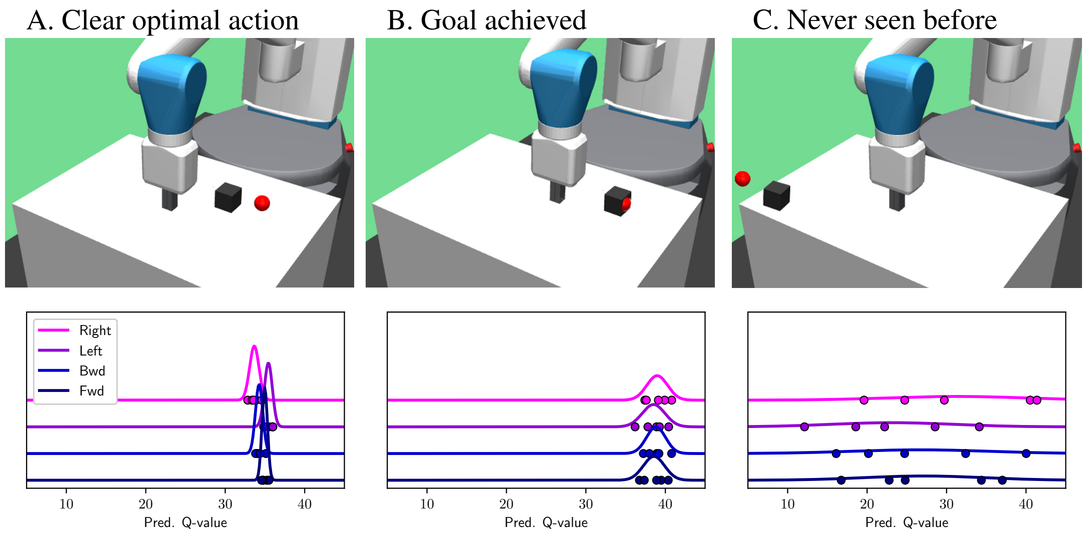


Figure 4.10 Anchored ensembling creates uncertainty-aware agents.

To test our method in a complex RL environment, an anchored ensemble (5xNNs, two hidden layers, 50 hidden nodes), was trained to complete a discretised version of FetchPush - an agent controls a robotic arm, with rewards received when a randomly placed cube (black) is pushed to a goal (red).

We used Bayesian Q-learning (Dearden et al., 1998), similar to regular Q-learning, but with Q-values modelled as *distributions* rather than point estimates - the wider the distribution, the less certain the agent. This is beneficial both to drive the exploration/exploitation process via Thompson sampling, and for identifying OOD examples.

Figure 4.10 shows the agent's awareness of its uncertainty. After training for $40,000$ episodes, its confidence over actions was plotted for three scenarios. In each case, the individual NN estimates are represented with five dots, with the ensemble's prediction modelled as a Gaussian distribution. Scenario A) Cube and goal are in positions often encountered during training, the agent has learnt that it must move the arm left - the narrow distributions with significantly different means reflect its confidence in this. B) The goal has already been achieved - narrow overlapping distributions with higher means. C) A peculiar goal position that has never been encountered - the broad distributions over all actions reflect its high uncertainty.

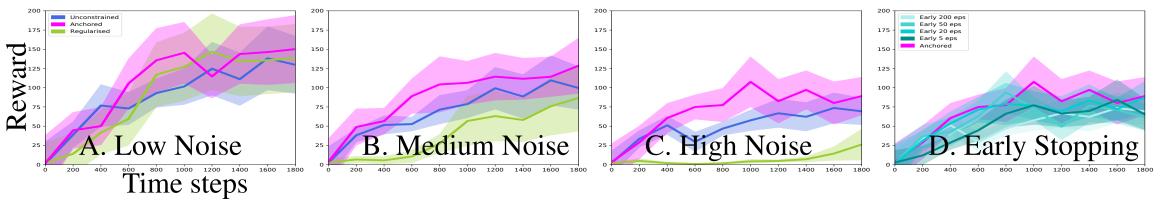


Figure 4.11 Anchored ensembling creates robust MBRL agents in noisy environments. Mean and standard error over five runs.

Model-Based RL in Noisy Environments

We tested the benefit of using an anchored ensemble in noisy RL environments. We modified the classic cartpole swingup environment such that different levels of stochastic noise could be added to the future state; consider a state action pair given by, s, a, and a noisy state, $\bar{s}$ such that $P(\bar{s}_{t+1}|a_t, s_t) = \mathcal{N}(s_{t+1}, \sigma_\epsilon^2)$.

The value of σ_ϵ^2 was given three settings: low, medium and high, $\sigma_\epsilon^2 \in \{0.001, 0.002, 0.005\}$. The task was learnt using a model-based RL approach similar to the heteroskedastic ensembles used by Chua et al. (2018). Fully-connected three-layer NNs learnt to predict the dynamics of the environment given some state and action. Planning was performed using the cross-entropy method, rolling out for a horizon of 25 steps, with 10 particles.

Figure 4.11 (A, B, C) shows learning curves for the three noise levels. All ensemble techniques perform similarly in the low noise setting. As noise is increased the overall performance of all methods drops. Anchored ensembles are most resiliant, followed by the unconstrained ensemble. In panel D, we compared against an unconstrained ensemble employing early stopping - this corresponds to the proposal that applying early stopping to an ensemble can produce approximate inference (Duvenaud et al., 2016). Whilst careful tuning did offer some improvement over the default setting, a performance gap to anchored ensembling remained.

4.5 Related Work

Section 2.4.3 provided a broad overview of methods for doing inference in BNNs. Here we provide a focused summary of work in ensembling, particularly applied to NNs for the purpose of uncertainty quantification.

Ensembling has most often been proposed as a method to improve *accuracy* over individual prediction models (Dietterich, 2000) (as opposed to for uncertainty quantification). Ensemble theory shows that if models are accurate individually, as well as diverse from one other, combining their predictions can lead to improved performance. Multiple proposals exist for how diversity can be achieved, such as manipulating training data for each model or randomising the learning algorithm. Breiman (1996) provides further analysis relating to model instability (how sensitive model predictions are to a change in training data).

Ensembling was applied to NNs to improve accuracy as early as 1990 (Hansen and Salamon, 1990). The justification being that each time a NN is optimised it finds a different minimum, which leads to a different way of generalising. The ensemble effectively averages across several plausible generalisation hypotheses. Two papers from 1996 proposed using ensembles of NNs to estimate uncertainty (Heskes, 1996; Tibshirani, 1996). These were both motivated by theory of the, then recently proposed, boostrap, with each NN trained on a different version of the dataset (sampled with replacement from the full training set).

Key to an ensemble's success is diversity between models. Two further early works proposed a method to explicitly encourage diversity between individual NNs, called 'negative correlation learning' (Brown, 2004; Liu and Yao, 1999), which optimises a loss augmented with a repulsive component, without requiring bootstrapping. This was explored only from the perspective of improving accuracy.

Interest in the area subsequently waned, and was only rekindled following the success of deep learning in the 2010's. Modern works revisited several of these early ideas. Deep ensembles (Lakshminarayanan et al., 2017), perhaps the most important modern work in this area, showed empirically that ensembling works well for uncertainty quantification for modern NN architectures and large datasets. The essence of the idea was similar to Heskes (1996), who combined ensembling with the output of mean and variance estimates, itself from Nix & Weigend (1994).

One notable difference was that deep ensembles removed the bootstrapping portion of the scheme which they found empirically harmed performance, finding randomising parameter initialisations was itself sufficient. This was also reported in a comprehensive empirical study of ensembles (Lee et al., 2015). This is interesting since bootstrapping was the original justification for using ensembling for uncertainty estimation (Heskes, 1996; Tibshirani, 1996) and some researchers continue to support bootstrapping as an integral part of the scheme (Osband et al., 2016, 2019; Osband and Van Roy, 2015). Theoretical explanations for the negative effect of bootstrapping remain something of an open question (Fort et al., 2019).

A further open question is whether the entire NN needs to be ensembled. Several works have proposed all models share a common set of early layers, with mutliple heads splitting off later in the NN (Lee et al., 2015; Osband et al., 2016). Justification is largely driven by empirical results.

Ensembles are generally presented as a non-Bayesian alternative to BNNs, which is the dominant approach to uncertainty in deep NNs. The absence of connection to Bayesian methodology has drawn critics and inhibited uptake - e.g. Gal (2016) [p. 27] - with ensembles viewed as a heuristic ad-hoc method. However, empirical results often favour ensembles of NNs over BNNs (Gustafsson et al., 2019; Lakshminarayanan et al., 2017; Osband et al., 2016; Tibshirani, 1996), and several works have studied similarities between the ensembles and BNNs. Duvenaud et al. (2016) interpreted training via maximum likelihood from random initialisations, with early stopping, as approximate inference. Fort et al. (2019) suggested that ensembles return point estimates of various modes of the posterior, which is arguably a more effective posterior approximation than modelling a single mode, as variational methods aim to do. Wilson & Izmailov (2020) also argued that ensembles do Bayesian model averaging, and proposed using ensembling to find multiple modes, and then fitting a posterior to each mode separately. Finally, the neural tangent kernel (Jacot et al., 2018) suggests dynamics of SGD in wide NNs produce individual NNs sampled from a specific posterior distribution.

One further work related to our proposal is Stein variational gradient descent (Liu and Wang, 2016); a Bayesian inference algorithm that employs gradient descent with a specific update rule. It was shown to minimise the KL divergence between a distribution represented by a set of M 'particles' and the true posterior. There are similarities with our RMS procedure; 1) particles are synonymous with ensembles, 2) both therefore consider an empirical distribution that may be optimised by gradient descent, 3) the update rule is similar to the loss we propose in eq. 4.7, a sum of the log likelihood and a randomised term (although their log likelihood is weighted). Studying deeper links between the two methods is an interesting avenue for future work.

4.6 Discussion & Conclusion

This chapter connected ensembling - a practical, scalable method for uncertainty quantification - to approximate Bayesian inference, by proposing, analysing, and testing a modification to the usual NN ensembling process; regularising parameters around values drawn from a prior distribution.

Under simplifying assumptions, we derived an abstracted form of RMS motivating this. We analysed a practical RMS variant to understand the bias of its approximate posterior. Two special conditions were shown to lead to recovery of the true posterior: perfectly correlated parameters and extrapolation parameters. We discussed the validity of applying RMS to NNs, arguing that these two special conditions are partially present in NNs, and that assumptions used for analysis weren't unreasonable. Quantitatively analysing these arguments is an important avenue for future work.

Experiments evidenced success regarding the two goals of this thesis; to create robust and practical techniques for quantifying uncertainty in NNs. On regression benchmarking experiments, state-of-the-art performance was achieved on 3/10 datasets - outperforming alternative ensemble methods and competitive with popular approximate inference methods (table A.3). On image and text classification tasks, anchored ensembles were shown to be more robust to OOD examples than alternative ensemble methods. Its value was further demonstrated on two RL tasks; one in a noisy environment, and one complex robotics problem. Experiments were particularly effective for fully-connected NNs, and there is room for further investigation into why this is the case.

The review in section 4.5 broadly summarised work at the intersection of ensembling and NNs. It's interesting to observe that there has been a departure from theoretically motivated techniques, to those more empirically motivated. Early work focused on developing general ensembling theory, e.g. (Breiman, 1996; Dietterich, 2000), and it was from this angle that ensembling was first applied to NNs (Heskes, 1996; Tibshirani, 1996). More recent work tends to optimise empirical results, e.g. (Lakshminarayanan et al., 2017; Lee et al., 2015), with qualitative explanations offered for why certain elements work or don't.

This reflects trends in deep learning more broadly, where traditional statistical theory doesn't always apply. However, there are several recent examples where theory has successfully been adapted or created to better answer open questions about deep NNs, e.g. (Belkin et al., 2019; Dziugaite and Roy, 2017). From this, one may conclude that the time is right to revisit ensemble theory with today's deep NNs in mind.

Chapter 5

High-Quality Prediction Intervals

TLDR: How to train NNs to directly output prediction intervals.

Key background sections 2.1, 2.2, 2.3.

The previous two chapters explored solutions to challenges arising when applying the Bayesian framework to NNs. Bayesian NNs are just one approach to quantifying uncertainty with NNs, and section 2.3 introduced a variety of alternatives. These sidestep some of the challenges encountered in BNNs.

This chapter presents a new non-Bayesian method, drawn from our paper (Pearce et al., 2018b). Specifically it considers generation of prediction intervals (PIs) from NNs for quantifying uncertainty in regression tasks. PI output is framed as an optimisation objective; it assumes the axiom that high-quality PIs should be as narrow as possible, whilst capturing a specified portion of data. A loss function is derived directly from this axiom that requires no distributional assumption. We show how its form corresponds to a maximum likelihood approach, and that it can be used with gradient descent.

The contributions primarily target quantification of aleatoric uncertainty, but by combining with basic ensembling techniques, epistemic uncertainty is also accounted for. Benchmark experiments show the method outperforms current state-of-the-art uncertainty quantification methods, reducing average PI width by over 10%.

Heuristically motivated methods may provide less guarantee over the *robustness* of the uncertainty estimates, and the chapter closes by investigating a potential failure mode of the method.

5.1 Contributions

Our method considers a NN with two outputs, representing a lower and upper bound of a PI, and formulates PI output as a constrained optimisation problem. We assume the axiom that high-quality PIs should be as narrow as possible, whilst capturing some specified proportion of data points (hereafter referred to as the *HQ principle*). Indeed it is through these metrics that PI quality is often assessed (Galván et al., 2017; Khosravi et al., 2011a; Papadopoulos et al., 2000). We show how a loss function can be derived directly from this HQ principle. In an ensemble it can produce PIs accounting for both model (epistemic) uncertainty and data noise (aleatoric) variance. The key advantages of the method are its intuitive objective, low computational demand, robustness to outliers, and lack of distributional assumption.

Notably we build on the work of Khosravi et al. (2011b) who developed the Lower Upper Bound Estimation (LUBE) method, incorporating the HQ principle directly into the NN loss function for the first time. LUBE is gaining popularity in several communities, for example in the forecasting of energy demand and wind speed (section 5.7). However, we have identified several limitations of LUBE.

- **Gradient Descent** - It was stated that the method was incompatible with gradient descent (GD), a belief carried forward, unchallenged, in all subsequent work (section 5.7). Implementations therefore require non-gradient based methods for training, such as Simulated Annealing (SA) and Particle Swarm Optimisation (PSO). This is inconvenient since GD has become the standard training method for NNs (Goodfellow et al., 2016), used by all modern NN APIs.
- **Loss Form** - Its current form suffers from several problems. The function is at a global minimum when all PIs are reduced to zero. It was also designed through qualitative assessment of the desired behaviour rather than on a statistical basis.
- **Model Uncertainty** - LUBE accounts only for data-noise variance and *not* model uncertainty (section 5.2). This is an oversimplification (Heskes, 1996), implicitly assuming that training data fully populates the input space, which is seldom the case.

Our model addresses each of these issues - henceforth referred to as the quality-driven PI method (QD), and QD-Ens when explicitly referring to the ensembled form.

We link early literature on PIs for NNs (Heskes, 1996; Khosravi et al., 2011b; Papadopoulos et al., 2000; Tibshirani, 1996), with recent work on uncertainty in deep learning (Gal and Ghahramani, 2015; Hernández-Lobato and Adams, 2015; Lakshminarayanan et al.,

2017) - areas which have remained surprisingly distinct. We achieve this by following the same experimental procedure of recent work, assessing performance across ten benchmark regression datasets. We compare QD's performance with the current best performing model, originally named Deep Ensembles (Lakshminarayanan et al., 2017), here referred to as MVE-Ens. We show that QD outperforms in PI quality metrics, achieving closer to the desired coverage proportion, and reducing average PI width by around 10%.

5.2 Prediction Intervals

PIs directly communicate uncertainty, offering a lower and upper bound for a prediction and assurance that, with some high probability (e.g. 95% or 99%), the realised data point will fall between these bounds. Having this information allows for better-informed decisions.

As an example, a point estimate stating that a machine will fail in 60 days may not be sufficient to schedule a repair (does the repair need to be made tomorrow or can it be left for 59 days?), however given a PI of 45-65 days with 99% probability, timing of a repair is more easily scheduled.

Section 2.1.1 introduced uncertainty in the context of regression problems. To briefly recap, we assume a model with additive noise, ϵ, of the form,

$$y = f(\mathbf{x}) + \epsilon. \tag{5.1}$$

Generally NNs trained on regression tasks produce an estimate $\hat{f}(\mathbf{x})$, corresponding to prediction of point estimates of the mean of y (ϵ is assumed to have mean zero). The uncertainty, or variance, of y is denoted σ_y^2 and is estimated according to,

$$\sigma_y^2 = \sigma_{model}^2 + \sigma_{noise}^2, \tag{5.2}$$

with σ_{model}^2 termed *model uncertainty* or *epistemic uncertainty* - uncertainty in $\hat{f}(\mathbf{x})$ - and σ_{noise}^2 *irreducible variance*, *data noise variance*, or *aleatoric uncertainty*.

It is worth here distinguishing confidence intervals (CIs) from PIs. CIs consider the distribution $P(f(\mathbf{x})|\hat{f}(\mathbf{x}))$, and hence only require estimation of σ_{model}^2, whilst PIs consider $P(y|\hat{f}(\mathbf{x}))$ and must also consider σ_{noise}^2. PIs are necessarily wider than CIs (Heskes, 1996).

To construct PIs, σ_y^2 must be estimated at each prediction point. In regions of the input space with more data, σ_{model}^2 decreases, and σ_{noise}^2 may become the larger component. In regions of the input space with little data, σ_{model}^2 grows.

5.3 A Quality-Driven, Distribution-Free Loss Function

5.3.1 Derivation

We now derive a loss function based on the HQ principle. Let the set of input covariates and target observations be $\mathbf{X}$ and $\mathbf{y}$, for n data points, and with $\mathbf{x}_i \in \mathbb{R}^D$ denoting the ith D dimensional input corresponding to y_i, for $1 \leq i \leq n$. The predicted lower and upper PI bounds are $\hat{\mathbf{y}}_{\mathbf{L}}, \hat{\mathbf{y}}_{\mathbf{U}}$. A PI should capture some desired proportion of the observations, $(1-\alpha)$, common choices of α being 0.01 or 0.05,

$$P(\hat{y}_{Li} \leq y_i \leq \hat{y}_{Ui}) \geq (1-\alpha). \tag{5.3}$$

A vector, $\mathbf{k}$, of length n represents whether each data point has been captured by the estimated PIs, with each element $k_i \in \{0,1\}$ given by,

$$k_i = \begin{cases} 1, & \text{if } y_{Li} \leq y_i \leq y_{Ui} \\ 0, & \text{else.} \end{cases} \tag{5.4}$$

We define the total number of data points captured as c,

$$c := \sum_{i=1}^{n} k_i. \tag{5.5}$$

Let Prediction Interval Coverage Probability ($PICP$) and Mean Prediction Interval Width ($MPIW$) be defined as,

$$PICP := \frac{c}{n}, \tag{5.6}$$

$$MPIW := \frac{1}{n}\sum_{i=1}^{n} \hat{y}_{Ui} - \hat{y}_{Li}. \tag{5.7}$$

According to the HQ principle, PIs should minimise $MPIW$ subject to $PICP \geq (1-\alpha)$. To minimise $MPIW$, eq. 5.7 could simply be included in the loss function, however PIs that fail to capture their data point should not be encouraged to shrink further. We therefore introduce *captured* $MPIW$ as the $MPIW$ of *only* those points for which $\hat{\mathbf{y}}_{\mathbf{L}} \leq \mathbf{y} \leq \hat{\mathbf{y}}_{\mathbf{U}}$ holds,

$$MPIW_{\text{capt.}} := \frac{1}{c}\sum_{i=1}^{n} (\hat{y}_{Ui} - \hat{y}_{Li}) \cdot k_i. \tag{5.8}$$

Regarding $PICP$, we take a maximum-likelihood approach, seeking NN parameters, θ, that maximise,

$$\mathcal{L}_\theta := \mathcal{L}(\theta|\mathbf{k}, \alpha). \tag{5.9}$$

Recognising that each element, k_i, is a binary variable taking 1 with probability $(1-\alpha)$, we represent it as a Bernoulli random variable (one per prediction), $k_i \sim \text{Bernoulli}(1-\alpha)$. We further assume that each k_i is iid. This independence assumption may not hold for data points clustered close together, however we believe holds sufficiently for a randomly sampled subset of all data points, as used in mini-batches for GD. This iid assumption allows the total number of captured points, c, to be represented by a binomial distribution, $c \sim \text{Binomial}(n, (1-\alpha))$. Substituting in the pmf,

$$\mathcal{L}_\theta = \binom{n}{c}(1-\alpha)^c \alpha^{n-c}. \tag{5.10}$$

The factorials in the binomial coefficient make computation inconvenient. However using the central limit theorem (specifically the de Moivre-Laplace theorem) it can further be approximated by a normal distribution. For large n,

$$\text{Binomial}(c|n, (1-\alpha)) \approx \mathcal{N}\big(c|n(1-\alpha), n\alpha(1-\alpha)\big) \tag{5.11}$$

$$= \frac{1}{\sqrt{2\pi n\alpha(1-\alpha)}} \exp\left[-\frac{(c-n(1-\alpha))^2}{2n\alpha(1-\alpha)}\right]. \tag{5.12}$$

We consider this a mild assumption provided a mini-batch of reasonable size, say > 50, is used.

It is common to minimise the NLL rather than maximise the likelihood, this simplifies eq. 5.12 to,

$$-\log \mathcal{L}_\theta \propto \frac{(n(1-\alpha)-c)^2}{n\alpha(1-\alpha)} \tag{5.13}$$

$$= \frac{n}{\alpha(1-\alpha)}((1-\alpha) - PICP)^2. \tag{5.14}$$

Eq. 5.3 demands, $PICP < (1-\alpha)$. Since this is an inequality rather than an equality, we convert eq. 5.14 to a one-sided loss when combining with eq. 5.8 by applying the max operator. A penalty term, λ, is also added controlling the importance of width vs. coverage. This gives a loss,

$$\text{Loss}_{QD} = MPIW_{\text{capt.}} + \lambda \frac{n}{\alpha(1-\alpha)} \max(0, (1-\alpha) - PICP)^2. \tag{5.15}$$

5.3.2 Comparison to LUBE

The derived loss function in eq. 5.15 may be compared to the LUBE loss (Khosravi et al., 2011b),

$$\text{Loss}_{LUBE} = \frac{MPIW}{r}\Big(1+\exp\big(\lambda\ \max(0,(1-\alpha)-PICP)\big)\Big), \tag{5.16}$$

where $r = \max(\mathbf{y}) - \min(\mathbf{y})$, is the range of the target variable.

Whilst still recognisable as having the same objective, the differences are significant, and are summarised as follows.

- The inclusion of n intuitively makes sense since a larger sample size provides more confidence in the value of $PICP$, and hence a larger loss should be incurred. Similar arguments follow for α. They remove the need to adjust λ based on batch size and target coverage.
- A squared term has replaced the exponential. Whilst the RHS for both is minimised when $PICP \geq (1-\alpha)$, the squared term was derived based on likelihood whilst the exponential term was selected qualitatively.
- $MPIW$ now has an additive rather than multiplicative effect. Multiplying has the attractive property of ensuring both terms are of the same magnitude. However it also means that a global minimum is found when all PIs are of zero width. We found in practice that NNs occasionally did produce this undesirable solution.
- $MPIW$ is no longer normalised by the range of $\mathbf{y}$. Data for a NN should already be normalised during preprocessing. Further normalisation is therefore redundant. It also merely scales the loss by a constant, having no overall effect on the optimisation.
- $MPIW_{\text{capt.}}$ is used rather than $MPIW$. As discussed in section 5.3.1 this avoids the NN benefiting by further reduction of PI widths for missed data.

5.3.3 Training QD with Gradient Descent

It was originally believed that the LUBE loss function was, *"nonlinear, complex, and non-differentiable... gradient descent-based algorithms cannot be applied for its minimization"* (Khosravi et al., 2011b). This belief has been carried forward, unchallenged, in all subsequent

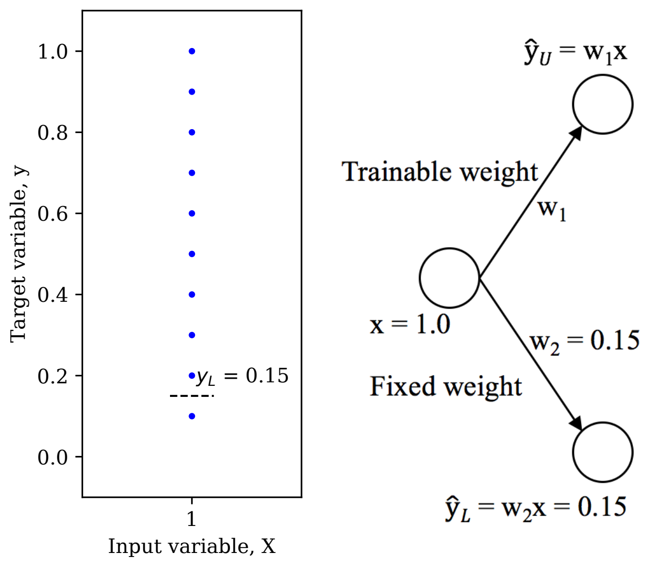


Figure 5.1 Set up for the toy problem: Left is input data (10 data points linearly spaced at fixed input $x = 1.0$), right is the NN used (one trainable weight).

work - see section 5.7 for numerous examples. It is inconvenient since GD is the standard method for training NNs, so implementations require extra coding effort.

Regarding the quoted justification, most standard loss functions are nonlinear - e.g. L_2 errors - and whilst the LUBE loss function is complex, this does not affect its compatibility with GD[1]. The non-differentiability comment is partially valid. Because the loss function requires the use of step functions, it is not differentiable everywhere. This also occurs in ReLUs which are not differentiable when the input is exactly zero[2]. However it does raise an additional challenge in its optimisation which will be discussed shortly.

GD Toy Example

Loss_{QD} can be directly implemented as shown in Algorithm 2 (Loss_H), however it fails to converge to a minimum. We demonstrate why this is the case and how it can be remedied through a toy example.

[1]Modern NN APIs generally handle gradient computation automatically, through application of the chain rule to the predefined operations. Provided functions within the API library are used, gradient calculations are automatically handled.

[2]Software implementations return one of the derivatives either side of zero when the input corresponds to the undefined point rather than raising an error Goodfellow et al. (2016).

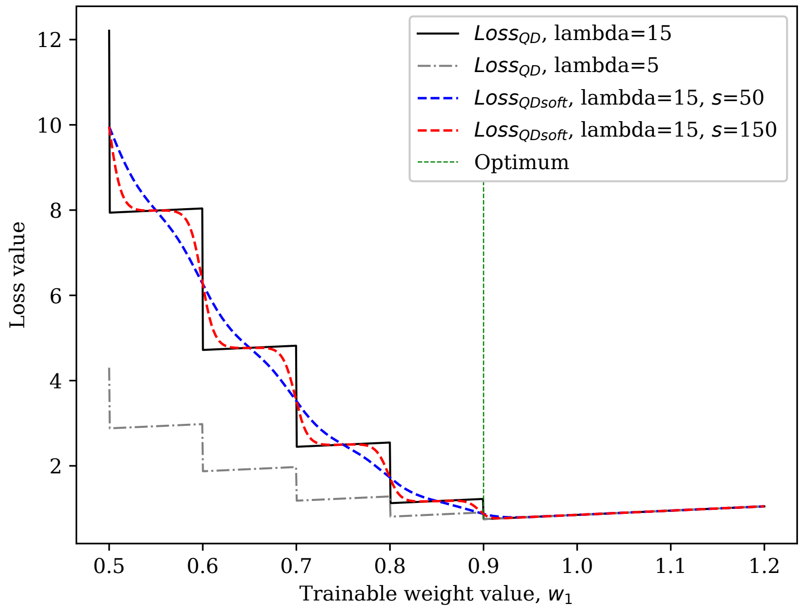


Figure 5.2 Error surface of Loss_{QD}, and $\text{Loss}_{QD\text{soft}}$ on toy problem, with effect of hyperparameters λ and s.

Consider a NN as in figure 5.1 with one input and two output neurons, linear activations and no bias. For purposes of clarity, one weight is fixed, $w_2 = 0.15$, to create a one dimensional problem with a single trainable weight, w_1. Given 10 data points evenly spaced at $x = 1.0$, and $\alpha = 0.2$, the optimal value for $\hat{y}_U$ (and therefore w_1) is 0.9, which gives the lowest $MPIW$, subject to $PICP \geq 1-\alpha = 0.8$.

Loss_{QD} is plotted in figure 5.2 (black line). Whilst the global minimum occurs at the desired point, this solution is not found through GD. Given the steepest descent weight update rule with some learning rate τ,

$$w_{1,t+1} = w_{1,t} - \tau \frac{\partial \text{Loss}_{QD}}{\partial w_{1,t}}, \tag{5.17}$$

the weight shrinks without converging. This is because the gradient, $\frac{\partial \text{Loss}}{\partial w_1}$, at any point is positive, except for the discontinuities which are never realised.

To remediate this, we introduce an approximation of the step function. The sigmoid function has been used in the past as a differentiable alternative (Yan et al., 2004). In eq. 5.4, **k**, the captured vector was defined. We redefine this as $\mathbf{k}_{\text{hard}}$ and introduce a relaxed version as

Algorithm 2 Construction of loss function using basic operations

```
**Input:** Target values, $\mathbf{y}$, predictions of lower and upper bound, $\hat{\mathbf{y}}_{\mathbf{L}}, \hat{\mathbf{y}}_{\mathbf{U}}$, desired coverage,
$(1-\alpha)$, and sigmoid softening factor, $s$, $\odot$ denotes the element-wise product.

# hard uses sign step fn, sign returns -1 if -ve, +1 if +ve
$\mathbf{k_{HU}} = \max(0, \text{sign}(\hat{\mathbf{y}}_{\mathbf{U}} - \mathbf{y}))$
$\mathbf{k_{HL}} = \max(0, \text{sign}(\mathbf{y} - \hat{\mathbf{y}}_{\mathbf{L}}))$
$\mathbf{k_H} = \mathbf{k_{HU}} \odot \mathbf{k_{HL}}$

# soft uses sigmoid fn
$\mathbf{k_{SU}} = \text{sigmoid}((\hat{\mathbf{y}}_{\mathbf{U}} - \mathbf{y}) \cdot s)$
$\mathbf{k_{SL}} = \text{sigmoid}((\mathbf{y} - \hat{\mathbf{y}}_{\mathbf{L}}) \cdot s)$
$\mathbf{k_S} = \mathbf{k_{SU}} \odot \mathbf{k_{SL}}$

$MPIW_c =$
  $\text{reduce\_sum}((\hat{\mathbf{y}}_{\mathbf{U}} - \hat{\mathbf{y}}_{\mathbf{L}}) \odot \mathbf{k_H}) / \text{reduce\_sum}(\mathbf{k_H})$
$PICP_H = \text{reduce\_mean}(\mathbf{k_H})$
$PICP_S = \text{reduce\_mean}(\mathbf{k_S})$
$\text{Loss}_H = MPIW_c +$
  $\lambda \cdot \frac{n}{\alpha(1-\alpha)} \cdot \max(0, (1-\alpha) - PICP_H)^2$
$\text{Loss}_S = MPIW_c +$
  $\lambda \cdot \frac{n}{\alpha(1-\alpha)} \cdot \max(0, (1-\alpha) - PICP_S)^2$
```

follows,

$$\mathbf{k}_{\text{soft}} = \sigma(s(\mathbf{y} - \hat{\mathbf{y}}_{\mathbf{L}})) \odot \sigma(s(\hat{\mathbf{y}}_{\mathbf{U}} - \mathbf{y})), \tag{5.18}$$

where σ is the sigmoid function, and $s > 0$ is some softening factor. We further define $PICP_{\text{soft}}$ and $\text{Loss}_{QD\text{soft}}$ by replacing $\mathbf{k}_{\text{hard}}$ with $\mathbf{k}_{\text{soft}}$ in equations (5.6) & (5.15) respectively - see also Loss_S in Algorithm 2.

Figure 5.2 shows the result of using $\text{Loss}_{QD\text{soft}}$ (red & blue lines). By choosing an appropriate value for s, following the steepest gradient *does* lead to a minimum, making GD a viable method. Setting $s = 160$ worked well in experiments in section 5.5.2, requiring no alteration across datasets.

5.3.4 Particle Swarm Optimisation

The original LUBE loss function, eq. 5.16, has been implemented with various evolutionary training schemes that do not require derivatives of the loss function. In order to test the efficacy of $\text{Loss}_{QD\text{soft}}$ with GD, we compared to an evolutionary-based training method (section 5.5.1). PSO (Kennedy and Eberhart, 1995) was chosen due to use in recent work with LUBE (Galván et al., 2017; Wang et al., 2017). We make the assumption that other evolutionary methods would offer similar performance (Jones, 2005). See Kennedy & Eberhart (2001) for an introduction to PSO.

5.4 Ensembles to Estimate Model Uncertainty

In section 5.2, two components of uncertainty were defined; model uncertainty (epistemic) and data noise variance (aleatoric). It appears that previous work assumed both were accounted for (section 5.7). In fact, LUBE & QD only estimate data noise variance, and there is a need to consider the uncertainty of these estimates themselves. This becomes particularly important when new data is encountered. Consider a NN trained for the example in figure 5.1: Despite being capable of estimating the data noise variance at $x = 1.0$, if shown new data at $x = 2.0$ it would predict $\hat{y}_U = 1.8$, with little basis.

Ensembling models provides a conceptually simple way to deal with this. Chapter 4 discusses ensembling in some detail, as well as providing a modification to encourage more robust, Bayesian estimates. Here only the basic form of ensembling is used; that of training several NNs beginning from different initialisations. The resulting ensemble of NNs contains some diversity, and the variance of their predictions can be used as an estimate of model uncertainty.

Given an ensemble of m NNs trained with Loss$_{QD\text{soft}}$, let $\tilde{\mathbf{y}}_{\mathbf{U}}, \tilde{\mathbf{y}}_{\mathbf{L}}$ represent the *ensemble's* upper and lower estimate of the PI. We calculate model uncertainty and hence the ensemble's PIs as follows,

$$\bar{y}_{Ui} = \frac{1}{m}\sum_{j=1}^{m} \hat{y}_{Uij}, \tag{5.19}$$

$$\hat{\sigma}^2_{model} = \sigma^2_{Ui} = \frac{1}{m-1}\sum_{j=1}^{m} (\hat{y}_{Uij} - \bar{y}_{Ui})^2, \tag{5.20}$$

$$\tilde{y}_{Ui} = \bar{y}_{Ui} + 1.96\sigma_{Ui}, \tag{5.21}$$

where $\hat{y}_{Uij}$ represents the upper bound of the PI for data point i, for NN j. A similar procedure is followed for $\tilde{y}_{Li}$, subtracting rather than adding $1.96\sigma_{Li}$. This implicitly assumes the ensemble's PI distribution is normally distributed – note that this is different from assuming the data noise is normally distributed.

5.5 Experiments

5.5.1 Qualitative Experiments

In this section behaviour of QD is qualitatively assessed on synthetic data. Firstly, the GD method of training explained in section 5.3.3 is compared to PSO. Next, the advantage of QD over MVE (section 5.7) in data with non-normal variance is shown. Finally, the effectiveness of the ensembled QD approach at estimating model uncertainty is demonstrated.

Appendix B.1 contains experimental details as well as numerical results for this synthetic data, covering a full permutation of loss functions and training methods [LUBE, QD, MVE]x[GD, PSO].

Training method: PSO vs. GD

Comparison of evolutionary methods vs. GD for NN training is its own research topic, and as such analysis here is limited. Preliminary experiments showed that GD performed slightly better than PSO in terms of $PICP$ and $MPIW$, producing smoother, tighter boundaries, both more consistently and with lower computational effort. See figure 5.3 for a graphical comparison of typical PI boundaries. Data was generated by $y = 0.3\sin(x) + 0.2\epsilon$, with $\epsilon \sim N(0, x^4)$.

Loss function: QD vs. MVE

Here, the advantage of a distribution-free loss function is demonstrated by comparing MVE, which assumes Gaussian data noise, to QD, which makes no such assumption, on two

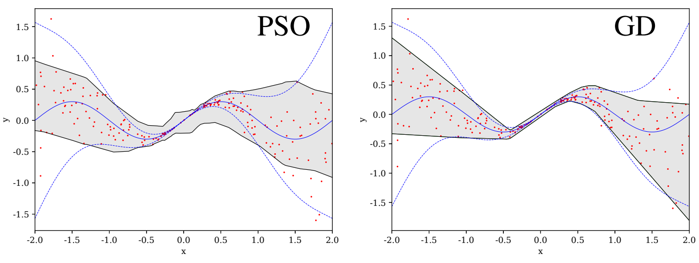


Figure 5.3 Comparison of PI boundaries for GD vs. PSO training methods. Shading is the predicted 95% PI. Ground truth is given by blue lines - the ideal 95% boundaries and mode.

synthetic datasets. The first was generated as in 5.5.1 with normal noise, the second with exponential noise, $\epsilon \sim \exp(1/x^2)$.

Figure 5.4 shows, unsurprisingly, that MVE outputs PIs very close to the ideal for normal noise, but struggles with exponential noise. QD approximates both reasonably, though does not learn the boundaries well where data is sparse. Though possible to alter MVE to assume an exponential distribution, this would require choosing this distribution a priori, and redefining the loss function accordingly. With real data, the distribution would be unknown, and likely irregular, putting QD at an advantage.

Model Uncertainty Estimation: Ensembles

This experiment demonstrated the ability of ensembling to estimate model uncertainty. Data was generated through $y = 0.02x^3 + 0.02\epsilon$, with $\epsilon \sim N(0, 3^2)$. Figure 5.5 shows ten individual QD PIs as well as the ensembled PIs. The estimated model uncertainty, $\hat{\sigma}^2_{model}$, calculated from eq. 5.20 is overlaid. Whilst it is difficult to reason about the correctness of the absolute value, its behaviour agrees with the intuition that uncertainty should increase in regions of the input space that are not represented in the training set, here $x \in [-1, 1]$, and $x > 4, x < -4$.

5.5.2 UCI Regression Benchmarks

To compare QD to recent work on uncertainty in deep learning, we adopted their shared experimental procedure (Gal and Ghahramani, 2015; Hernández-Lobato and Adams, 2015; Lakshminarayanan et al., 2017). This follows a similar protocol to experiments run in earlier chapters - table 4.1, over ten open-access datasets. Models were asked to output 95% PIs and used five NNs per ensemble. See appendix B.1 for full experimental details. Code is made

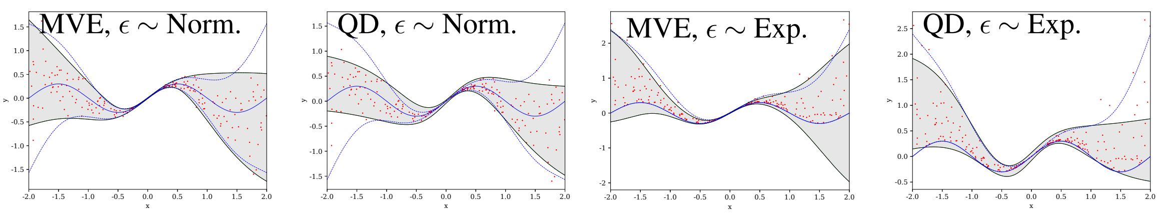


Figure 5.4 Comparison of PI boundaries for two loss functions, QD vs. MVE, given data noise variance drawn from different distributions. Legend as for figure 5.3.

available online[3]. Note that whilst single-layer NNs were used to be consistent with previous works, our method may be applied without modification to deeper architectures.

Previous work reported NLL & RMSE metrics (as did earlier thesis experiments). However, the important metrics for PI quality are $MPIW$ and $PICP$ (section 5.1). We chose to compare QD-Ens to MVE-Ens, since it had reported the best NLL results to date (Lakshminarayanan et al., 2017). We did not include LUBE since ensembling and GD had already been justified in section 5.5.1.

QD-Ens and MVE-Ens both output fundamentally different things; MVE-Ens a distribution, and QD-Ens upper and lower estimates of the PI. To compute NLL & RMSE for QD-Ens is possible only by imposing a distribution on the PI. This is not particularly fair since the attraction of the method is its lack of distributional assumption. Purely for comparison purposes we did this in appendix B.1.

A fairer comparison is to convert the MVE-Ens output distributions to PIs, and compute PI quality metrics. This was done by trimming the tails of the MVE-Ens output normal distributions by the appropriate amount, which allowed extraction of $MPIW$, $PICP$, and

[3]https://github.com/TeaPearce

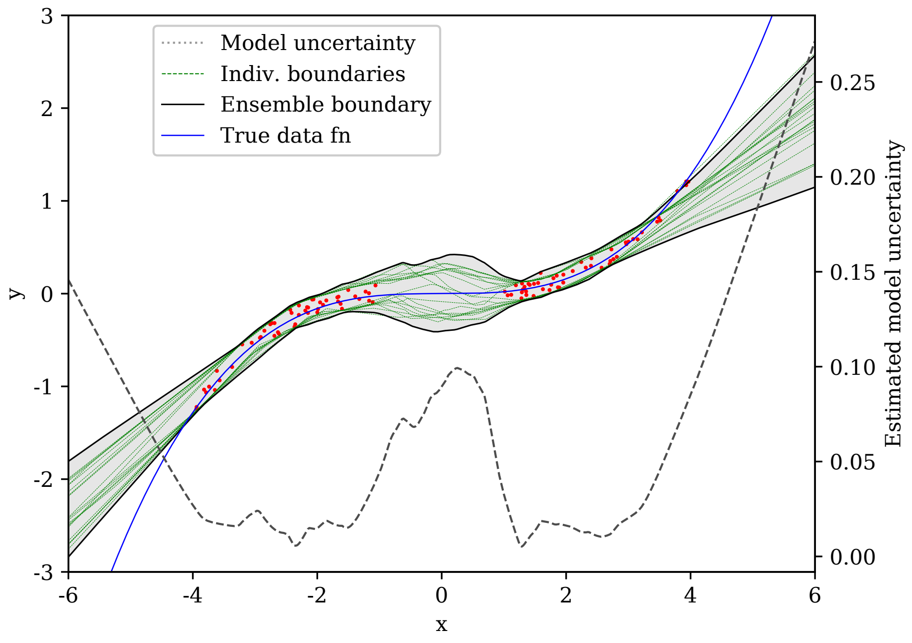


Figure 5.5 Estimation of model uncertainty with a QD ensemble.

Table 5.1 PI quality metrics on ten benchmarking regression datasets; mean $\pm$ one standard error, best result in bold. Best was assessed according to the following criteria. If $PICP \geq 0.95$ for both, both were best for $PICP$, and best $MPIW$ was given to smallest $MPIW$. If $PICP \geq 0.95$ for neither or for only one, largest $PICP$ was best, and $MPIW$ only assessed if the one with larger $PICP$ also had smallest $MPIW$.

	Loss$_{QD\text{soft}}$		PICP		MPIW		
	MVE-Ens	QD-Ens	MVE-Ens	QD-Ens	MVE-Ens	QD-Ens	improvement
Boston	1.76 ± 0.28	$\mathbf{1.33 \pm 0.05}$	0.89 ± 0.02	$\mathbf{0.92 \pm 0.01}$	0.87 ± 0.03	1.16 ± 0.02	NA
Concrete	1.23 ± 0.06	$\mathbf{1.16 \pm 0.02}$	0.92 ± 0.01	$\mathbf{0.94 \pm 0.01}$	1.00 ± 0.02	1.09 ± 0.01	NA
Energy	0.50 ± 0.02	$\mathbf{0.47 \pm 0.01}$	$\mathbf{0.99 \pm 0.00}$	$\mathbf{0.97 \pm 0.01}$	0.50 ± 0.02	$\mathbf{0.47 \pm 0.01}$	7%
Kin8nm	$\mathbf{1.14 \pm 0.01}$	1.24 ± 0.01	$\mathbf{0.97 \pm 0.00}$	$\mathbf{0.96 \pm 0.00}$	$\mathbf{1.14 \pm 0.01}$	1.25 ± 0.01	-10%
Naval	0.31 ± 0.01	$\mathbf{0.27 \pm 0.01}$	$\mathbf{0.99 \pm 0.00}$	$\mathbf{0.98 \pm 0.00}$	0.31 ± 0.01	$\mathbf{0.28 \pm 0.01}$	10%
Power plant	0.91 ± 0.00	$\mathbf{0.86 \pm 0.00}$	$\mathbf{0.96 \pm 0.00}$	$\mathbf{0.95 \pm 0.00}$	0.91 ± 0.00	$\mathbf{0.86 \pm 0.00}$	6%
Protein	2.70 ± 0.01	$\mathbf{2.28 \pm 0.01}$	$\mathbf{0.96 \pm 0.00}$	$\mathbf{0.95 \pm 0.00}$	2.69 ± 0.01	$\mathbf{2.27 \pm 0.01}$	15%
Wine	4.13 ± 0.31	$\mathbf{3.13 \pm 0.19}$	0.90 ± 0.01	$\mathbf{0.92 \pm 0.01}$	2.50 ± 0.02	$\mathbf{2.33 \pm 0.02}$	7%
Yacht	0.31 ± 0.02	$\mathbf{0.23 \pm 0.02}$	$\mathbf{0.98 \pm 0.01}$	$\mathbf{0.96 \pm 0.01}$	0.30 ± 0.02	$\mathbf{0.17 \pm 0.00}$	43%
Song year	$2.90 \pm$ ± NA	**2.47** $\pm$ **± NA**	**0.96** $\pm$ **±NA**	**0.96** $\pm$ **± NA**	$2.91 \pm$ ± NA	**2.48** $\pm$ **± NA**	15%

Loss$_{QD\text{soft}}$. In our experiments we ensured MVE-Ens achieved NLL & RMSE scores at least as good as those reported in the original work, before PI metrics were calculated.

Experimental Discussion

Full results of PI quality metrics are given in table 5.1, NLL & RMSE results are included in appendix B.1. Given that Loss$_{QD\text{soft}}$ is representative of PI quality, QD-Ens outperformed MVE-Ens on all but one dataset. $PICP$ was generally closer to the 95% target, and $MPIW$ was on average 11.6% narrower. The exception to this was the *Kin8nm* dataset. In fact, this dataset was synthetic (Danafar et al., 2010), and we suspect that Gaussian noise may have been used in its simulation, which would explain the superior performance of MVE-Ens.

One drawback of QD-Ens was the fragility of the training process. Compared to MVE-Ens it required a lower learning rate, was more sensitive to decay rate, and hence needed from two to ten times more training epochs.

Other comments are as follows. We found λ a convenient lever providing some control over $PICP$. We tried to establish a relationship between the normality of residual errors and improvement of QD-Ens over MVE-Ens, but due to the variable power of normality tests analysis was unreliable.

5.6 Limitations

This section draws attention to some potential failure modes of the method. The HQ principle provides our method with an intuitive objective that can be directly optimised with minimal assumptions made about the data distribution. However, the principle is heuristic in nature, which can bring disadvantages.

The HQ principle implicitly assumes that the coverage probability is global, $P(\hat{y}_{Li} \leq y_i \leq \hat{y}_{Ui}) \geq (1-\alpha)$ (eq. 5.3), rather than conditional on the input, $P(\hat{y}_{Li} \leq y_i \leq \hat{y}_{Ui}|\mathbf{x}) \geq (1-\alpha)$. This distinction can have impact in several situations, detailed below.

Though these examples are somewhat contrived, it is important for practitioners to be aware of such pitfalls. On the other hand, this behaviour is of value in certain situations, e.g. for datasets containing outliers, since predictions will be less influenced by data points falling far outside the norm.

5.6.1 Varying Data Noise

Figure 5.6 shows an extreme example of when highly varying data noise might be a problem. Here a single QD NN is trained to output a 95% PI. Since the high variance data noise comprises less than 5% of the total data population, the NN learns to simply ignore these data points in order to minimise $MPIW$, whilst $PICP$ remains above the required 95%.

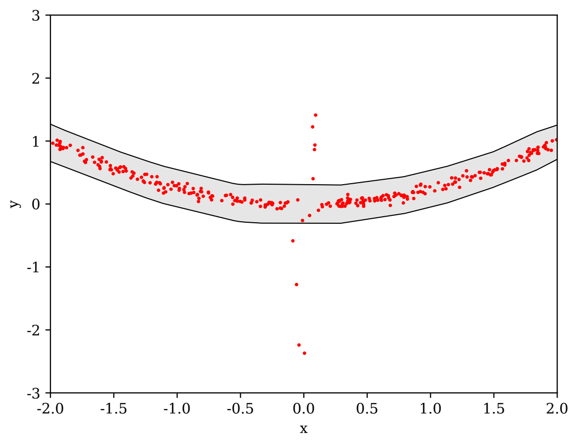


Figure 5.6 QD assures global but not pointwise coverage probability.

5.6.2 Uniform Data Noise

Figure 5.7 shows a function with uniform data noise. Due to its uniformity, a PI capturing the top $(1-\alpha)\%$ of data points will have the same $MPIW$ and $PICP$ as one capturing the bottom $(1-\alpha)\%$. Firstly, this might be less useful than capturing the middle $(1-\alpha)\%$ of data points. Secondly, this is problematic when one uses the diversity of an ensemble to estimate model uncertainty.

For data noise distributions with longer tails, this behaviour should be more benign.

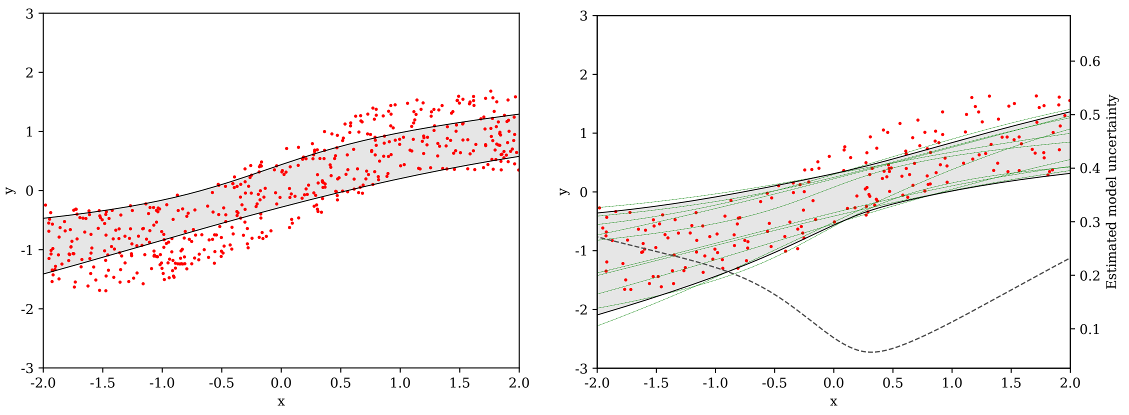


Figure 5.7 When fitting a single QD NN to uniform data noise, many PIs will give the same MPIW and PICP. This can lead to diversity in the ensemble, even in regions with much data, and model uncertainty estimates can be overestimated.

5.7 Related Work

Section 2.3 provided an overview of methods to quantify uncertainty in NNs. This section reviews research on outputting PIs directly from NNs.

Many methods *implicitly* output PIs from NNs - any distribution output from a NN can be transformed to a PI, as we have done in this chapter with the MVE method (Nix and Weigend, 1994). There are two prominent methods that more *explicitly* output PIs; LUBE and quantile regression.

LUBE (Khosravi et al., 2011b) was developed on the HQ principle. Originally it was proposed with SA as the training method, and much effort has gone toward trialling it with various non-gradient based training methods including Genetic Algorithms (Ak et al., 2013b), Gravitational Search Algorithms (Lian et al., 2016), PSO (Galván et al., 2017; Wang et al., 2017), Extreme Learning Machines (Sun et al., 2017), and Artificial Bee Colony Algorithms (Shen et al., 2018). Multi-objective optimisation has been found useful in considering the tradeoff between PI width and coverage (Galván et al., 2017; Shen et al., 2018).

LUBE has been used in a plethora of application-focused work: Particularly in energy load (Pinson and Kariniotakis, 2013; Quan et al., 2014) and wind speed forecasting (Ak et al., 2013b; Wang et al., 2017), but also prediction of landslide displacement (Lian et al., 2016), gas flow (Sun et al., 2017), solar energy (Galván et al., 2017), condition-based maintenance (Ak et al., 2013a), and others. All work used LUBE as a single NN, making no attempt to account for model uncertainty (section 5.2).

Section 2.3 introduced quantile regression NNs (Taylor, 2000). These can be trained to output a desired quantile of the output distribution (e.g. 50 percentile or 5 percentile). A NN can be built outputting two quantiles simultaneously (say 2.5% and 97.5%), and these outputs can be interpreted as the upper and lower bounds of a PI (here 95% coverage). The attraction over LUBE and QD is that quantiles are conditional on the input (section 5.6). Follow up works to our QD method have shown some advantage of this quantile regression approach over directly optimising PI metrics (Kivaranovic et al., 2020; Tagasovska and Lopez-Paz, 2019).

5.8 Discussion & Conclusion

In this chapter we developed a non-Bayesian technique to quantify uncertainty with NNs for regression tasks. We derived a loss function for the output of PIs based on the assumption that high-quality PIs should be as narrow as possible subject to a given coverage proportion. We contrasted it with a previous work, justified differences and showed that it can be used successfully with GD with only slight modification. We see this as a key contribution, given the popularity of LUBE and the ubiquity of GD.

We described why a single NN using the derived loss function underestimates uncertainty, and that this can be addressed by using the model in an ensemble. On ten benchmark regression datasets, the new model reduced PI widths by over 10%.

A potential failure mode of the method was highlighted, arising from desired coverage being global rather than conditional on the input. Speaking more generally, this provides an illustrative example of how using heuristically driven methods can lead to unexpected risks.

It’s possible that other heuristic methods introduced in section 2.3 also suffer from such problems. This provides justification for adopting the more rigorous Bayesian approach, as has been explored in chapters 3 & 4 of this thesis.

Chapter 6

Discussion & Conclusion

TLDR: Contributions of this thesis in the context of the field. Contrast current approaches to uncertainty-aware AI with theories in neuroscience.

This chapter begins by summarising the technical contributions presented in this thesis, placing them in the context of the field today. The chapter then takes on a more speculative tone, contrasting today's approaches to building AI systems capable of operating under uncertainty, with mechanisms that may exist in biological intelligences, which have proved evolutionarily successful in uncertain environments. This draws from the extended abstract (Pearce et al., 2020a). We close by listing fertile directions for future research.

6.1 Summary of Contributions

The goal of this thesis was to develop mechanisms enabling deep NNs to output robust uncertainty estimates, whilst minimising the impact on usability. This thesis contributed to several lines of research striving for this.

Chapters 3 & 4 contributed to the line of research applying the Bayesian framework to parameters of a NN, today termed 'Bayesian deep learning'. Chapter 3 discussed how to design suitable priors for a BNN, which is an important step in any Bayesian model, yet it is not obvious how this can be done for BNNs. We showed that the connection between BNNs and GPs offers a convenient lense to study the prior over functions of a BNN. The chapter's major contribution took a technique widely used in building expressive GPs (kernel combinations), and showed how it could be translated to build expressive BNNs.

Chapter 4 studied a second challenge arising in BNNs; how to efficiently learn the posterior distribution. Our proposal modified the regularisation terms of an ensemble of NNs, such that their resulting estimates are closer to the Bayesian posterior. This is useful since ensembles of NNs are easy to implement and scalable (limited impact on usability), and connecting them with the Bayesian framework provides some assurance that their uncertainty estimates are robust.

Finally, chapter 5 contributed to an alternative (non-Bayesian) line of research seeking to train NNs to directly output PIs. We proposed modifications to the key algorithm in this area; firstly proposing a more principled objective function, secondly showing how it can be used with standard gradient descent learning, and thirdly arguing that it should be used with a further technique, such as ensembling, to account for epistemic uncertainty.

6.1.1 Follow up Work

This section briefly reviews recent papers (*not* by the thesis author) that have applied and extended ideas presented in this thesis. These emerged after publication of the main proposals, and before thesis submission.

Bayesian Ensembling (Pearce et al., 2019a) - Chapter 4

Clements et al. (2020) use Bayesian ensembling in conjunction with quantile regression in an RL setting. Interestingly, they show that only two NNs are required in the ensemble to produce good uncertainty estimates, if the number of quantiles being estimated is large. He et al. (2020) explored the anchored loss from the perspective of the neural tangent kernel, reaching similar conclusions to our own about recovery of the Bayesian posterior. Several works have applied our method to real-world problems. Sengupta et al. (2020) successfully used it to predict when instabilities in combustion might occur, which could be used in prognostics of certain machinery. Yong et al. (2020) combined our technique with autoencoder architectures, motivated by recognising drift of sensors in industrial settings. The method was also applied to filling in missing entries in historical environmental recordings (Sengupta et al., 2020). This last case extended anchored ensembles to incorporate heteroskedastic aleatoric uncertainty, and also used insights from **Expressive Priors** (chapter 3) (Pearce et al., 2019b) to set appropriate prior distributions.

High-Quality Prediction Intervals (Pearce et al., 2018b) - Chapter 5

The quality-driven loss and algorithm is beginning to establish itself as a baseline in works involving PIs and NNs, e.g. (Kivaranovic et al., 2020; Tagasovska and Lopez-Paz, 2019; Wang et al., 2020; Zhu et al., 2019). Several works have offered improvements over our method, which may derive from the shortcomings discussed in section 5.6. Kivaranovic et al.

(2020) firstly propose adding a third output to the NN representing the mean prediction. More significantly, they suggest quantile regression in conjunction with a conformal prediction framework can provide more rigourous coverage guarantees. Tagasovska & Lopez-Paz (2018) combine quantile regression with 'training certificates'. Their empirical results appear to offer improvement. Salem et al. (2020) also proposed combining PI output with point estimate prediction, and added another term to the loss. This appeared to help stabilise training.

6.2 Visualising the Field Today

Building in mechanisms that allow NNs to estimate their uncertainty remains an active research area. Comparison between methods is often done empirically through benchmark tasks, as in many other areas of machine learning – through standardised benchmarks such as UCI datasets (Hernández-Lobato and Adams, 2015) and recently released tasks on real-world problems (Filos et al., 2019), as well as through tailored experiments such as ours in section 4.4.

These empirical tests are good at assessing the robustness of uncertainty estimates, yet they say little about impact on usability. This is important since it has a direct effect on adoption rate by practitioners in the wider machine learning community.

The popularity of MC Dropout (Gal and Ghahramani, 2015) is an excellent example of this. NNs can easily be designed to contain dropout layers, allowing practitioners a convenient method for obtaining uncertainty estimates. Despite the *quality* of the estimates produced by MC Dropout often being lower than alternatives (e.g. figure 4.4, (Foong et al., 2019; Gustafsson et al., 2019)), this convenience has led to it being widely adopted.

Practitioners requiring more precise uncertainty estimates will likely be willing to go to extra lengths in terms of implementation and computation, in which case ensembling or VI might be a good choice. And for those requiring uncertainty quality above all else, running HMC or switching to a GP might be suitable.

Observing this, it becomes apparent that uncertainty quality and usability can be thought of as a *continuum* rather than discrete attributes. All scalable Bayesian inference methods applied to NNs today are approximate, and it becomes a matter of trading off the accuracy of the approximation with the impact on usability.

Figure 6.1 visualises this spectrum for the broad classes of techniques encountered in this thesis (left), as well as the impact of our specific proposals (right). Certainly there is a great

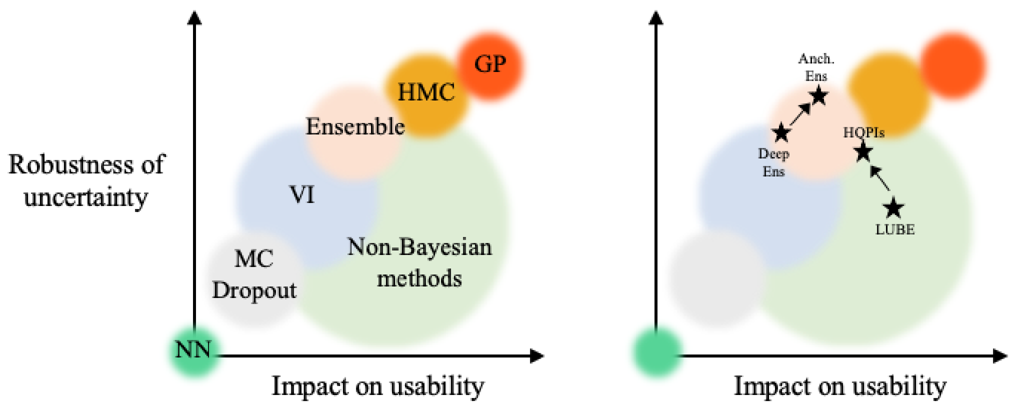


Figure 6.1 Qualitative, approximate view of today's approaches to uncertainty estimation in NNs plotted according to the two criteria targeted in this thesis. Left: The broad classes of techniques. Right: Impact of work in this thesis, comparing the original methods (Deep Ens, LUBE), to the proposals of this thesis (Anch Ens, HQPIs). Circle size roughly represents variability within methods and error of the estimation.

deal of noise and judgement behind this, and the size of the circles attempts to represent both the variability within method (e.g. VI can be implemented with factorised or full covariance matrix, and HMC can be computed over the full dataset or over batches), as well as our confidence in their position. The largest circle represents various non-Bayesian methods, some of which were introduced in chapter 2.

As deep learning develops and matures, it's likely that methods will move around on these axes. For example, whilst dropout used to be common in state-of-the-art models (circa 2015), its use is on decline – both in modern visual models where it has been found to conflict with batch normalisation (Lee et al., 2019) and in RL where the presence of extra variance is undesirable. This may open an opportunity for new approximate methods with low impact on usability.

To summarise, we argue that rather than optimising purely for uncertainty robustness on benchmarks, we should be searching for methods that sit on the pareto front of robustness and usability. Ultimately this will lead to set of useful tools for the wider community.

6.3 Looking to the Future

Many of today's approaches to handling uncertainty in NNs are derived from techniques that work well in simpler predictive models. This thesis has largely dealt with the challenge of how the Bayesian framework, which has a rich history, can be applied to the parameters of a NN. Whilst the Bayesian framework has the attraction of being theoretically well-grounded, scaling it to modern NNs and datasets is a formidable challenge.

The longer term goal of AI is to build ever more general systems. It is likely this will require models and datasets of ever increasing size, suggesting the difficulty of applying the Bayesian framework will only increase. This section speculatively considers a question: **Is continuing to scale up these simpler frameworks for handling uncertainty to ever larger and more complex models feasible, or will an entirely different paradigm be needed?**

One way to approach this question is to consider the only example of (relatively) general intelligence we have; brains. They operate in a world filled with uncertainties – deriving from incomplete experience and imperfect perceptions. A normative argument suggests that in order to have evolved successfully, brains must have learnt to handle these uncertainties.

This leads to an interesting perspective – perhaps understanding how brains evaluate uncertainty can guide us to building better uncertainty-aware AI systems.

A further motivation for exploring this perspective is that the fundamental unit of NNs directly grew out of a neuroscientific understanding of the brain (McCulloch and Pitts, 1943; Rosenblatt, 1958) (chapter 2). Despite this commonality, the fields of neuroscience and NNs have never been more separate (Hassabis et al., 2017). Uncertainty is no exception; this is well demonstrated by examining how each field cites the other. We were unable to find any such references in recent Bayesian deep learning papers (here defined as papers exploring NNs with parameter uncertainty, and excluding work in the wider field, e.g. on VAEs and generative models), whilst neuroscience papers on the topic are only marginally better, occasionally citing classic machine learning papers, such as (MacKay, 1992a), and never modern work. We found only a single instance citing heavily across both fields (Ghahramani, 2015).

6.3.1 Uncertainty in the Brain

This section briefly reviews some select psychology, cognitive science and neuroscience literature on brains and uncertainty – note that this area goes far beyond the set up considered by BNNs, for example considering cue combination and uncertainty over latent variables. It remains an active research area and largely unresolved.

Several experimental studies have demonstrated that animals as well as humans can be cognisant of their own uncertainty (Call, 2010; Smith et al., 1997, 1995). In certain cases it's been demonstrated that uncertainties are combined according to Bayes' rule (Kording and Wolpert, 2004).

A typical experimental setup consists of an animal repeatedly receiving some variable stimulus (e.g. a tone of varying pitch), with a choice of three actions. If the stimulus is above a certain threshold (e.g. 1000Hz) the animal is rewarded for making the action associated with a 'high' response (e.g. move left), if low it is rewarded for making a 'low' response (e.g. move right). A third option allows the animal to abstain from the decision, which offers a small but certain reward. Animals often learn behaviour that aligns with their perceptual limits (Smith, 2009). Note that this setup is more a test of response to aleatoric than epistemic uncertainty.

These experiments have led to a hypothesis that brains both represent probability distributions and perform probabilistic computations (sometimes termed 'the Bayesian brain'). Classically, a neuron's firing rate has been thought to encode a single value of a stimulus' attribute, such as brightness or shape. Under the new hypothesis a *distribution* over the attribute is somehow represented, which can then be used downstream in decision making.

Several biologically plausible schemes to implement this have been proposed (Averbeck et al., 2006; Beck et al., 2008; Fiser et al., 2010; Knill and Pouget, 2004; Pouget et al., 2003), offering connections to Bayesian inference. Probabilistic population codes propose that a probability distribution can be represented through the activity of populations of neurons, whilst sampling based representations suggest the activity of a single neuron sampled over time can represent a distribution.

Although on the surface such schemes appear to be closely related to BNNs - both are modelling uncertainty with neurons - there is a difference. The neuroscience schemes consider how uncertainty is encoded into neuronal activity whilst BNNs consider representing uncertainty in synaptic weights. Whilst there may be ways to reconcile these two ideas, we found only one neuroscience work directly considering synpatic weight uncertainty (Aitchison et al., 2014). This suggests exploring applying the Bayesian framework to a NN at higher levels than directly over parameters.

The topic of uncertainty often arises under the broader concept of metacognition (Crystal and Foote, 2009), which is 'cognition about cognition', or the ability to perceive one's own mental states and cognitive processes (Jozefowiez et al., 2009). Various models for how metacognitive representations might arise have been proposed. For example, confidence could be estimated based on clarity of input stimulus, or it could be inferred based on speed or ease of processing or response. Shea et al. (2014) propose two levels of metacognition, suggesting that human metacognition might be uniquely related to higher level 'system 2' thinking. This is an intriguing suggestion, implying that our uncertainty awareness may be bundled together with other cognitive abilities.

Finally we draw attention to arguments *against* the brain handling uncertainty effectively. Bowers & Davis (2012) provide detailed commentary in this direction. They argue that it is easy to adapt priors and likelihood functions to produce behavioural models of good fit, and that the biologically plausible computational schemes have not been verified through experimental neural firing data (Chen et al., 2006). They also recognise that through evolution biological systems tend to produce 'good enough' solutions, but not optimal solutions (Dawkins, 1982). Work has also demonstrated empirically that Bayesian behaviour can emerge in non-Bayesian NNs in supervised (Emin Orhan and Ma, 2017) and RL (Weisswange et al., 2011) settings. The ability to perform Bayesian inference was shown to improve with age, which is further evidence against Bayesian computation being hardwired into the brain (Chambers et al., 2018; Weisswange et al., 2011).

The discussion has so far examined the perspective that individual biological neurons are approximated by artificial neurons in a BNN. An alternative perspective could consider a BNN as modelling the 'software', rather than the hardware, of the brain, which might point to searching for deeper connections with cognitive science, rather than neuroscience.

Differences and Inspiration

Following from this brief review, we summarise several key observations.

1. Neuroscience models generally assume uncertainty is represented through activity of neurons. BNNs encode uncertainty through probability distributions on the NN parameters.
2. In NN modelling there is a careful distinction between two types of uncertainty; epistemic and aleatoric. The neuroscience literature rarely makes this distinction, with experiments and models appearing to target aleatoric uncertainty (Platt and Huettel, 2008).
3. Uncertainty-awareness is often described in the context of a broader concept: metacognition.
4. There is debate over how Bayesian animal and human behaviour is, and also whether Bayesian behaviour implies processing must be Bayesian, or if this is an emergent property.

These differences suggest several promising directions to explore in future research building uncertainty-aware NNs.

1. Incorporate Bayesian principles into NNs at higher levels than directly over parameters.
2. Further investigate theories about metacognition, and how analogies can be applied in NNs, for example learning of secondary representations based on the neural activity, or measuring ease of processing.
3. Further consider the role of system 2 cognition in uncertainty awareness, and its connection to other problems in machine learning.
4. Precisely describe conditions that lead to emergent Bayesian behaviour from a non-Bayesian NN.
5. Examine connections between BNNs and cognitive science, rather than neuroscience.

6.4 Future Work

A number of of unanswered questions and unsolved challenges emerged over the course of this thesis. Technical chapters closed by suggesting a promising direction for future work related to the idea presented. We highlight some of these below.

Chapter 3: Develop methods enabling BNN priors to be specified at a more abstract level than granular properties, such as length scale. This may require rethinking the paradigm in which we currently specify priors - rather than encoding knowledge directly into parameter priors, it might be more appropriate to communicate priors via expert demonstrations, representative data points, or simple hard-coded rules.

Chapter 4: Revisit classical ensemble theory with deep NN's in mind, answering questions such as: What is the optimal layer sharing scheme? Why does bootstrapping often decrease performance?

This chapter closed the thesis with a more speculative comparison between current approaches to handling uncertainty in NNs and theories about uncertainty in the brain. As artificial systems become gradually more general, it's possible that the way we build mechanisms to handle uncertainty may have to change. Taking inspiration from theories in psychology and neuroscience could provide valuable inspiration for longer term solutions.

6.5 Conclusions

This thesis contributed to the line of research seeking to build NNs capable of providing robust uncertainty estimates alongside their predictions and decisions. This will lead to safer, more robust, and ultimately more useful deep learning systems. Contributions were made to both Bayesian and non-Bayesian approaches.

Bibliography

Aitchison, L. (2020). Why bigger is not always better: on finite and infinite neural networks. In *ICML*.

Aitchison, L., Pouget, A., and Latham, P. E. (2014). Probabilistic Synapses. In *Arxiv 1410.1029*.

Ak, R., Li, Y., Vitelli, V., Zio, E., López Droguett, E., and Magno Couto Jacinto, C. (2013a). NSGA-II-trained neural network approach to the estimation of prediction intervals of scale deposition rate in oil & gas equipment. *Expert Systems with Applications*, 40(4):1205–1212.

Ak, R., Li, Y.-f., Vitelli, V., and Zio, E. (2013b). Multi-objective Genetic Algorithm Optimization of a Neural Network for Estimating Wind Speed Prediction Intervals.

Arora, S., Du, S., Zhiyuan, L., Salakhutdinov, R., Wang, R., and Yu, D. (2020). Harnessing the Power of Infinitely Wide Deep Nets on Small-data Tasks. In *ICLR*.

Arora, S., Du, S. S., Hu, W., Li, Z., Salakhutdinov, R., and Wang, R. (2019). On Exact Computation with an Infinitely Wide Neural Net. In *Neural Information Processing Systems*.

Averbeck, B. B., Latham, P. E., and Pouget, A. (2006). Neural correlations, population coding and computation. *Nature Reviews Neuroscience*, 7(5):358–366.

Azizzadenesheli, K., Brunskill, E., and Anandkumar, A. (2018). Efficient exploration through Bayesian deep Q-networks. *2018 Information Theory and Applications Workshop, ITA 2018*, pages 1–9.

Barber, D., Hill, F., and Bishop, C. M. (1998). Ensemble Learning in Bayesian Neural Networks. *Neural Networks and Machine Learning*.

Bardsley, J. M. (2012). MCMC-based image reconstruction with uncertainty quantification. *SIAM Journal on Scientific Computing*, 34(3):1316–1332.

Bardsley, J. M., Solonen, A., Haario, H., and Laine, M. (2014). Randomize-Then-Optimize: A Method for Sampling from Posterior Distributions in Nonlinear Inverse Problems. *SIAM Journal on Scientific Computing*, 36(4).

Beck, J. M., Ma, W. J., Kiani, R., Hanks, T., Churchland, A. K., Roitman, J., Shadlen, M. N., Latham, P. E., and Pouget, A. (2008). Probabilistic Population Codes for Bayesian Decision Making. *Neuron*, 60(6):1142–1152.

Belkin, M., Hsu, D., Ma, S., and Mandal, S. (2019). Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences of the United States of America*, 116(32):15849–15854.

Bengio, Y. (2009). Learning Deep Architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–127.

Bengio, Y. and Lecun, Y. (2007). Scaling Learning Algorithms towards AI. In *Large-scale kernel machines*. MIT Press.

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

Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstr, D. (2015). Weight Uncertainty in Neural Networks. In *Proceedings of the 32nd International Conference on Machine Learning*.

Bowers, J. S. and Davis, C. J. (2012). Bayesian just-so stories in psychology and neuroscience. *Psychological Bulletin*, 138(3):389–414.

Breiman, L. (1996). Bagging predictors. *Machine Learning*, 24(2):123–140.

Brown, G. (2004). *Diversity in Neural Network Ensembles*. PhD thesis.

Burt, D. R., Rasmussen, C. E., and van der Wilk, M. (2019). Rates of convergence for sparse variational Gaussian process regression. *36th International Conference on Machine Learning, ICML 2019*, 2019-June:1390–1404.

Call, J. (2010). Do apes know that they could be wrong? *Animal Cognition*, 13(5):689–700.

Chambers, C., Sokhey, T., Gaebler-Spira, D., and Kording, K. P. (2018). The development of Bayesian integration in sensorimotor estimation. *Journal of Vision*, 18(12):1–16.

Chen, T., Fox, E. B., and Guestrin, C. (2014). Stochastic gradient Hamiltonian Monte Carlo. *31st International Conference on Machine Learning, ICML 2014*, 5:3663–3676.

Chen, Y., Geisler, W. S., and Seidemann, E. (2006). Optimal decoding of correlated neural population responses in the primate visual cortex. *Nature Neuroscience*.

Chen, Y. and Oliver, D. S. (2012). Ensemble Randomized Maximum Likelihood Method as an Iterative Ensemble Smoother. *International Association for Mathematical Geosciences*, (D):1–26.

Cheng, X., Khomtchouk, B., Matloff, N., and Mohanty, P. (2018). Polynomial Regression As an Alternative to Neural Nets. In *arXiv:1806.06850*.

Cho, Y. and Saul, L. K. (2009). Kernel Methods for Deep Learning. In *Advances in Neural Information Processing Systems 22*.

Chua, K., Calandra, R., McAllister, R., and Levine, S. (2018). Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models. In *NeurIPS*.

Clements, W., Robaglia, B., Van Delft, B., Slaoui, R. B., and Toth, S. (2020). Estimating Risk and Uncertainty in Deep Reinforcement Learning. *arXiv preprint 1905.09638*.

Coker, B., Pradier, M. F., and Doshi-Velez, F. (2019). Towards Expressive Priors for Bayesian Neural Networks: Poisson Process Radial Basis Function Networks. In *arXiv preprint 1912.05779*.

Crystal, J. D. and Foote, A. L. (2009). Metacognition in Animals. *Comparative Cognition & Behavior Reviews*, 4:1–16.

Csato, L. and Opper, M. (2002). Sparse Online Gaussian Processes. *Neural Computation*.

Cui, T., Havulinna, A., Marttinen, P., and Kaski, S. (2020). Informative Gaussian Scale Mixture Priors for Bayesian Neural Networks.

Damianou, A. C. and Lawrence, N. D. (2013). Deep Gaussian Processes, Damianou & Lawrence. *Proceedings of the 16th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 31.

Danafar, S., Gretton, A., and Schmidhuber, J. (2010). Characteristic kernels on structured domains excel in robotics and human action recognition. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6321 LNAI(PART 1):264–279.

Dawkins, R. (1982). *The extended phenotype: The gene as the unit of selection*. Oxfordshire: Freeman.

De, A. G., Matthews, G., Nickson, T., Fujii, K., Boukouvalas, A., León-Villagrá, P., Ghahramani, Z., and Hensman, J. (2017). GPflow: A Gaussian Process Library using TensorFlow Mark van der Wilk. *Journal of Machine Learning Research*, 18:1–6.

Dearden, R., Friedman, N., and Russell, S. (1998). Bayesian Q-learning. In *American Association for Artificial Intelligence (AAAI)*.

Debao, C. (1993). Degree of approximation by superpositions of a sigmoidal function. *Approximation Theory and its Applications*, 9(3):17–28.

Der Kiureghian, A. and Ditlevsen, O. (2008). Aleatoric or epistemic ? Does it matter ? *Special Workshop on Risk Acceptance and Risk Communication*, pages 1–13.

Dietterich, T. G. (2000). Ensemble Methods in Mac hine Learning. *Multiple Classifier Systems*.

Du, Y. and Narasimhan, K. (2019). Task-agnostic dynamics priors for deep reinforcement learning. In *36th International Conference on Machine Learning, ICML 2019*.

Dubey, R., Agrawal, P., Pathak, D., Griffiths, T. L., and Efros, A. A. (2018). Investigating Human Priors for Playing Video Games. In *Proceedings of the 35th International Conference on Machine Learning*.

Duvenaud, D., Maclaurin, D., and Adams, R. P. (2016). Early Stopping as Nonparametric Variational Inference. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 51, pages 1070–1077.

Duvenaud, D., Rippel, O., Adams, R. P., and Ghahramani, Z. (2014). Avoiding pathologies in very deep networks. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*.

Duvenaud, D. K. (2014). *Automatic Model Construction with Gaussian Processes*. PhD thesis.

Dziugaite, G. K. and Roy, D. M. (2017). Computing nonvacuous generalization bounds for deep (stochastic) neural networks with many more parameters than training data. *Uncertainty in Artificial Intelligence - Proceedings of the 33rd Conference, UAI 2017*.

Emin Orhan, A. and Ma, W. J. (2017). Efficient probabilistic inference in generic neural networks trained with non-probabilistic feedback. *Nature Communications*, 8(1).

Filos, A., Farquhar, S., Gomez, A. N., Rudner, T. G. J., Kenton, Z., Smith, L., Alizadeh, M., de Kroon, A., and Gal, Y. (2019). A Systematic Comparison of Bayesian Deep Learning Robustness in Diabetic Retinopathy Tasks. In *Workshop on Bayesian Deep Learning, NeurIPS 2019*.

Finn, C., Xu, K., and Levine, S. (2018). Probabilistic Model-Agnostic Meta-Learning. In *32nd Conference on Neural Information Processing Systems (NIPS 2018)*.

Fiser, J., Berkes, P., Orbán, G., and Lengyel, M. (2010). Statistically optimal perception and learning: from behavior to neural representations: Perceptual learning, motor learning, and automaticity. *Trends in Cognitive Sciences*.

Flam-Shepherd, D., Requeima, J., and Duvenaud, D. (2017). Mapping Gaussian Process Priors to Bayesian Neural Networks. *Bayesian Deep Learning Workshop, Neural Information Processing Systems (NeurIPS)*.

Foong, A. Y. K., Li, Y., Hernández-Lobato, J. M., and Turner, R. E. (2019). 'In-Between' Uncertainty in Bayesian Neural Networks. In *Workshop on Uncertainty and Robustness in Deep Learning, ICML*.

Fort, S., Hu, H., and Lakshminarayanan, B. (2019). Deep Ensembles: A Loss Landscape Perspective. In *ICLR*.

Gaier, A. and Ha, D. (2019). Weight Agnostic Neural Networks. *arXiv preprint: 1906.04358*.

Gal, Y. (2016). *Uncertainty in Deep Learning*. PhD thesis.

Gal, Y. and Ghahramani, Z. (2015). Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. In *Proceedings of the 33rd International Conference on Machine Learning*.

Galván, I. M., Valls, J. M., Cervantes, A., and Aler, R. (2017). Multi-objective evolutionary optimization of prediction intervals for solar energy forecasting with neural networks. *Information Sciences*.

Garnelo, M., Schwarz, J., Rosenbaum, D., Viola, F., Rezende, D. J., Ali Eslami, S. M., and Whye Teh, Y. (2018). Neural Processes.

Garriga-Alonso, A., Rasmussen, C. E., and Aitchison, L. (2019). Deep Convolutional Networks as shallow Gaussian Processes. In *ICLR*.

Gelman, A. (2008). Objections to Bayesian statistics. *Bayesian Analysis*, 3(3):445–450.

Ghahramani, Z. (2015). Probabilistic machine learning and artificial intelligence. *Nature*.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*.

Gu, Y., Oliver, D. S., and Oklahoma, U. (2007). An Iterative Ensemble Kalman Filter for Multiphase Fluid Flow Data Assimilation. *SPE Journal 12(4)*.

Gustafsson, F. K., Danelljan, M., and Schön, T. B. (2019). Evaluating Scalable Bayesian Deep Learning Methods for Robust Computer Vision. *arXiv preprint 1906.01620*.

Hafner, D., Tran, D., Irpan, A., Lillicrap, T., and Davidson, J. (2018). Reliable Uncertainty Estimates in Deep Neural Networks using Noise Contrastive Priors.

Hansen, L. K. A. I. and Salamon, P. (1990). Neural Network Ensembles. *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, 12(October).

Hassabis, D., Kumaran, D., Summerfield, C., and Botvinick, M. (2017). Neuroscience-Inspired Artificial Intelligence. *Neuron*, 95(2):245–258.

He, B., Lakshminarayanan, B., and Teh, Y. W. (2020). Bayesian Deep Ensembles via the Neural Tangent Kernel. In *ArXiv 2007.05864*.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classificatio. In *ICCV*, pages 1026–1034.

Hebb, D. O. (1950). *The Organization of Behavior; A Neuropsychological Theory*, volume 63.

Hernández-Lobato, J. M. and Adams, R. P. (2015). Probabilistic Backpropagation for Scalable Learning of Bayesian Neural Networks. In *Proceedings of the 32nd International Conference on Machine Learning*.

Heskes, T. (1996). Practical confidence and prediction intervals. In *Advances in Neural Information Processing Systems 9*.

Hron, J., Matthews, A. G. d. G., and Ghahramani, Z. (2018). Variational Bayesian dropout: pitfalls and fixes. In *Proceedings of the 35th International Conference on Machine Learning*.

Hüllermeier, E. and Waegeman, W. (2019). Aleatoric and Epistemic Uncertainty in Machine Learning: A Tutorial Introduction. pages 1–46.

Jacot, A., Gabriel, F., and Hongler, C. (2018). Neural tangent kernel: Convergence and generalization in neural networks. *Advances in Neural Information Processing Systems*, 2018-Decem(5):8571–8580.

Jones, K. O. (2005). Comparison of Genetic Algorithm and Particle Swarm Optimisation. *International Conference on Computer Systems and Technologies - CompSysTech'2005 COMPARISON*, pages 1–6.

Jozefowiez, J., Staddon, J. E. R., and Cerutti, D. T. (2009). Metacognition in animals: how do we know that they know? *Comparative Cognition & Behavior Reviews*, 4:29–39.

Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. *Neural Networks, 1995. Proceedings., IEEE International Conference on*, 4:1942–1948 vol.4.

Kennedy, J. and Eberhart, R. (2001). *Swarm Intelligence*.

Khan, M. E., Nielsen, D., Tangkaratt, V., Lin, W., Gal, Y., and Srivastava, A. (2018). Fast and Scalable Bayesian Deep Learning by Weight-Perturbation in Adam. In *ICML*.

Khosravi, A., Nahavandi, S., Creighton, D., and Atiya, A. F. (2011a). A Comprehensive Review of Neural Network-based Prediction Intervals and New Advances. *IEEE Transactions on Neural Networks*, 22(9):1341–56.

Khosravi, A., Nahavandi, S., Creighton, D., and Atiya, A. F. (2011b). Lower upper bound estimation method for construction of neural network-based prediction intervals. *IEEE Transactions on Neural Networks*, 22(3):337–346.

Kingma, D. P. and Ba, J. (2015). Adam: A Method for Stochastic Optimization. *Iclr*, pages 1–15.

Kivaranovic, D., Johnson, K. D., and Leeb, H. (2020). Adaptive, Distribution-Free Prediction Intervals for Deep Networks. In *AISTATS*.

Knill, D. C. and Pouget, A. (2004). The Bayesian brain: The role of uncertainty in neural coding and computation. *Trends in Neurosciences*, 27(12):712–719.

Koenker, R. and Hallock, K. F. (2001). Quantile Regression. *Journal of Economic Perspectives*, 15(4):143–156.

Kording, K. and Wolpert, D. (2004). Bayesian integration in sensorimotor learning. *Nature Letters*.

Krishnan, R., Subedar, M., and Tickoo, O. (2019). MOPED: Efficient priors for scalable variational inference in Bayesian deep neural networks. In *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*.

Krizhevsky, A., Sutskever, I., and Hinton, G. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In *NeurIPS*.

Lakshminarayanan, B., Pritzel, A., and Blundell, C. (2017). Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles. In *31st Conference on Neural Information Processing Systems*.

Lee, J., Bahri, Y., Novak, R., Schoenholz, S. S., Pennington, J., and Sohl-dickstein, J. (2018). Deep neural networks as gaussian processes. In *ICLR 2018*.

Lee, J., Xiao, L., Schoenholz, S. S., Bahri, Y., Novak, R., Sohl-Dickstein, J., and Pennington, J. (2019). Wide Neural Networks of Any Depth Evolve as Linear Models Under Gradient Descent. In *NeurIPS*.

Lee, S., Purushwalkam, S., Cogswell, M., Crandall, D., and Batra, D. (2015). Why M Heads are Better than One: Training a Diverse Ensemble of Deep Networks.

Lian, C., Zeng, Z., Member, S., Yao, W., Tang, H., Lung, C., and Chen, P. (2016). Landslide Displacement Prediction With Uncertainty Based on Neural Networks With Random Hidden Weights. *IEEE Transactions on Neural Networks and Learning Systems*, 27(12):1–13.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2016). Continuous control with deep reinforcement learning. In *ICLR*.

Liu, Q. and Wang, D. (2016). Stein variational gradient descent: A general purpose Bayesian inference algorithm. *Advances in Neural Information Processing Systems*, pages 2378–2386.

Liu, Y. and Yao, X. (1999). Ensemble learning via negative correlation. *Neural Networks*, 12(10):1399–1404.

López-Muñoz, F., Boya, J., and Alamo, C. (2006). Neuron theory, the cornerstone of neuroscience, on the centenary of the Nobel Prize award to Santiago Ramón y Cajal. *Brain Research Bulletin*, 70(4-6):391–405.

Lu, X. and Van Roy, B. (2017). Ensemble Sampling. In *31st Conference on Neural Information Processing Systems*.

Ma, C., Li, Y., and Hernández-Lobato, J. M. (2019). Variational Implicit Processes. In *Proceedings of the 36th International Conference on Machine Learning*.

MacKay, D. (1995). Probable networks and plausible predictions - a review of practical Bayesian methods for supervised neural networks.

MacKay, D. J. C. (1992a). A Practical Bayesian Framework for Backpropagation Networks. *Neural Computation*, 4(3):448–472.

MacKay, D. J. C. (1992b). *Bayesian Methods for Adaptive Models*. PhD thesis.

MacKay, D. J. C. (1998). *Introduction to Gaussian Processes*. Springer-Verlag.

Malinin, A. and Gales, M. (2018). Predictive Uncertainty Estimation via Prior Networks. In *Neural Information Processing Systems*.

Matthews, A., Hron, J., Rowland, M., Turner, R., and Ghahramani, Z. (2018). Gaussian Process Behaviour in Wide Deep Networks. In *ICLR*, pages 1–15.

McCulloch, W. and Pitts, W. (1943). A logical calculus nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133.

Menda, K., Driggs-campbell, K., and Kochenderfer, M. J. (2019). EnsembleDAgger: A Bayesian Approach to Safe Imitation Learning. *Arxiv 1807.08364*.

Mnih, V., Antonoglou, I., Fidjeland, A. K., Wierstra, D., King, H., Bellemare, M. G., Legg, S., Petersen, S., Riedmiller, M., Beattie, C., Graves, A., Sadik, A., Kavukcuoglu, K., Ostrovski, G., Veness, J., Rusu, A. A., Silver, D., Hassabis, D., and Kumaran, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

Mukhoti, J., Stenetorp, P., and Gal, Y. (2018). On the Importance of Strong Baselines in Bayesian Deep Learning. In *Bayesian Deep Learning Workshop, Neural Information Processing Systems (NeurIPS)*, pages 1–4.

Nagabandi, A., Kahn, G., Fearing, R. S., and Levine, S. (2017). Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning. In *Deep Reinforcement Learning Symposium, NIPS 2017*.

Nalisnick, E. T. (2018). *On Priors for Bayesian Neural Networks*. PhD thesis.

Neal, R. M. (1997). *Bayesian Learning for Neural Networks*. PhD thesis.

Nguyen, C. V., Li, Y., Bui, T. D., and Turner, R. E. (2018). Variational Continual Learning. In *ICLR 2018*.

Nix, D. and Weigend, A. (1994). Estimating the mean and variance of the target probability distribution. In *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*, pages 55–60 vol.1.

Novak, R., Xiao, L., Lee, J., Bahri, Y., Abolafia, D. A., Pennington, J., and Sohl-dickstein, J. (2019). Bayesian Deep Convolutional Networks with Many Channels are Gaussian Processes. In *ICLR 2019*.

Osband, I., Aslanides, J., and Cassirer, A. (2018). Randomized Prior Functions for Deep Reinforcement Learning. In *32nd Conference on Neural Information Processing Systems (NIPS 2018)*.

Osband, I., Blundell, C., Pritzel, A., and Van Roy, B. (2016). Deep Exploration via Bootstrapped DQN. In *Advances in neural information processing systems*, pages 1–18.

Osband, I., Russo, D., Wen, Z., and Van Roy, B. (2019). Deep Exploration via Randomized Value Functions. *JMLR*.

Osband, I. and Van Roy, B. (2015). Bootstrapped Thompson Sampling and Deep Exploration.

Papadopoulos, G., Edwards, P. J., and Murray, A. F. (2000). Confidence Estimation Methods for Neural Networks: A Practical Comparison. In *European Symposium on Artificial Neural Networks*.

Parascandolo, G., Huttunen, H., and Virtanen, T. (2017). Taming the Waves: Sine As Activation Function in Deep Neural Networks. *openreview preprint*.

Pearce, T., Anastassacos, N., Zaki, M., and Neely, A. (2018a). Bayesian Inference with Anchored Ensembles of Neural Networks, and Application to Exploration in Reinforcement Learning. In *Exploration in Reinforcement Learning Workshop, ICML*.

Pearce, T., Brintrup, A., and Neely, A. (2019a). Uncertainty in Neural Networks: Approximately Bayesian Ensembling. In *AISTATS*.

Pearce, T., Brintrup, A., and Neely, A. (2020a). A Neuroscience-inspired Approach to Building Uncertainty-aware AI. In *NAISys*.

Pearce, T., Foong, A. Y. K., and Brintrup, A. (2020b). Structured Weight Priors for Convolutional Neural Networks. In *Workshop on Uncertainty and Robustness in Deep Learning, ICML*.

Pearce, T., Tsuchida, R., Zaki, M., Brintrup, A., and Neely, A. (2019b). Expressive Priors in Bayesian Neural Networks: Kernel Combinations and Periodic Functions. In *Uncertainty in Artifical Intelligence (UAI)*.

Pearce, T., Zaki, M., Brintrup, A., and Neely, A. (2018b). High-Quality Prediction Intervals for Deep Learning: A Distribution-Free, Ensembled Approach. In *Proceedings of the 35th International Conference on Machine Learning , ICML*, Stockholm.

Pedersen, M. S., Baxter, B., Templeton, B., Rishøj, C., Theobald, D. L., Hoegh-rasmussen, E., Casteel, G., Gao, J. B., Dedecius, K., Strim, K., Christiansen, L., Hansen, L. K., Wilkinson, L., He, L., Bar, M., Winther, O., Sakov, P., Hattinger, S., Petersen, K. B., and Rishø j, C. (2008). The Matrix Cookbook. *Matrix*, M:1–71.

Pfeiffer, M. and Pfeil, T. (2018). Deep Learning With Spiking Neurons: Opportunities and Challenges. *Frontiers in Neuroscience*, 12(October).

Ping, W., Peng, K., Gibiansky, A., Arık, S., Kannan, A., Narang, S., Raiman, J., and Miller, J. (2018). Deep Voice 3: Scaling text-to-speech with convolutional sequence learning. In *ICLR*.

Pinson, P. and Kariniotakis, G. (2013). Optimal Prediction Intervals of Wind Power Generation. *IEEE Transactions on Power Systems*, 25:1845–1856.

Platt, M. L. and Huettel, S. A. (2008). Risky business: the neuroeconomics of decision making under uncertainty. *Nature Neuroscience*, 11(4):398–403.

Poggio, T., Mhaskar, H., Rosasco, L., Miranda, B., and Liao, Q. (2017). Why and when can deep-but not shallow-networks avoid the curse of dimensionality: A review. *International Journal of Automation and Computing*, 14(5):503–519.

Poole, B., Lahiri, S., Raghu, M., Sohl-Dickstein, J., and Ganguli, S. (2016). Exponential expressivity in deep neural networks through transient chaos. *Advances in Neural Information Processing Systems*, (Nips):3368–3376.

Pouget, A., Dayan, P., and Zemel, R. S. (2003). Inference and computation with population codes. *Annual Review of Neuroscience*, 26(1):381–410.

Quan, H., Srinivasan, D., and Khosravi, A. (2014). Uncertainty handling using neural network-based prediction intervals for electrical load forecasting. *Energy*, 73:916–925.

Quiñonero-Candela, J. and Rasmussen, C. E. (2005). A unifying view of sparse approximate Gaussian process regression. *Journal of Machine Learning Research*, 6:1939–1959.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2018). Language Models are Unsupervised Multitask Learners.

Ramachandran, P., Zoph, B., and Le, Q. V. (2017). Searching for Activation Functions. *arXiv preprint 1710.05941.*

Rao, A. S. and Verweij, G. (2017). What ' s the real value of AI for your business and how can you capitalise ? Technical report.

Rasmussen, C. E. and Williams, C. K. I. (2006). *Gaussian Processes for Machine Learning.*

Ritter, H., Botev, A., and Barber, D. (2018). A Scalable Laplace Approximation for Neural Networks. In *ICLR*, pages 1–15.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3):211–252.

Salem, T. S., Langseth, H., and Ramampiaro, H. (2020). Prediction Intervals : Split Normal Mixture from Quality-Driven Deep Ensembles. In *Uncertainty in Artifical Intelligence (UAI).*

Saul, A. D., Hensman, J., and Lawrence, N. D. (2016). Chained Gaussian Processes. In *19th International Conference on Artificial Intelligence and Statistics (AISTATS) 2016.*

Schoenholz, S. S., Gilmer, J., Ganguli, S., and Sohl-Dickstein, J. (2016). Deep Information Propagation. (2016):1–18.

Sengupta, U., Rasmussen, C. E., and Juniper, M. P. (2020). Bayesian machine learning for the prognosis of combustion instabilities from noise. In *Proceedings of the ASME 2020 Turbomachinery Technical Conference Exposition.*

Shafer, G. and Vovk, V. (2008). A tutorial on conformal prediction. *Journal of Machine Learning Research*, 9:371–421.

Shea, N., Boldt, A., Bang, D., Yeung, N., Heyes, C., and Frith, C. D. (2014). Supra-personal cognitive control and metacognition. *Trends in Cognitive Sciences*, 18(4):186–193.

Shen, Y., Wang, X., and Chen, J. (2018). Wind Power Forecasting Using Multi-Objective Evolutionary Algorithms for Wavelet Neural Network-Optimized Prediction Intervals. *Applied Sciences*, 8(2):185.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., and Hassabis, D. (2017). Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm.

Smith, J. D. (2009). The study of animal metacognition. *Trends in Cognitive Sciences*, 13(9):389–396.

Smith, J. D., Shields, W. E., and Schull, J. (1997). The uncertain response in humans and animals. *Cognition*, 62:75–97.

Smith, J. D., Strote, J., Erb, L., Egnor, R., Schull, J., and McGee, K. (1995). The uncertain response in the bottlenosed dolphin (Tursiops truncatus). *Journal of Experimental Psychology: General*, 124(4):391–408.

Smith, L. and Gal, Y. (2018). Understanding Measures of Uncertainty for Adversarial Example Detection. In *Uncertainty in Artifical Intelligence (UAI)*.

Snelson, E. and Ghahramani, Z. (2006). Sparse Gaussian Processes using Pseudo-inputs. In *Advances in Neural Information Processing Systems*.

Springenberg, J. T., Klein, A., Falkner, S., and Hutter, F. (2016). Bayesian optimization with Robust Bayesian neural networks. *Advances in Neural Information Processing Systems*.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15:1929–1958.

Sun, S., Zhang, G., Shi, J., and Grosse, R. (2019). Functional Variational Bayesian Neural Networks. In *ICLR 2019*.

Sun, S., Zhang, G., Wang, C., Zeng, W., Li, J., and Grosse, R. (2018). Differentiable compositional kernel learning for Gaussian processes. *35th International Conference on Machine Learning, ICML 2018*, 11:7676–7696.

Sun, X., Wang, Z., and Hu, J. (2017). Prediction Interval Construction for Byproduct Gas Flow Forecasting Using Optimized Twin Extreme Learning Machine. *Mathematical Problems in Engineering*, 2017.

Tagasovska, N. and Lopez-paz, D. (2018). Frequentist uncertainty estimates for deep learning. In *Bayesian Deep Learning Workshop, Neural Information Processing Systems (NeurIPS)*.

Tagasovska, N. and Lopez-Paz, D. (2019). Single-Model Uncertainties for Deep Learning. (NeurIPS).

Taylor, J. W. (2000). A Quantile Regression Neural Network Approach to Estimating the Conditional Density of Multiperiod Returns. *Journal of Forecasting*, 19:299–311.

Thomas, P., Mansot, J., Delbe, K., Sauldubois, A., and Bilas, P. (2012). Standard Particle Swarm Optimisation.

Tibshirani, R. (1996). A Comparison of Some Error Estimates for Neural Network Models. *Neural Computation*, 8:152–163.

Titsias, M. K. (2009). Variational Learning of Inducing Variables in Sparse Gaussian Processes. *AISTATS*.

Tomczak, M. B., Swaroop, S., and Turner, R. E. (2018). Neural network ensembles and variational inference revisited. In *1st Symposium on Advances in Approximate Bayesian Inference*.

Tsuchida, R., Pearce, T., van der Heide, C., Roosta, F., and Gallagher, M. (2020). Avoiding Kernel Fixed Points: Computing with ELU and GELU Infinite Networks. *AAAI*.

Tsuchida, R., Roosta-Khorasani, F., and Gallagher, M. (2018). Invariance of Weight Distributions in Rectified MLPs. In *Proceedings of the 35th International Conference on Machine Learning*.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., Oh, J., Horgan, D., Kroiss, M., Danihelka, I., Huang, A., Sifre, L., Cai, T., Agapiou, J. P., Jaderberg, M., Vezhnevets, A. S., Leblond, R., Pohlen, T., Dalibard, V., Budden, D., Sulsky, Y., Molloy, J., Paine, T. L., Gulcehre, C., Wang, Z., Pfaff, T., Wu, Y., Ring, R., Yogatama, D., Wünsch, D., McKinney, K., Smith, O., Schaul, T., Lillicrap, T., Kavukcuoglu, K., Hassabis, D., Apps, C., and Silver, D. (2019). Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782):350–354.

Wang, B., Li, T., Yan, Z., Zhang, G., and Lu, J. (2020). DeepPIPE : A distribution-free uncertainty quantification approach for time series forecasting. *Neurocomputing*.

Wang, J., Fang, K., Pang, W., and Sun, J. (2017). Wind power interval prediction based on improved PSO and BP neural network. *Journal of Electrical Engineering and Technology*, 12(3):989–995.

Weisswange, T. H., Rothkopf, C. A., Rodemann, T., and Triesch, J. (2011). Bayesian cue integration as a developmental outcome of reward mediated learning. *PLoS ONE*, 6(7).

Williams, C. K. I. (1996). Computing with infinite networks. In *Advances in Neural Information Processing Systems 9*.

Wilson, A. G. and Adams, R. P. (2013). Gaussian Process Kernels for Pattern Discovery and Extrapolation. In *Proceedings of the 30th International Conference on Machine Learning*.

Wilson, A. G. and Izmailov, P. (2020). Bayesian Deep Learning and a Probabilistic Perspective of Generalization. *arXiv preprint*.

Wu, A., Nowozin, S., Meeds, E., Turner, R. E., Hernández-Lobato, J. M., and Gaunt, A. L. (2019). Deterministic variational inference for robust Bayesian neural networks. *7th International Conference on Learning Representations, ICLR 2019*.

Yan, L., Verbel, D., and Saidi, O. (2004). Predicting prostate cancer recurrence via maximizing the concordance index. In *Proceedings of the 2004 ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '04*, Seattle, Washington.

Yang, G. (2019). Tensor Programs I: Wide Feedforward or Recurrent Neural Networks of Any Architecture are Gaussian Processes. (NeurIPS):1–73.

Yang, G. and Schoenholz, S. S. (2017). Mean Field Residual Networks: On the Edge of Chaos. (Nips).

Yao, Y., Vehtari, A., Simpson, D., and Gelman, A. (2018). Yes, but Did It Work?: Evaluating Variational Inference. In *ICML*.

Yong, B. X., Fathy, Y., and Brintrup, A. (2020). Bayesian autoencoders for drift detection inindustrial environments.

Zhu, L., Lu, J., and Chen, Y. (2019). HDI-Forest : Highest Density Interval Regression Forest. In *IJCAI*.

Appendix A

Bayesian Ensembling

A.1 Proofs

Definition 1. Data likelihood and parameter likelihood

We take care to define two versions of the likelihood, one in output space, $P_{\mathcal{D}}(\mathcal{D}|\boldsymbol{\theta})$ (data likelihood), and one in parameter space, $P_{\boldsymbol{\theta}}(\mathcal{D}|\boldsymbol{\theta})$ (parameter likelihood). Both return the same values given some data set $\mathcal{D}$ and parameter values $\boldsymbol{\theta}$, and hence are exchangeable, but their forms are subtly different.

The data likelihood, $P_{\mathcal{D}}(\mathcal{D}|\boldsymbol{\theta})$, is defined on the output domain. Typically the log of this, $\log(P_{\mathcal{D}}(\mathcal{D}|\boldsymbol{\theta}))$, might be optimised as the cross entropy loss or (negative) mean squared error.

In contrast, $P_{\boldsymbol{\theta}}(\mathcal{D}|\boldsymbol{\theta})$ defines a likelihood function in the parameter domain.

Illustrative Example

Consider a linear regression model with dataset, $\mathcal{D}$, consisting of tuples, $\{\mathbf{x}, y\}$; a vector of predictor variables $\mathbf{x} \in \mathbb{R}^p$, predicting a single scalar $y \in \mathbb{R}$. If the model is of the form, $\boldsymbol{\theta}^T\mathbf{x}$, a Gaussian data likelihood with variance σ_ϵ^2 on the output might be assumed.

This leads to a data likelihood for the target, y,

$$P_{\mathcal{D}}(\mathcal{D}|\boldsymbol{\theta}) = \mathcal{N}(y|\boldsymbol{\theta}^T\mathbf{x}, \sigma_\epsilon^2). \tag{A.1}$$

For this linear model, the corresponding parameter likelihood is a multivariate normal distribution,

$$P_{\boldsymbol{\theta}}(\mathcal{D}|\boldsymbol{\theta}) \propto \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu}_{\text{like}}, \boldsymbol{\Sigma}_{\text{like}}). \tag{A.2}$$

where, $\boldsymbol{\mu}_{\text{like}} \in \mathbb{R}^{p}$ & $\boldsymbol{\Sigma}_{\text{like}} \in \mathbb{R}^{p\times p}$, can be found analytically. They are implicitly functions of the dataset, $\mathcal{D}$, although to lighten notation we do not write this. Subsequently we also drop the explicit referral to $\boldsymbol{\theta}$.

Note that whilst both the data and parameter likelihood follow a normal distribution, they are defined in different domains.

The correspondence between a Gaussian data likelihood and multivariate normal parameter likelihood is only exact for a linear regression model. For non-linear models with Gaussian data likelihoods, and other data likelihoods, the parameter likelihood is not in general multivariate normal. Nevertheless it can be convenient to model it as such.

Standard Result 1. Product of two multivariate Gaussians (§8.1.8, The Matrix Cookbook, 2008)

$$\mathcal{N}(\boldsymbol{\mu}_{\text{like}},\boldsymbol{\Sigma}_{\text{like}})\mathcal{N}(\boldsymbol{\mu}_{\text{prior}},\boldsymbol{\Sigma}_{\text{prior}}) \propto \mathcal{N}(\boldsymbol{\mu}_{\text{post}},\boldsymbol{\Sigma}_{\text{post}}) \tag{A.3}$$

$$\boldsymbol{\Sigma}_{\text{post}} = (\boldsymbol{\Sigma}_{\text{prior}}^{-1} + \boldsymbol{\Sigma}_{\text{like}}^{-1})^{-1}, \tag{A.4}$$

$$\boldsymbol{\mu}_{\text{post}} = \boldsymbol{\Sigma}_{\text{post}}\boldsymbol{\Sigma}_{\text{prior}}^{-1}\boldsymbol{\mu}_{\text{prior}} + \boldsymbol{\Sigma}_{\text{post}}\boldsymbol{\Sigma}_{\text{like}}^{-1}\boldsymbol{\mu}_{\text{like}}. \tag{A.5}$$

Standard Result 2. Affine transform of a normal random variable (§8.1.4, The Matrix Cookbook, 2008)

$$\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu},\boldsymbol{\Sigma}), \tag{A.6}$$

$$\mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{b}, \tag{A.7}$$

$$\mathbf{y} \sim \mathcal{N}(\mathbf{A}\boldsymbol{\mu} + \mathbf{b}, \mathbf{A}\boldsymbol{\Sigma}\mathbf{A}^{T}). \tag{A.8}$$

Theorem 1. *Assume that a model's parameter likelihood follows a multivariate normal distribution, $P_{\boldsymbol{\theta}}(\mathcal{D}|\boldsymbol{\theta}) \propto \mathcal{N}(\boldsymbol{\mu}_{like},\boldsymbol{\Sigma}_{like})$, and the prior also, $P(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\mu}_{prior},\boldsymbol{\Sigma}_{prior})$. The posterior is then also multivariate normal, $P(\boldsymbol{\theta}|\mathcal{D}) = \mathcal{N}(\boldsymbol{\mu}_{post},\boldsymbol{\Sigma}_{post})$.*

Further assume availability of some function which returns MAP parameter estimates taking as input the location of the prior centre, $\boldsymbol{f}_{MAP}(\boldsymbol{\theta}_{anc})$. In order that, $P(\boldsymbol{f}_{MAP}(\boldsymbol{\theta}_{anc})) = P(\boldsymbol{\theta}|\mathcal{D})$, then the required distribution of $\boldsymbol{\theta}_{anc}$ is also multivariate normal, $P(\boldsymbol{\theta}_{anc}) = \mathcal{N}(\boldsymbol{\mu}_{anc},\boldsymbol{\Sigma}_{anc})$, where, $\boldsymbol{\mu}_{anc} = \boldsymbol{\mu}_{prior}$, and, $\boldsymbol{\Sigma}_{anc} = \boldsymbol{\Sigma}_{prior} + \boldsymbol{\Sigma}_{prior}\boldsymbol{\Sigma}_{like}^{-1}\boldsymbol{\Sigma}_{prior}$.

Proof. Consider a model's parameters having a multivariate normal prior,

$$P(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\mu}_{\text{prior}},\boldsymbol{\Sigma}_{\text{prior}}), \tag{A.9}$$

where, $\boldsymbol{\theta} \in \mathbb{R}^p$, $\boldsymbol{\mu}_{\text{prior}} \in \mathbb{R}^p$, $\boldsymbol{\Sigma}_{\text{prior}} \in \mathbb{R}^{p \times p}$.

This theorem makes the assumption that the form of the parameter likelihood (def. 1) is multivariate normal,

$$P_{\boldsymbol{\theta}}(\mathcal{D}|\boldsymbol{\theta}) \propto \mathcal{N}(\boldsymbol{\mu}_{\text{like}}, \boldsymbol{\Sigma}_{\text{like}}) \tag{A.10}$$

where, $\boldsymbol{\mu}_{\text{like}} \in \mathbb{R}^p$, $\boldsymbol{\Sigma}_{\text{like}} \in \mathbb{R}^{p \times p}$. Here $\propto$ is used since it is not a true probability distribution in $\boldsymbol{\theta}$ so need not sum to 1.

The posterior is calculated by Bayes' rule. Recalling that data likelihood and parameter likelihood are exchangeable (def. 1), and using Standard Result 1,

$$P(\boldsymbol{\theta}|\mathcal{D}) = \frac{P_{\mathcal{D}}(\mathcal{D}|\boldsymbol{\theta})P(\boldsymbol{\theta})}{P(\mathcal{D})} = \frac{P_{\boldsymbol{\theta}}(\mathcal{D}|\boldsymbol{\theta})P(\boldsymbol{\theta})}{P(\mathcal{D})} \tag{A.11}$$

$$\propto \mathcal{N}(\boldsymbol{\mu}_{\text{like}}, \boldsymbol{\Sigma}_{\text{like}})\mathcal{N}(\boldsymbol{\mu}_{\text{prior}}, \boldsymbol{\Sigma}_{\text{prior}}) \propto \mathcal{N}(\boldsymbol{\mu}_{\text{post}}, \boldsymbol{\Sigma}_{\text{post}}), \tag{A.12}$$

where, $\boldsymbol{\mu}_{\text{post}}$ & $\boldsymbol{\Sigma}_{\text{post}}$ are given by eq. A.5 & A.4.

We introduce a further distribution, termed 'anchor distribution', which we enforce as multivariate normal,

$$P(\boldsymbol{\theta}_{anc}) = \mathcal{N}(\boldsymbol{\mu}_{anc}, \boldsymbol{\Sigma}_{anc}). \tag{A.13}$$

It will be used as described in the main text (see figure 4.2 and algorithm 1) so that samples are drawn from the anchor distribution, $\boldsymbol{\theta}_{anc} \sim P(\boldsymbol{\theta}_{anc})$, with a prior then recentred at each sample, denoted $P_{anc}(\boldsymbol{\theta})$,

$$P_{anc}(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}_{anc}, \boldsymbol{\Sigma}_{\text{prior}}). \tag{A.14}$$

Note that this anchor distribution is in the same position as a hyperprior on $\boldsymbol{\mu}_{\text{prior}}$, but will have a subtly different role. $\boldsymbol{\Sigma}_{\text{prior}}$ is unchanged from eq. A.9,

Denote $\boldsymbol{f}_{\text{MAP}}(\boldsymbol{\theta}_{anc})$ as the MAP estimates given this recentred prior and the original likelihood from eq. A.10.

$$\boldsymbol{f}_{\text{MAP}}(\boldsymbol{\theta}_{anc}) := \text{argmax}_{\boldsymbol{\theta}} P_{anc}(\boldsymbol{\theta}) P_{\boldsymbol{\theta}}(\mathcal{D}|\boldsymbol{\theta}) \tag{A.15}$$

In order to prove the theorem, three things regarding $\boldsymbol{f}_{\text{MAP}}(\boldsymbol{\theta}_{anc})$ must be shown:

1. Its distribution is multivariate normal - denote mean and covariance $\boldsymbol{\mu}_{\text{post}}^{\text{RMS}}, \boldsymbol{\Sigma}_{\text{post}}^{\text{RMS}}$,

$$P(\boldsymbol{f}_{\text{MAP}}(\boldsymbol{\theta}_{anc})) = \mathcal{N}(\boldsymbol{\mu}_{\text{post}}^{\text{RMS}}, \boldsymbol{\Sigma}_{\text{post}}^{\text{RMS}}), \tag{A.16}$$

2. That $\boldsymbol{\mu}_{anc}$ & Σ_{anc} can be selected in such a way that the mean of the distribution is equal to that of the original posterior

$$\boldsymbol{\mu}_{\text{post}}^{\text{RMS}} = \boldsymbol{\mu}_{\text{post}}, \tag{A.17}$$

3. And also so that the covariance of the distribution is equal to that of the original posterior

$$\boldsymbol{\Sigma}_{\text{post}}^{\text{RMS}} = \boldsymbol{\Sigma}_{\text{post}}. \tag{A.18}$$

For a multivariate normal distribution, the MAP solution is simply equal to the mean of the posterior, $\boldsymbol{\mu}_{\text{post}}$. For the typical case this is given by eq. A.5. In our procedure, the location of the prior mean has been replaced by $\boldsymbol{\theta}_{anc}$, so the MAP solution is given by,

$$\boldsymbol{f}_{\text{MAP}}(\boldsymbol{\theta}_{anc}) = \boldsymbol{\Sigma}_{\text{post}}\boldsymbol{\Sigma}_{\text{prior}}^{-1}\boldsymbol{\theta}_{anc} + \boldsymbol{\Sigma}_{\text{post}}\boldsymbol{\Sigma}_{\text{like}}^{-1}\boldsymbol{\mu}_{\text{like}} \tag{A.19}$$

$$= \mathbf{A}_1\boldsymbol{\theta}_{anc} + \mathbf{b}_1 \tag{A.20}$$

where two constants have been defined for convenience,

$$\mathbf{A}_1 = \boldsymbol{\Sigma}_{\text{post}}\boldsymbol{\Sigma}_{\text{prior}}^{-1} \tag{A.21}$$

$$\mathbf{b}_1 = \boldsymbol{\Sigma}_{\text{post}}\boldsymbol{\Sigma}_{\text{like}}^{-1}\boldsymbol{\mu}_{\text{like}}, \tag{A.22}$$

which is the same form as eq. A.7. Hence, from Standard Result 2, if $\boldsymbol{\theta}_{anc}$ is normally distributed, $\boldsymbol{f}_{\text{MAP}}(\boldsymbol{\theta}_{anc})$ **will also be normally distributed**.

Regarding the mean of $\boldsymbol{f}_{\text{MAP}}(\boldsymbol{\theta}_{anc})$, we have,

$$\mathbb{E}[\boldsymbol{f}_{\text{MAP}}(\boldsymbol{\theta}_{anc})] = \mathbb{E}[\mathbf{A}_1\boldsymbol{\theta}_{anc} + \mathbf{b}_1] \tag{A.23}$$

$$= \mathbf{A}_1\mathbb{E}[\boldsymbol{\theta}_{anc}] + \mathbf{b}_1. \tag{A.24}$$

By choosing the anchor distribution to be centred about the original prior, $\mathbb{E}[\boldsymbol{\theta}_{anc}] = \boldsymbol{\mu}_{\text{prior}}$, we have,

$$= \mathbf{A}_1\boldsymbol{\mu}_{\text{prior}} + \mathbf{b}_1 \tag{A.25}$$

$$= \boldsymbol{\Sigma}_{\text{post}}\boldsymbol{\Sigma}_{\text{prior}}^{-1}\boldsymbol{\mu}_{\text{prior}} + \boldsymbol{\Sigma}_{\text{post}}\boldsymbol{\Sigma}_{\text{like}}^{-1}\boldsymbol{\mu}_{\text{like}}, \tag{A.26}$$

This is consistent with eq. A.5 and proves that **the means of the distributions are aligned when $\boldsymbol{\mu}_{anc} = \boldsymbol{\mu}_{\text{prior}}$.**

Finally we consider the variance of $\boldsymbol{f}_{\text{MAP}}(\boldsymbol{\theta}_{anc})$, which we wish to equal $\boldsymbol{\Sigma}_{\text{post}}$ by choosing $\boldsymbol{\Sigma}_{anc}$. Using the form from eq. A.20 and applying Standard Result 2,

$$\mathbb{V}\text{ar}[\boldsymbol{f}_{\text{MAP}}(\boldsymbol{\theta}_{anc})] = \mathbb{V}\text{ar}[\mathbf{A}_1\boldsymbol{\theta}_{anc} + \mathbf{b}_1] \tag{A.27}$$

$$= \mathbf{A}_1\mathbb{V}\text{ar}[\boldsymbol{\theta}_{anc}]\mathbf{A}_1^T \tag{A.28}$$

$$= \mathbf{A}_1\boldsymbol{\Sigma}_{anc}\mathbf{A}_1^T \tag{A.29}$$

We require $\mathbb{V}\text{ar}[\boldsymbol{f}_{\text{MAP}}(\boldsymbol{\theta}_{anc})] = \boldsymbol{\Sigma}_{\text{post}}$.

$$\boldsymbol{\Sigma}_{\text{post}} = \mathbf{A}_1\boldsymbol{\Sigma}_{anc}\mathbf{A}_1^T. \tag{A.30}$$

Note that transposes of covariance matrices may be ignored since they are symmetric.

$$\boldsymbol{\Sigma}_{anc} = \mathbf{A_1}^{-1}\boldsymbol{\Sigma}_{\text{post}}\mathbf{A_1}^{-1T} \tag{A.31}$$

$$= (\boldsymbol{\Sigma}_{\text{post}}\boldsymbol{\Sigma}_{\text{prior}}^{-1})^{-1}\boldsymbol{\Sigma}_{\text{post}}(\boldsymbol{\Sigma}_{\text{prior}}^{-1}\boldsymbol{\Sigma}_{\text{post}})^{-1} \tag{A.32}$$

$$= \boldsymbol{\Sigma}_{\text{prior}}\boldsymbol{\Sigma}_{\text{post}}^{-1}\boldsymbol{\Sigma}_{\text{post}}\boldsymbol{\Sigma}_{\text{post}}^{-1}\boldsymbol{\Sigma}_{\text{prior}} \tag{A.33}$$

$$= \boldsymbol{\Sigma}_{\text{prior}}\boldsymbol{\Sigma}_{\text{post}}^{-1}\boldsymbol{\Sigma}_{\text{prior}} \tag{A.34}$$

$$= \boldsymbol{\Sigma}_{\text{prior}}(\boldsymbol{\Sigma}_{\text{prior}}^{-1} + \boldsymbol{\Sigma}_{\text{like}}^{-1})\boldsymbol{\Sigma}_{\text{prior}} \tag{A.35}$$

$$= \boldsymbol{\Sigma}_{\text{prior}} + \boldsymbol{\Sigma}_{\text{prior}}\boldsymbol{\Sigma}_{\text{like}}^{-1}\boldsymbol{\Sigma}_{\text{prior}}. \tag{A.36}$$

This proves that **the covariances of the two distributions are aligned when $\boldsymbol{\Sigma}_{anc} = \boldsymbol{\Sigma}_{\text{prior}} + \boldsymbol{\Sigma}_{\text{prior}}\boldsymbol{\Sigma}_{\text{like}}^{-1}\boldsymbol{\Sigma}_{\text{prior}}$.**

$\square$

Corollary 1.1. *Following from theorem 1 (and under the same assumptions), set $\boldsymbol{\mu}_{anc} := \boldsymbol{\mu}_{prior}$ and $\boldsymbol{\Sigma}_{anc} := \boldsymbol{\Sigma}_{prior}$. The RMS approximate posterior is $P(\boldsymbol{f}_{MAP}(\boldsymbol{\theta}_{anc})) = \mathcal{N}(\boldsymbol{\mu}_{post}, \boldsymbol{\Sigma}_{post}\boldsymbol{\Sigma}_{prior}^{-1}\boldsymbol{\Sigma}_{post})$.*

Proof. Independent of the choice of anchor distribution covariance Σ_{anc}, theorem 1 demonstrated that the resulting posterior, $P(\boldsymbol{f}_{\text{MAP}}(\boldsymbol{\theta}_{anc}))$ is normally distributed, with mean equal to that of the true posterior $\boldsymbol{\mu}_{\text{post}}$.

To discover the covariance of the resulting distribution, $\mathbb{V}\text{ar}[\boldsymbol{f}_{\text{MAP}}(\boldsymbol{\theta}_{anc})]$, we take eq. A.29 and simply set, $\boldsymbol{\Sigma}_{anc} \coloneqq \boldsymbol{\Sigma}_{\text{prior}}$.

$$\begin{aligned}
\mathbb{V}\text{ar}[\boldsymbol{f}_{\text{MAP}}(\boldsymbol{\theta}_{anc})] &= \mathbf{A}_1 \boldsymbol{\Sigma}_{anc} \mathbf{A}_1^T && \text{(A.37)}\\
&= \mathbf{A}_1 \boldsymbol{\Sigma}_{\text{prior}} \mathbf{A}_1^T && \text{(A.38)}\\
&= \boldsymbol{\Sigma}_{\text{post}} \boldsymbol{\Sigma}_{\text{prior}}^{-1} \boldsymbol{\Sigma}_{\text{prior}} \boldsymbol{\Sigma}_{\text{prior}}^{-1} \boldsymbol{\Sigma}_{\text{post}} && \text{(A.39)}\\
&= \boldsymbol{\Sigma}_{\text{post}} \boldsymbol{\Sigma}_{\text{prior}}^{-1} \boldsymbol{\Sigma}_{\text{post}} && \text{(A.40)}
\end{aligned}$$

□

Lemma 1.1. *Following from corollary 1.1 (and under the same assumptions), when $\boldsymbol{\mu}_{anc} \coloneqq \boldsymbol{\mu}_{prior}$, $\Sigma_{anc} \coloneqq \Sigma_{prior}$, the RMS approximate posterior will in general underestimate the marginal variance compared to the true posterior, $\mathbb{V}ar[\boldsymbol{f}_{MAP}(\theta_{anc})] < \mathbb{V}ar[\theta|\mathcal{D}]$.*

Proof. We consider the marginal posterior of a single parameter, $\theta \coloneqq \boldsymbol{\theta}_i$, again assuming multivariate normal prior and parameter likelihood. First consider the following rearrangement of eq. A.40, beginning by noting, $\boldsymbol{\Sigma}_{\text{prior}}^{-1} = \boldsymbol{\Sigma}_{\text{post}}^{-1} - \boldsymbol{\Sigma}_{\text{like}}^{-1}$.

$$\begin{aligned}
\boldsymbol{\Sigma}_{\text{post}} \boldsymbol{\Sigma}_{\text{prior}}^{-1} \boldsymbol{\Sigma}_{\text{post}} &= \boldsymbol{\Sigma}_{\text{post}} (\boldsymbol{\Sigma}_{\text{post}}^{-1} - \boldsymbol{\Sigma}_{\text{like}}^{-1}) \boldsymbol{\Sigma}_{\text{post}} && \text{(A.41)}\\
&= (\mathbb{I} - \boldsymbol{\Sigma}_{\text{post}} \boldsymbol{\Sigma}_{\text{like}}^{-1}) \boldsymbol{\Sigma}_{\text{post}} && \text{(A.42)}\\
&= \boldsymbol{\Sigma}_{\text{post}} - \boldsymbol{\Sigma}_{\text{post}} \boldsymbol{\Sigma}_{\text{like}}^{-1} \boldsymbol{\Sigma}_{\text{post}} && \text{(A.43)}
\end{aligned}$$

To show that RMS generally underestimates the marginal variance, it must hold that diagonal elements of the true posterior covariance matrix are greater than or equal to the same diagonal element of the RMS posterior.

$$\begin{aligned}
\mathbb{V}\text{ar}[\boldsymbol{f}_{\text{MAP}}(\theta_{anc})] &< \mathbb{V}\text{ar}[\theta|\mathcal{D}] && \text{(A.44)}\\
\text{diag}(\boldsymbol{\Sigma}_{\text{post}} \boldsymbol{\Sigma}_{\text{prior}}^{-1} \boldsymbol{\Sigma}_{\text{post}})_i &< \text{diag}(\boldsymbol{\Sigma}_{\text{post}})_i && \text{(A.45)}
\end{aligned}$$

substituting in the diagonal of the rearrangement in eq. A.43,

$$\text{diag}(\boldsymbol{\Sigma}_{\text{post}})_i - \text{diag}(\boldsymbol{\Sigma}_{\text{post}} \boldsymbol{\Sigma}_{\text{like}}^{-1} \boldsymbol{\Sigma}_{\text{post}})_i < \text{diag}(\boldsymbol{\Sigma}_{\text{post}})_i \qquad \text{(A.46)}$$

We know that ABA^T is positive definite if A, B are positive definite, and also that the inverse of a positive definite matrix is positive definite (§9.6.4, §9.6.10, The Matrix Cookbook, 2008). The diagonal of a positive definite matrix is positive. Hence, $\text{diag}(\mathbf{\Sigma}_{\text{post}}\mathbf{\Sigma}_{\text{like}}^{-1}\mathbf{\Sigma}_{\text{post}})_i > 0$, and we have shown that $\mathbb{V}\text{ar}[\boldsymbol{f}_{\text{MAP}}(\theta_{anc})] < \mathbb{V}\text{ar}[\theta|\mathcal{D}]$.

□

Lemma 1.2. *This lemma follows from corollary 1.1. Again parameter likelihood and prior are assumed normally distributed. The prior is additionally assumed isotropic. When $\boldsymbol{\mu}_{anc} := \boldsymbol{\mu}_{prior}$, $\mathbf{\Sigma}_{anc} := \mathbf{\Sigma}_{prior}$ the eigenvectors (or 'orientation') of the RMS approximate posterior equal those of the true posterior.*

Proof. From eq. A.40, $\mathbb{V}\text{ar}[\boldsymbol{f}_{\text{MAP}}(\boldsymbol{\theta}_{anc})] = \mathbf{\Sigma}_{\text{post}}\mathbf{\Sigma}_{\text{prior}}^{-1}\mathbf{\Sigma}_{\text{post}}$. If the prior is isotropic, $\mathbf{\Sigma}_{\text{prior}} = \sigma_{\text{prior}}^2\mathbb{I}$, then, $\mathbb{V}\text{ar}[\boldsymbol{f}_{\text{MAP}}(\boldsymbol{\theta}_{anc})] = 1/\sigma_{\text{prior}}^2\mathbf{\Sigma}_{\text{post}}^2$. Hence the prior only scales the eigenvalues, and doesn't affect the eigenvectors. (Note that this won't be the case for non-isotropic $\mathbf{\Sigma}_{\text{prior}}$.)

Consider some matrix A and a specific eigenvalue λ_i and eigenvector $\mathbf{v}_i$ so that, $A\mathbf{v}_i = \lambda_i\mathbf{v}_i$. It then follows that if A is squared, $A^2\mathbf{v}_i = A(A\mathbf{v}_i) = \lambda_i A\mathbf{v}_i = \lambda_i^2\mathbf{v}_i$. Hence eigenvalues are squared but eigenvectors are unaffected. This applies to the transformation $\mathbf{\Sigma}_{\text{post}}^2$.

Hence both the square and the multiplication of prior covariance, $1/\sigma_{\text{prior}}^2\mathbf{\Sigma}_{\text{post}}^2$, do not modify the original eigenvectors of $\mathbf{\Sigma}_{\text{post}}$ and its orientation is unaffected.

□

Theorem 2. *For a two parameter model with normally distributed parameter likelihood and isotropic prior, the RMS approximate posterior will in general overestimate the magnitude of the true posterior parameter correlation coefficient, $|\rho|$. However, if $|\rho| = 1$, then it will recover it precisely. We set $\boldsymbol{\mu}_{anc} := \boldsymbol{\mu}_{prior}, \mathbf{\Sigma}_{anc} := \mathbf{\Sigma}_{prior}$.*

Proof. From corollary 1.1, we have that the RMS approximate posterior is given by $\mathbf{\Sigma}_{\text{post}}\mathbf{\Sigma}_{\text{prior}}^{-1}\mathbf{\Sigma}_{\text{post}}$. Let $\mathbf{\Sigma}_{\text{prior}} := \sigma_{\text{prior}}^2\mathbb{I}$, the RMS approximate posterior is then given by $1/\sigma_{\text{prior}}^2\mathbf{\Sigma}_{\text{post}}^2$. Denote the true posterior covariance as the following 2×2 matrix.

$$\mathbf{\Sigma}_{\text{post}} = \begin{bmatrix} a & b \\ b & c \end{bmatrix} \tag{A.47}$$

For general covariance matrices, the correlation coefficient, ρ, can be found by solving, $b = \rho\sqrt{ac}$.

Our RMS approximate posterior is given as follows.

$$1/\sigma^2_{\text{prior}}\mathbf{\Sigma}^2_{\text{post}} = 1/\sigma^2_{\text{prior}} \begin{bmatrix} a & b \\ b & c \end{bmatrix} \begin{bmatrix} a & b \\ b & c \end{bmatrix} \tag{A.48}$$

$$= 1/\sigma^2_{\text{prior}} \begin{bmatrix} a^2+b^2 & ab+bc \\ ab+bc & b^2+c^2 \end{bmatrix} \tag{A.49}$$

The correlation coefficient here, denoted ρ_{RMS}, is found by solving, $ab+bc = \rho_{\text{RMS}}\sqrt{(a^2+b^2)(b^2+c^2)}$.

To prove the correlation coefficient is generally overestimated, we must show that $\rho^2_{\text{RMS}} > \rho^2$ when $\rho^2 < 1$.

$$\rho^2_{\text{RMS}} > \rho^2 \tag{A.50}$$

$$\frac{(ab+bc)^2}{(a^2+b^2)(b^2+c^2)} > \frac{b^2}{ac} \tag{A.51}$$

$$(ab+bc)^2 ac > b^2(a^2+b^2)(b^2+c^2) \tag{A.52}$$

$$(a+c)^2 ac > (a^2+b^2)(b^2+c^2) \tag{A.53}$$

$$a^3c + ac^3 + 2a^2c^2 > a^2b^2 + a^2c^2 + b^4 + b^2c^2 \tag{A.54}$$

$$a^3c + ac^3 + a^2c^2 > a^2b^2 + b^4 + b^2c^2 \tag{A.55}$$

We now note that from the set up of the proof, $b^2 = \rho^2 ac$. Since $\rho^2 < 1$, we have, $b^2/ac < 1 \implies b^2 < ac$. We can use this to provide an upper bound on the right hand side of eq. A.55.

$$a^2b^2 + b^4 + b^2c^2 < a^2(ac) + (ac)^2 + (ac)c^2 \tag{A.56}$$

$$< a^3c + a^2c^2 + ac^3 \tag{A.57}$$

Coincidentally, this is precisely the inequality in eq. A.55, that we were proving.

Alternatively, if $|\rho| = 1 \implies b^2 = ac$, and $\rho^2_{\text{RMS}} = \rho^2$. $\square$

Definition 2. Extrapolation Parameters

We define extrapolation parameters as model parameters which have no effect on the data likelihood of a training dataset, but which nevertheless could influence model predictions made on a new data point.

Illustrative Example

Consider a fully-connected NN trained on the MNIST digit dataset. Further consider a preprocessing such that the pixel values by default are set to 0, and where they contain part of the digit take values $(0, 1]$.

Certain pixels may be zero across the entire training dataset, such as those in the corners of the image. First layer weights connected to such pixels will never receive input across the whole training dataset. Hence, the values of these parameter weights have no effect on the data likelihood. However, the weights would still influence predictions for some test image containing values for these pixels. Hence, these are named extrapolation parameters, since they influence extrapolation properties of the model.

The top row of figure 4.5 empirically shows examples of flat likelihoods for precisely these types of weights on MNIST.

Theorem 3. *For extrapolation parameters (definition 2) of a model, setting $\boldsymbol{\mu}_{anc} := \boldsymbol{\mu}_{prior}$, $\Sigma_{anc} := \Sigma_{prior}$, means the marginal RMS approximate posterior equals that of the marginal true posterior. This holds for any distributional form of parameter likelihood.*

Proof. Extrapolation parameters (definition 2) do not have any effect on the data likelihood, therefore their parameter likelihoods are flat. This means that their marginal posterior equals their marginal prior.

This results in a posterior covariance matrix structure as follows, where parameter i is an extrapolation parameter (here shown in the first row for convenience), so has marginal variance equal to the prior variance and is uncorrelated with all other parameters.

$$\boldsymbol{\Sigma}_{\text{post}} = \begin{bmatrix} \sigma^2_{prior,i} & 0 & \dots & 0 \\ 0 & a_{22} & \dots & a_{2D} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & a_{D2} & \dots & a_{DD} \end{bmatrix}$$

From corollary 1.1, $\mathbb{V}\text{ar}[\boldsymbol{f}_{\text{MAP}}(\boldsymbol{\theta}_{anc})] = \boldsymbol{\Sigma}_{\text{post}}\boldsymbol{\Sigma}_{\text{prior}}^{-1}\boldsymbol{\Sigma}_{\text{post}}$.

$$\mathbb{V}\text{ar}[\boldsymbol{f}_{\text{MAP}}(\boldsymbol{\theta}_{anc})] = \begin{bmatrix} \sigma^2_{prior,i} & 0 & \dots & 0 \\ 0 & a_{22} & \dots & a_{2D} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & a_{D2} & \dots & a_{DD} \end{bmatrix} \begin{bmatrix} 1/\sigma^2_{prior,i} & 0 & \dots & 0 \\ 0 & b_{22} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & b_{DD} \end{bmatrix} \begin{bmatrix} \sigma^2_{prior,i} & 0 & \dots & 0 \\ 0 & a_{22} & \dots & a_{2D} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & a_{D2} & \dots & a_{DD} \end{bmatrix}$$
$$= \begin{bmatrix} \sigma^2_{prior,i} & 0 & \dots & 0 \\ 0 & c_{22} & \dots & c_{2D} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & c_{D2} & \dots & c_{DD} \end{bmatrix}$$

This shows that the marginal variance of the RMS approximate posterior equals that of the true posterior, $\mathbb{V}\text{ar}[\boldsymbol{f}_{\text{MAP}}(\boldsymbol{\theta}_{anc})]_{i,i} = [\boldsymbol{\Sigma}_{\text{post}}]_{i,i}$, for extrapolation parameters. Note that the values of the rest of the covariance matrices (a's, b's, c's) are irrelevant since these have no effect on the marginals of interest.

This proof did not assume any specific distributional form of parameter likelihood, only that it is flat for these extrapolation parameters.

□

Theorem 4. *Set $\boldsymbol{\mu}_{anc} := \boldsymbol{\mu}_{prior}, \boldsymbol{\Sigma}_{anc} := \boldsymbol{\Sigma}_{prior}$. The RMS approximate posterior will exactly equal the true posterior, $\boldsymbol{\Sigma}_{post}$, when all eigenvalues of a scaled version of $\boldsymbol{\Sigma}_{post}$ (scaled such that the prior equals the identity matrix) are equal to either* 0 *or* 1. *This corresponds to posteriors that are a mixture of perfectly correlated and perfectly uncorrelated parameters.*

Proof. We will initially consider a scaled version of the parameter space. This conveniently allows standard results for idempotent matrices to apply to the posterior covariance. A reverse scaling is subsequently applied to show that results hold for the original unscaled version. Finally, we articulate arguments allowing relaxation of the distributional assumptions.

Corollary 1.1 showed that the RMS approximate posterior is normally distributed and centered at the true posterior mean, but with modified variance, $\mathcal{N}(\boldsymbol{\mu}_{\text{post}}, \boldsymbol{\Sigma}_{\text{post}}\boldsymbol{\Sigma}_{\text{prior}}^{-1}\boldsymbol{\Sigma}_{\text{post}})$. This proof requires specifying conditions that allow $\boldsymbol{\Sigma}_{\text{post}}\boldsymbol{\Sigma}_{\text{prior}}^{-1}\boldsymbol{\Sigma}_{\text{post}} = \boldsymbol{\Sigma}_{\text{post}}$ to hold.

One solution is given by $\boldsymbol{\Sigma}_{\text{post}} = \boldsymbol{\Sigma}_{\text{prior}}$, and is a trivial extension of theorem 3. Here we consider alternative solutions.

The two inputs into the inference process are the likelihood and prior covariances. Consider a scaling $\boldsymbol{\Sigma}'_{\text{like}} := \boldsymbol{\Sigma}_{\text{prior}}^{-1/2}\boldsymbol{\Sigma}_{\text{like}}\boldsymbol{\Sigma}_{\text{prior}}^{-1/2}$ and $\boldsymbol{\Sigma}'_{\text{prior}} := \boldsymbol{\Sigma}_{\text{prior}}^{-1/2}\boldsymbol{\Sigma}_{\text{prior}}\boldsymbol{\Sigma}_{\text{prior}}^{-1/2} = \mathbb{I}$. The posterior for this scaled version will be denoted by $\boldsymbol{\Sigma}'_{\text{post}}$, and is given as follows.

$$\boldsymbol{\Sigma}'_{\text{post}} = (\boldsymbol{\Sigma}'^{-1}_{\text{like}} + \boldsymbol{\Sigma}'^{-1}_{\text{prior}})^{-1} \tag{A.58}$$

$$= (\boldsymbol{\Sigma}_{\text{prior}}^{1/2}\boldsymbol{\Sigma}_{\text{like}}^{-1}\boldsymbol{\Sigma}_{\text{prior}}^{1/2} + \boldsymbol{\Sigma}_{\text{prior}}^{1/2}\boldsymbol{\Sigma}_{\text{prior}}^{-1}\boldsymbol{\Sigma}_{\text{prior}}^{1/2})^{-1} \tag{A.59}$$

$$= \boldsymbol{\Sigma}_{\text{prior}}^{-1/2}(\boldsymbol{\Sigma}_{\text{like}}^{-1} + \boldsymbol{\Sigma}_{\text{prior}}^{-1})^{-1}\boldsymbol{\Sigma}_{\text{prior}}^{-1/2} \tag{A.60}$$

$$= \boldsymbol{\Sigma}_{\text{prior}}^{-1/2}\boldsymbol{\Sigma}_{\text{post}}\boldsymbol{\Sigma}_{\text{prior}}^{-1/2} \tag{A.61}$$

Hence, unsurprisingly the same scaling applies to the posterior covariance, $\boldsymbol{\Sigma}'_{\text{post}} = \boldsymbol{\Sigma}_{\text{prior}}^{-1/2}\boldsymbol{\Sigma}_{\text{post}}\boldsymbol{\Sigma}_{\text{prior}}^{-1/2}$.

We now consider conditions under which $\boldsymbol{\Sigma}'_{\text{post}}\boldsymbol{\Sigma}'^{-1}_{\text{prior}}\boldsymbol{\Sigma}'_{\text{post}} = \boldsymbol{\Sigma}'_{\text{post}}$ holds. From our choice of rescaling, we have that $\boldsymbol{\Sigma}'^{-1}_{\text{prior}} = \mathbb{I}$. So we require that $\boldsymbol{\Sigma}'^{2}_{\text{post}} = \boldsymbol{\Sigma}'_{\text{post}}$.

This conveniently allows use of results for idempotent matrices - defined as a square matrix, A, for which $A^2 = A$. Aside from the case when $A = \mathbb{I}$ (which corresponds to $\boldsymbol{\Sigma}_{\text{post}} = \boldsymbol{\Sigma}_{\text{prior}}$), a matrix is idempotent if and only if it is singular and all eigenvalues are 0 or 1.

In order that, $\boldsymbol{\Sigma}'_{\text{post}}\boldsymbol{\Sigma}'^{-1}_{\text{prior}}\boldsymbol{\Sigma}'_{\text{post}} = \boldsymbol{\Sigma}'_{\text{post}}$, it is therefore sufficient that our scaled posterior, $\boldsymbol{\Sigma}'_{\text{post}}$, is singular with all eigenvalues 0 or 1. Any possible permutation is allowed.

Naturally, applying a reverse scaling recovers the original parameter space, $\boldsymbol{\Sigma}_{\text{post}} = \boldsymbol{\Sigma}_{\text{prior}}^{1/2}\boldsymbol{\Sigma}'_{\text{post}}\boldsymbol{\Sigma}_{\text{prior}}^{1/2}$.

Remark. *To summarise, we have shown that provided the RMS approximate posterior equals the true posterior in the scaled space, it will also be equal in the original unscaled space. In order for this equality to hold, eigenvalues must be* 0 *or* 1 *in the scaled space.*

See section A.3 for numerical examples in a three parameter model when this condition holds.

$\square$

A.2 MAP solution and regularisation interpretation

For completeness, we write out the MAP solution for the case of normally distributed prior, and data likelihoods often used in regression and classification. From this derives our interpretation of the regularisation matrix, $\boldsymbol{\Gamma}$.

$$\begin{aligned}\boldsymbol{\theta}_{MAP} &= \text{argmax}_{\boldsymbol{\theta}} P(\boldsymbol{\theta}|\mathcal{D}) \\ &= \text{argmax}_{\boldsymbol{\theta}} P_{\mathcal{D}}(\mathcal{D}|\boldsymbol{\theta})P(\boldsymbol{\theta}) \\ &= \text{argmax}_{\boldsymbol{\theta}} \log(P_{\mathcal{D}}(\mathcal{D}|\boldsymbol{\theta})) + \log(P(\boldsymbol{\theta}))\end{aligned}$$

If prior is normally distributed, $P(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$,

$$\begin{aligned}&= \text{argmax}_{\boldsymbol{\theta}} \log(P_{\mathcal{D}}(\mathcal{D}|\boldsymbol{\theta})) - \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{\theta} - \boldsymbol{\mu}) + \text{const.} \\ &= \text{argmax}_{\boldsymbol{\theta}} \log(P_{\mathcal{D}}(\mathcal{D}|\boldsymbol{\theta})) - \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{\theta} - \boldsymbol{\mu}).\end{aligned}$$

Typically in BNNs the prior covariance is chosen as diagonal. This is sometimes set as isotropic, $\boldsymbol{\Sigma} = \lambda\mathbb{I}$, but here we will keep it in matrix form (but assuming it is diagonal) so that different prior variances can be assigned to different layer weights.

$$= \text{argmax}_{\boldsymbol{\theta}} \log(P_{\mathcal{D}}(\mathcal{D}|\boldsymbol{\theta})) - \frac{1}{2}||\boldsymbol{\Sigma}^{-1/2} \cdot (\boldsymbol{\theta} - \boldsymbol{\mu})||_2^2$$

In the case that the prior mean is zero $\boldsymbol{\mu} = \mathbf{0}$,

$$= \text{argmax}_{\boldsymbol{\theta}} \log(P_{\mathcal{D}}(\mathcal{D}|\boldsymbol{\theta})) - \frac{1}{2}||\boldsymbol{\Sigma}^{-1/2} \cdot \boldsymbol{\theta}||_2^2.$$

One is free to choose any suitable expression for $\log(P_{\mathcal{D}}(\mathcal{D}|\boldsymbol{\theta}))$. Next we describe the resulting forms for common choices of log likelihood in regression and classification tasks.

Regression

For regression, a common choice is that the NN predicts the mean of the function, $\hat{\mathbf{y}}$, and there is additive noise on the true targets $\mathbf{y}$, $P_{\mathcal{D}}(\mathcal{D}|\boldsymbol{\theta}) = \mathcal{N}(\mathbf{y}|\hat{\mathbf{y}}, \sigma_\epsilon^2)$

$$\begin{aligned}\boldsymbol{\theta}_{MAP} &= \text{argmax}_{\boldsymbol{\theta}} - \frac{1}{2\sigma_\epsilon^2}||\hat{\mathbf{y}} - \mathbf{y}||_2^2 + \text{const.} - \frac{1}{2}||\boldsymbol{\Sigma}^{-1/2} \cdot \boldsymbol{\theta}||_2^2 \\ &= \text{argmax}_{\boldsymbol{\theta}} - \frac{1}{2\sigma_\epsilon^2}||\hat{\mathbf{y}} - \mathbf{y}||_2^2 - \frac{1}{2}||\boldsymbol{\Sigma}^{-1/2} \cdot \boldsymbol{\theta}||_2^2\end{aligned}$$

Generally the mean squared error is minimised,

$$= \text{argmin}_{\boldsymbol{\theta}} \frac{1}{N} ||\hat{\mathbf{y}} - \mathbf{y}||_2^2 + \frac{1}{N} ||\sigma_\epsilon \mathbf{\Sigma}^{-1/2} \cdot \boldsymbol{\theta}||_2^2$$

More compactly, we can define $\mathbf{\Gamma} := \sigma_\epsilon^2 \mathbf{\Sigma}^{-1}$, as a diagonal matrix with, $\text{diag}(\mathbf{\Gamma})_i = \sigma_\epsilon^2 / \sigma_{prior,i}^2$,

$$= \text{argmin}_{\boldsymbol{\theta}} \frac{1}{N} ||\hat{\mathbf{y}} - \mathbf{y}||_2^2 + \frac{1}{N} ||\mathbf{\Gamma}^{1/2} \cdot \boldsymbol{\theta}||_2^2$$

Classification

The data likelihood is commonly chosen as a multinomial distribution, $P_{\mathcal{D}}(\mathcal{D}|\boldsymbol{\theta}) \propto \prod_{n=1}^{N} \prod_{c=1}^{C} \hat{y}_{n,c}^{y_{n,c}}$, for C classes, and N data points, where $\hat{y} \in [0,1]$ denotes predicted probability, and $y_{n,c} \in \{0,1\}$ the true targets.

$$\boldsymbol{\theta}_{MAP} = \text{argmax}_{\boldsymbol{\theta}} \sum_{n=1}^{N} \sum_{c=1}^{C} y_{n,c} \log(\hat{y}_{n,c}) - \frac{1}{2} ||\mathbf{\Sigma}^{-1/2} \cdot \boldsymbol{\theta}||_2^2$$

Cross entropy is typically minimised,

$$= \text{argmin}_{\boldsymbol{\theta}} - \sum_{n=1}^{N} \sum_{c=1}^{C} y_{n,c} \log(\hat{y}_{n,c}) + \frac{1}{2} ||\mathbf{\Sigma}^{-1/2} \cdot \boldsymbol{\theta}||_2^2$$

Here we can simply define $\mathbf{\Gamma} := \frac{1}{2} \mathbf{\Sigma}^{-1}$, with $\text{diag}(\mathbf{\Gamma})_i = 1/2\sigma_{prior,i}^2$.

A.3 Numerical Examples

Numerical Examples of Proofs

In this section we print covariance matrices that illustrate theoretical results numerically.

General case: Example of lemma 1.1, 1.2, theorem 2

For general $\mathbf{\Sigma}_{\text{post}}$, RMS will return a posterior with underestimated marginal variancesa and overestimated correlations. Since the prior is isotropic, the orientation (eigenvectors) of the RMS approximate posterior will be unchanged. Here a three parameter is shown. Note $\mathbf{\Sigma}_{\text{post}}\mathbf{\Sigma}_{\text{prior}}^{-1}\mathbf{\Sigma}_{\text{post}}$ represents the RMS approximate posterior.

$$\mathbf{\Sigma}_{\text{prior}} = \begin{bmatrix} 2.0 & 0.0 & 0.0 \\ 0.0 & 2.0 & 0.0 \\ 0.0 & 0.0 & 2.0 \end{bmatrix} \qquad \mathbf{\Sigma}_{\text{like}} = \begin{bmatrix} 2.0 & 0.707 & 0.283 \\ 0.707 & 1.0 & 0.4 \\ 0.283 & 0.4 & 1.0 \end{bmatrix}$$

$$\mathbf{\Sigma}_{\text{post}} = \begin{bmatrix} 0.953 & 0.238 & 0.067 \\ 0.238 & 0.589 & 0.166 \\ 0.067 & 0.166 & 0.638 \end{bmatrix} \qquad \mathbf{\Sigma}_{\text{post}}\mathbf{\Sigma}_{\text{prior}}^{-1}\mathbf{\Sigma}_{\text{post}} = \begin{bmatrix} 0.485 & 0.189 & 0.073 \\ 0.189 & 0.215 & 0.11 \\ 0.073 & 0.11 & 0.22 \end{bmatrix}$$

Note, $\mathbf{\Sigma}_{\text{post}_{i,i}} > \mathbf{\Sigma}_{\text{post}}\mathbf{\Sigma}_{\text{prior}}^{-1}\mathbf{\Sigma}_{\text{post}_{i,i}}, \forall i$.

$$\text{Correlation}(\mathbf{\Sigma}_{\text{post}}) = \begin{bmatrix} 1.0 & 0.317 & 0.086 \\ 0.317 & 1.0 & 0.27 \\ 0.086 & 0.27 & 1.0 \end{bmatrix} \qquad \text{Correlation}(\mathbf{\Sigma}_{\text{post}}\mathbf{\Sigma}_{\text{prior}}^{-1}\mathbf{\Sigma}_{\text{post}}) = \begin{bmatrix} 1.0 & 0.585 & 0.224 \\ 0.585 & 1.0 & 0.504 \\ 0.224 & 0.504 & 1.0 \end{bmatrix}$$

Note, $\text{Correlation}(\mathbf{\Sigma}_{\text{post}})_{i,j} < \text{Correlation}(\mathbf{\Sigma}_{\text{post}}\mathbf{\Sigma}_{\text{prior}}^{-1}\mathbf{\Sigma}_{\text{post}})_{i,j}, \forall i \neq j$.

Eigenvalues and eigenvectors of $\mathbf{\Sigma}_{\text{post}}$,

$$\lambda = 1.1101, \mathbf{v} = \begin{bmatrix} -0.8352 & -0.471 & -0.284 \end{bmatrix}$$
$$\lambda = 0.6667, \mathbf{v} = \begin{bmatrix} -0.465 & 0.3288 & 0.822 \end{bmatrix}$$
$$\lambda = 0.4032, \mathbf{v} = \begin{bmatrix} 0.2938 & -0.8186 & 0.4936 \end{bmatrix}$$

Eigenvalues and eigenvectors of $\mathbf{\Sigma}_{\text{post}}\mathbf{\Sigma}_{\text{prior}}^{-1}\mathbf{\Sigma}_{\text{post}}$,

$$\lambda = 0.6162, \mathbf{v} = \begin{bmatrix}-0.8352 & -0.471 & -0.284\end{bmatrix}$$
$$\lambda = 0.2222, \mathbf{v} = \begin{bmatrix}-0.465 & 0.3288 & 0.822\end{bmatrix}$$
$$\lambda = 0.0813, \mathbf{v} = \begin{bmatrix}0.2938 & -0.8186 & 0.4936\end{bmatrix}$$

Special case: Examples of theorem 3, 4

We again print out covariance matrices for a three parameter model. Firstly we provide an example where two parameters are perfectly correlated, and one has no effect on the likelihood. We print unscaled and scaled versions. Note that all eigenvalues of the scaled posterior, $\mathbf{\Sigma}'_{\text{post}}$, are either 0 or 1.

$$\mathbf{\Sigma}_{\text{post}} = \begin{bmatrix}1.0 & 1.0 & 0.0\\ 1.0 & 1.0 & 0.0\\ 0.0 & 0.0 & 2.0\end{bmatrix} \quad \mathbf{\Sigma}_{\text{prior}} = \begin{bmatrix}2.0 & 0.0 & 0.0\\ 0.0 & 2.0 & 0.0\\ 0.0 & 0.0 & 2.0\end{bmatrix} \quad \mathbf{\Sigma}_{\text{post}}\mathbf{\Sigma}_{\text{prior}}^{-1}\mathbf{\Sigma}_{\text{post}} = \begin{bmatrix}1.0 & 1.0 & 0.0\\ 1.0 & 1.0 & 0.0\\ 0.0 & 0.0 & 2.0\end{bmatrix}$$

$$\mathbf{\Sigma}'_{\text{post}} = \begin{bmatrix}0.5 & 0.5 & 0.0\\ 0.5 & 0.5 & 0.0\\ 0.0 & 0.0 & 1.0\end{bmatrix} \quad \mathbf{\Sigma}'_{\text{prior}} = \begin{bmatrix}1.0 & 0.0 & 0.0\\ 0.0 & 1.0 & 0.0\\ 0.0 & 0.0 & 1.0\end{bmatrix} \quad \mathbf{\Sigma}'_{\text{post}}\mathbf{\Sigma}'^{-1}_{\text{prior}}\mathbf{\Sigma}'_{\text{post}} = \begin{bmatrix}0.5 & 0.5 & 0.0\\ 0.5 & 0.5 & 0.0\\ 0.0 & 0.0 & 1.0\end{bmatrix}$$

Eigenvalues and eigenvectors of $\mathbf{\Sigma}_{\text{post}} = \mathbf{\Sigma}_{\text{post}}\mathbf{\Sigma}_{\text{prior}}^{-1}\mathbf{\Sigma}_{\text{post}}$,

$$\lambda = 2.0, \mathbf{v} = \begin{bmatrix}0.7071 & 0.7071 & 0.\end{bmatrix}$$
$$\lambda = 0.0, \mathbf{v} = \begin{bmatrix}-0.7071 & 0.7071 & 0.\end{bmatrix}$$
$$\lambda = 2.0, \mathbf{v} = \begin{bmatrix}0. & 0. & 1.\end{bmatrix}$$

Eigenvalues and eigenvectors of $\mathbf{\Sigma}'_{\text{post}} = \mathbf{\Sigma}'_{\text{post}}\mathbf{\Sigma}'^{-1}_{\text{prior}}\mathbf{\Sigma}'_{\text{post}}$,

$$\lambda = 1.0, \mathbf{v} = \begin{bmatrix} 0.707 & 0.707 & 0. \end{bmatrix}$$
$$\lambda = 0.0, \mathbf{v} = \begin{bmatrix} -0.707 & 0.707 & 0. \end{bmatrix}$$
$$\lambda = 1.0, \mathbf{v} = \begin{bmatrix} 0. & 0. & 1. \end{bmatrix}$$

Following is the same set up as the previous case, but now all parameters are perfectly correlated. Eigenvalues of the scaled posterior, $\mathbf{\Sigma}'_{\text{post}}$, are again either 0 or 1.

$$\mathbf{\Sigma}_{\text{post}} = \begin{bmatrix} 0.667 & 0.667 & 0.667 \\ 0.667 & 0.667 & 0.667 \\ 0.667 & 0.667 & 0.667 \end{bmatrix} \quad \mathbf{\Sigma}_{\text{prior}} = \begin{bmatrix} 2.0 & 0.0 & 0.0 \\ 0.0 & 2.0 & 0.0 \\ 0.0 & 0.0 & 2.0 \end{bmatrix} \quad \mathbf{\Sigma}_{\text{post}}\mathbf{\Sigma}^{-1}_{\text{prior}}\mathbf{\Sigma}_{\text{post}} = \begin{bmatrix} 0.667 & 0.667 & 0.667 \\ 0.667 & 0.667 & 0.667 \\ 0.667 & 0.667 & 0.667 \end{bmatrix}$$

$$\mathbf{\Sigma}'_{\text{post}} = \begin{bmatrix} 0.333 & 0.333 & 0.333 \\ 0.333 & 0.333 & 0.333 \\ 0.333 & 0.333 & 0.333 \end{bmatrix} \quad \mathbf{\Sigma}'_{\text{prior}} = \begin{bmatrix} 1.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 1.0 \end{bmatrix} \quad \mathbf{\Sigma}'_{\text{post}}\mathbf{\Sigma}'^{-1}_{\text{prior}}\mathbf{\Sigma}'_{\text{post}} = \begin{bmatrix} 0.333 & 0.333 & 0.333 \\ 0.333 & 0.333 & 0.333 \\ 0.333 & 0.333 & 0.333 \end{bmatrix}$$

Eigenvalues and eigenvectors of $\mathbf{\Sigma}_{\text{post}} = \mathbf{\Sigma}_{\text{post}}\mathbf{\Sigma}^{-1}_{\text{prior}}\mathbf{\Sigma}_{\text{post}}$,

$$\lambda = 2.0, \mathbf{v} = \begin{bmatrix} -0.5774 & -0.5774 & -0.5774 \end{bmatrix}$$
$$\lambda = 0.0, \mathbf{v} = \begin{bmatrix} -0. & -0.7071 & 0.7071 \end{bmatrix}$$
$$\lambda = 0.0, \mathbf{v} = \begin{bmatrix} -0.6667 & -0.0749 & 0.7416 \end{bmatrix}$$

Eigenvalues and eigenvectors of $\mathbf{\Sigma}'_{\text{post}} = \mathbf{\Sigma}'_{\text{post}}\mathbf{\Sigma}'^{-1}_{\text{prior}}\mathbf{\Sigma}'_{\text{post}}$,

$$\lambda = 1.0, \mathbf{v} = \begin{bmatrix} 0.577 & 0.577 & 0.577 \end{bmatrix}$$
$$\lambda = 0.0, \mathbf{v} = \begin{bmatrix} 0. & -0.707 & 0.707 \end{bmatrix}$$
$$\lambda = 0.0, \mathbf{v} = \begin{bmatrix} -0.521 & -0.284 & 0.805 \end{bmatrix}$$

Now we detail an example of a non-isometric prior.

$$\mathbf{\Sigma}_{\text{post}} = \begin{bmatrix} 1.818 & 0.0 & 1.818 \\ 0.0 & 2.0 & 0.0 \\ 1.818 & 0.0 & 1.818 \end{bmatrix} \quad \mathbf{\Sigma}_{\text{prior}} = \begin{bmatrix} 20.0 & 0.0 & 0.0 \\ 0.0 & 2.0 & 0.0 \\ 0.0 & 0.0 & 2.0 \end{bmatrix} \quad \mathbf{\Sigma}_{\text{post}}\mathbf{\Sigma}_{\text{prior}}^{-1}\mathbf{\Sigma}_{\text{post}} = \begin{bmatrix} 1.818 & 0.0 & 1.818 \\ 0.0 & 2.0 & 0.0 \\ 1.818 & 0.0 & 1.818 \end{bmatrix}$$

$$\mathbf{\Sigma}'_{\text{post}} = \begin{bmatrix} 0.091 & 0.0 & 0.287 \\ 0.0 & 1.0 & 0.0 \\ 0.287 & 0.0 & 0.909 \end{bmatrix} \quad \mathbf{\Sigma}'_{\text{prior}} = \begin{bmatrix} 1.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 1.0 \end{bmatrix} \quad \mathbf{\Sigma}'_{\text{post}}\mathbf{\Sigma}'^{-1}_{\text{prior}}\mathbf{\Sigma}'_{\text{post}} = \begin{bmatrix} 0.091 & 0.0 & 0.287 \\ 0.0 & 1.0 & 0.0 \\ 0.287 & 0.0 & 0.909 \end{bmatrix}$$

Eigenvalues and eigenvectors of $\mathbf{\Sigma}_{\text{post}} = \mathbf{\Sigma}_{\text{post}}\mathbf{\Sigma}_{\text{prior}}^{-1}\mathbf{\Sigma}_{\text{post}}$,

$$\lambda = 3.636, \mathbf{v} = \begin{bmatrix} 0.707 & 0. & 0.707 \end{bmatrix}$$
$$\lambda = 0.0, \mathbf{v} = \begin{bmatrix} -0.707 & 0. & 0.707 \end{bmatrix}$$
$$\lambda = 2.0, \mathbf{v} = \begin{bmatrix} 0. & 1. & 0. \end{bmatrix}$$

Eigenvalues and eigenvectors of $\mathbf{\Sigma}'_{\text{post}} = \mathbf{\Sigma}'_{\text{post}}\mathbf{\Sigma}'^{-1}_{\text{prior}}\mathbf{\Sigma}'_{\text{post}}$,

$$\lambda = 0.0, \mathbf{v} = \begin{bmatrix} -0.953 & 0. & 0.302 \end{bmatrix}$$
$$\lambda = 1.0, \mathbf{v} = \begin{bmatrix} -0.302 & 0. & -0.953 \end{bmatrix}$$
$$\lambda = 1.0, \mathbf{v} = \begin{bmatrix} 0. & 1. & 0. \end{bmatrix}$$

Mixtures of Parameter Types

Here, we provide examples of a five parameter model containing a mixture of perfectly correlated, partially correlated, and extrapolation parameters.

First we consider distinct blocks of perfectly and partially correlated parameters, as well as one extrapolation parameter. In this situation both the perfectly correlated block and the extrapolation parameter posterior are recovered exactly. The RMS approximate posterior of the partially correlated block is biased as per the general case.

$$\mathbf{\Sigma}_{\text{post}} = \begin{bmatrix} 1.0 & 1.0 & 0.0 & 0.0 & 0.0 \\ 1.0 & 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.5 & 0.2 & 0.0 \\ 0.0 & 0.0 & 0.2 & 0.8 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 2.0 \end{bmatrix} \mathbf{\Sigma}_{\text{prior}} = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 2 \end{bmatrix} \mathbf{\Sigma}_{\text{post}}\mathbf{\Sigma}_{\text{prior}}^{-1}\mathbf{\Sigma}_{\text{post}} = \begin{bmatrix} 1.0 & 1.0 & 0.0 & 0.0 & 0.0 \\ 1.0 & 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.145 & 0.13 & 0.0 \\ 0.0 & 0.0 & 0.13 & 0.34 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 2.0 \end{bmatrix}$$

Secondly the perfectly correlated block overlaps with the partially correlated block. In this scenario, a small amount of bias is introduced on the perfectly correlated block, but not in terms of the correlation. The extrapolation parameter is unaffected.

$$\mathbf{\Sigma}_{\text{post}} = \begin{bmatrix} 1.0 & 1.0 & 0.1 & 0.2 & 0.0 \\ 1.0 & 1.0 & 0.1 & 0.2 & 0.0 \\ 0.1 & 0.1 & 0.5 & 0.2 & 0.0 \\ 0.2 & 0.2 & 0.2 & 0.8 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 2.0 \end{bmatrix} \mathbf{\Sigma}_{\text{prior}} = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 2 \end{bmatrix} \mathbf{\Sigma}_{\text{post}}\mathbf{\Sigma}_{\text{prior}}^{-1}\mathbf{\Sigma}_{\text{post}} = \begin{bmatrix} 1.03 & 1.03 & 0.15 & 0.29 & 0.0 \\ 1.03 & 1.03 & 0.15 & 0.29 & 0.0 \\ 0.15 & 0.15 & 0.16 & 0.15 & 0.0 \\ 0.29 & 0.29 & 0.15 & 0.38 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 2.0 \end{bmatrix}$$

A.4 Additional Material

Regression Benchmarking

Tables A.1 & A.2 show all experiments run on the regression benchmarking datasets. The below discussion focuses on NLL results in table A.2.

ERF GP refers to the equivalent GP for an infinite width, single-layer BNN with ERF activations. It was tuned and implemented as for the ReLU GP. We were interested to discover how different activation functions would affect uncertainty estimates. In general the ReLU GP performed better than the ERF GP, with some exceptions, such as for Wine. The target variable for Wine is ordinal, containing five factors, it is therefore understandable that the ReLU GP, which extrapolates linearly, is at a slight disadvantage.

10x 50 NNs refers to an anchored ensemble of ten NNs with 50 hidden nodes. We find that these results fall in between the 5x 50 NNs and the ReLU GP. This agrees with the convergence analysis done in section 4.4.2.

We also implemented an anchored ensemble of five two-layer NNs, 5x 50-50 NNs. Even with minimal hyperparameter tuning (section A.5) we found an extra layer gave a performance boost over the 5x 50 NNs. We expect with more careful tuning this margin would increase.

Single 50 NN refers to a single regularised NN, of one hidden layer with 50 hidden nodes, for which we used a constant value of predictive variance. Although this performs poorly in several cases, e.g. Boston and Yacht, the results are surprisingly close to those achieved by both our method and Deep Ensembles, even surpassing them on the Energy dataset. A method outputting constant predictive variance should not perform well in experiments designed to test uncertainty quantification, and this raises questions over the validity of the benchmarks.

Table A.3 compares anchored ensembles against results reported for other methods.

Table A.1 Variants of our method on benchmark regression datasets, RMSE. Deep ensembles are included for reference, using published results.

			RMSE						
	N	D	ReLU GP	ERF GP	5x 50 NNs	10x 50 NNs	5x 50-50 NNs	Single 50 NN	Deep Ensembles
Boston	506	13	2.86 ± 0.16	2.94 ± 0.18	3.09 ± 0.17	3.09 ± 0.17	3.00 ± 0.18	3.40 ± 0.20	3.28 ± 1.00
Concrete	1,030	8	4.88 ± 0.13	5.21 ± 0.12	4.87 ± 0.11	4.73 ± 0.11	4.75 ± 0.12	5.17 ± 0.13	6.03 ± 0.58
Energy	768	8	0.60 ± 0.02	0.78 ± 0.03	0.35 ± 0.01	0.34 ± 0.01	0.40 ± 0.01	0.40 ± 0.01	2.09 ± 0.29
Kin8nm	8,192	8	0.07 ± 0.00	0.08 ± 0.00	0.07 ± 0.00	0.07 ± 0.00	0.06 ± 0.00	0.07 ± 0.00	0.09 ± 0.00
Naval	11,934	16	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
Power	9,568	4	3.97 ± 0.04	3.94 ± 0.04	4.07 ± 0.04	4.07 ± 0.04	4.03 ± 0.04	4.23 ± 0.04	4.11 ± 0.17
Protein	45,730	9	4.34 ± 0.02	4.23 ± 0.02	4.36 ± 0.02	4.34 ± 0.02	4.23 ± 0.02	4.56 ± 0.02	4.71 ± 0.06
Wine	1,599	11	0.61 ± 0.01	0.60 ± 0.01	0.63 ± 0.01	0.62 ± 0.01	0.62 ± 0.01	0.64 ± 0.01	0.64 ± 0.04
Yacht	308	6	0.60 ± 0.08	1.48 ± 0.15	0.57 ± 0.05	0.54 ± 0.05	0.85 ± 0.08	0.81 ± 0.07	1.58 ± 0.48
Song Year	515,345	90	9.01 ± NA	8.90 ± NA	8.82 ± NA	8.82 ± NA	8.66 ± NA	8.77 ± NA	8.89 ± NA

Table A.2 Variants of our method on benchmark regression datasets, NLL.

		NLL						
	$\hat{\sigma}_\epsilon^2$	ReLU GP	ERF GP	5x 50 NNs	10x 50 NNs	5x 50-50 NNs	Single 50 NN	Deep Ensembles
Boston	0.08	2.45 ± 0.05	2.46 ± 0.05	2.52 ± 0.05	2.50 ± 0.05	2.50 ± 0.07	2.70 ± 0.05	2.41 ± 0.25
Concrete	0.05	2.96 ± 0.02	3.06 ± 0.02	2.97 ± 0.02	2.94 ± 0.02	2.94 ± 0.02	3.08 ± 0.03	3.06 ± 0.18
Energy	1e-7	0.86 ± 0.02	1.06 ± 0.03	0.96 ± 0.13	0.52 ± 0.06	0.61 ± 0.07	0.57 ± 0.03	1.38 ± 0.22
Kin8nm	0.02	-1.22 ± 0.01	-1.17 ± 0.00	-1.09 ± 0.01	-1.16 ± 0.01	-1.25 ± 0.01	-1.17 ± 0.01	-1.20 ± 0.02
Naval	1e-7	-10.05 ± 0.02	-9.66 ± 0.04	-7.17 ± 0.03	-7.29 ± 0.02	-7.08 ± 0.13	-6.58 ± 0.04	-5.63 ± 0.05
Power	0.05	2.80 ± 0.01	2.79 ± 0.01	2.83 ± 0.01	2.83 ± 0.01	2.82 ± 0.01	2.86 ± 0.01	2.79 ± 0.04
Protein	0.5	2.88 ± 0.00	2.86 ± 0.00	2.89 ± 0.01	2.88 ± 0.01	2.86 ± 0.01	2.94 ± 0.00	2.83 ± 0.02
Wine	0.5	0.92 ± 0.01	0.91 ± 0.01	0.95 ± 0.01	0.94 ± 0.01	0.94 ± 0.01	0.97 ± 0.01	0.94 ± 0.12
Yacht	1e-7	0.49 ± 0.07	1.50 ± 0.13	0.37 ± 0.08	0.18 ± 0.03	0.04 ± 0.08	1.50 ± 0.02	1.18 ± 0.21
Song Year	0.7	3.62 ± NA	3.61 ± NA	3.60 ± NA	3.60 ± NA	3.57 ± NA	3.59 ± NA	3.35 ± NA

Fashion MNIST

Table A.4 provides a breakdown of results from the OOD classification test in section 4.4.4 for fashion MNIST. Also included are results for entropy, where high entropy represents high uncertainty. These correlated strongly with the proportion metrics, which was true across all three OOD experiments.

Table A.3 Comparison against inference methods on UCI benchmark regression datasets, log likelihood. Adapted from Mukhoti et al. (2018), also included is the single layer homoskedastic VI results from Tomczak et al. (2018).

	Log Likelihood (*not* negative)								
	Anch. Ens.	Drop conv.	Drop tune	VMG	HS-BNN	PBP-MV	SGHMC tune	SGHMC adap.	VI-IIDV-1HL
Boston	-2.52 ± 0.05	-2.40 ± 0.04	-2.40 ± 0.04	-2.46 ± 0.09	-2.54 ± 0.15	-2.54 ± 0.08	-2.49 ± 0.15	-2.54 ± 0.04	-2.45 ± 0.15
Concrete	-2.97 ± 0.02	-2.97 ± 0.02	-2.93 ± 0.02	-3.01 ± 0.03	-3.09 ± 0.06	-3.04 ± 0.03	-4.17 ± 0.72	-3.38 ± 0.24	-3.05 ± 0.08
Energy	-0.96 ± 0.13	-1.72 ± 0.01	-1.21 ± 0.01	-1.06 ± 0.03	-2.66 ± 0.13	-1.01 ± 0.01	$--$	$--$	-0.72 ± 0.10
Kin8nm	1.09 ± 0.01	0.97 ± 0.00	1.14 ± 0.01	1.10 ± 0.01	1.12 ± 0.03	1.28 ± 0.01	$--$	$--$	1.13 ± 0.04
Naval	7.17 ± 0.03	3.91 ± 0.01	4.45 ± 0.00	2.46 ± 0.00	5.52 ± 0.10	4.85 ± 0.06	$--$	$--$	5.86 ± 0.28
Power	-2.83 ± 0.01	-2.79 ± 0.01	-2.80 ± 0.01	-2.82 ± 0.01	-2.81 ± 0.03	-2.78 ± 0.01	$--$	$--$	-2.82 ± 0.04
Protein	-2.89 ± 0.01	-2.87 ± 0.00	-2.87 ± 0.00	-2.84 ± 0.00	-2.89 ± 0.00	-2.77 ± 0.01	$--$	$--$	-2.93 ± 0.01
Wine	-0.95 ± 0.01	-0.92 ± 0.01	-0.93 ± 0.01	-0.95 ± 0.01	-0.95 ± 0.05	-0.97 ± 0.01	-1.29 ± 0.28	-1.04 ± 0.17	-0.95 ± 0.06
Yacht	-0.37 ± 0.08	-1.38 ± 0.01	-1.25 ± 0.01	-1.30 ± 0.02	-2.33 ± 0.01	-1.64 ± 0.02	-1.75 ± 0.19	-1.10 ± 0.08	-0.86 ± 0.27

Table A.4 Fashion MNIST results: proportion of predictions made with $\geq 90\%$ probability, and entropy of predicted categorical distribution. Also shown is relative advantage (percentage change) for each method compared to anchored ensembles. Averaged over five runs/random seeds, mean $\pm$ 1 standard error. Best result in blue.

		——-Edge Cases——-		—Out-of-distribution—		————Natural Adversarial————			—Pure Adversarial—	
	Train	Sneaker	Trouser	CIFAR	MNIST	Rotate	Flip	Invert	Noise	Sparse
					Proportion $\geq 90\%$ (smaller better)					
reg 1xNN	0.660 ± 0.006	0.739 ± 0.056	0.429 ± 0.047	0.143 ± 0.008	0.160 ± 0.007	0.609 ± 0.007	0.330 ± 0.009	0.349 ± 0.015	0.271 ± 0.007	0.456 ± 0.006
free 5xNN	0.733 ± 0.001	0.781 ± 0.015	0.380 ± 0.030	0.301 ± 0.013	0.104 ± 0.010	0.571 ± 0.011	0.300 ± 0.011	0.222 ± 0.052	0.042 ± 0.005	0.048 ± 0.003
reg 5xNN	0.634 ± 0.002	0.589 ± 0.054	0.269 ± 0.020	0.115 ± 0.004	0.072 ± 0.007	0.556 ± 0.007	0.256 ± 0.012	0.213 ± 0.002	0.112 ± 0.005	0.174 ± 0.005
anc 5xNN	0.631 ± 0.002	0.578 ± 0.049	0.325 ± 0.037	0.065 ± 0.002	0.041 ± 0.002	0.497 ± 0.003	0.215 ± 0.005	0.025 ± 0.010	0.006 ± 0.001	0.006 ± 0.001
					Proportion Relative Advantage					
1xNN Reg. to 5xNN Anch.	-4.4%	-21.8%	-24.2%	-54.5%	-74.4%	-18.4%	-34.8%	-92.8%	-97.8%	-98.7%
5xNN Uncons. to 5xNN Anch.	-13.9%	-26.0%	-14.5%	-78.4%	-60.6%	-13.0%	-28.3%	-88.7%	-85.7%	-87.5%
5xNN Reg. to 5xNN Anch.	-0.5%	-1.9%	20.8%	-43.5%	-43.1%	-10.6%	-16.0%	-88.3%	-94.6%	-96.6%
					Entropy (larger better)					
1xNN Reg.	0.328 ± 0.005	0.253 ± 0.043	0.575 ± 0.050	1.176 ± 0.010	0.984 ± 0.015	0.484 ± 0.008	0.713 ± 0.009	0.836 ± 0.035	0.808 ± 0.010	0.580 ± 0.008
5xNN Uncons.	0.230 ± 0.001	0.161 ± 0.010	0.535 ± 0.021	0.688 ± 0.009	1.016 ± 0.021	0.453 ± 0.011	0.685 ± 0.011	0.573 ± 0.037	1.036 ± 0.014	0.992 ± 0.012
5xNN Reg.	0.352 ± 0.001	0.365 ± 0.039	0.707 ± 0.019	1.239 ± 0.009	1.161 ± 0.012	0.564 ± 0.008	0.807 ± 0.017	1.014 ± 0.014	1.048 ± 0.008	0.919 ± 0.009
5xNN Anch.	0.349 ± 0.001	0.327 ± 0.034	0.623 ± 0.042	1.251 ± 0.011	1.295 ± 0.013	0.624 ± 0.006	0.868 ± 0.002	1.098 ± 0.035	1.238 ± 0.013	1.191 ± 0.014
					Entropy Relative Advantage					
1xNN Reg. to 5xNN Anch.	6.4%	29.2%	8.3%	6.4%	31.6%	28.9%	21.7%	31.3%	53.2%	105.3%
5xNN Uncons. to 5xNN Anch.	51.7%	103.1%	16.4%	81.8%	27.5%	37.7%	26.7%	91.6%	19.5%	20.1%
5xNN Reg. to 5xNN Anch.	-0.9%	-10.4%	-11.9%	1.0%	11.5%	10.6%	7.6%	8.3%	18.1%	29.6%

A.5 Experimental Details

Introduction to Anchored Ensembles

Experimental details for figure 4.1 are as follows.

Six randomly generated data points were used.

Hyperparameters: activation = ERF, σ_ϵ^2 = 0.003, b_1 variance = 1, W_1 variance = 1, H = 100, M = 3 (number of ensembles), optimiser = adam, epochs = 400, learning rate = 0.005.

Panel of Inference Methods

Experimental details for figure 4.4 are as follows.

Same six data points were used for all methods and activation functions, generated by $y = x\sin(5x)$, evaluated at, [-0.8, -0.1, 0.02, 0.2, 0.6, 0.8].

Hyperparameters: b_1 variance = 10, W_1 variance = 10, H = 100, M = 10, epochs= 4,000, σ_ϵ^2 = 0.001, leaky ReLU α = 0.2, optimiser = adam, MC Dropout probability = 0.4, MC Dropout samples = 200, HMC step size = 0.001, HMC no. steps = 150, HMC burn in = 500, HMC total samples = 1000, HMC predict samples = 50, VI predict samples = 50, VI iterations = 2000, VI gradient samples = 200.

Ensembling Loss Functions

Experimental details for figure 4.7 are as follows.

Regression

Generated $\mathbf{X}$ by sampling 20 points linearly spaced from the interval [-1.5, 1.5], $y = sin(2x) + \epsilon$ with $\epsilon \sim \mathcal{N}(0, 0.2^2)$. The y value corresponding to the largest x value was shifted -0.4 to produce a slight outlier.

Sub-plot A was trained via mean square error, B was regularised, C was anchored. D shows a ReLU GP.

Hyperparameters: activation = ReLU, σ_ϵ^2 = 0.08, b_1 variance = 10, W_1 variance = 10, H = 1000, optimiser = adam, epochs = 2,000, learning rate = 0.003, M = 10, hidden layers = 1.

Classification

Generated $\mathbf{X}$ using sklearn's 'make blobs' function, n samples = 30.

Sub-plot A was trained via cross entropy, B was regularised, C was anchored. D shows inference with HMC.

Hyperparameters: activation = ReLU, b_1 variance = 15/2, W_1 variance = 15/2, b_2 variance = 1/50, W_2 variance = 1/50, W_3 variance = 10/50, H = 50, optimiser = adam, epochs = 100, learning rate = 0.001, M = 10, hidden layers = 2.

1-D Convergence Plots

Experimental details for figure 4.8 are as follows.

Data as in section A.5 was used, with M = [3,5,10,20].

Hyperparameters: activation = ReLU, σ_ϵ^2 = 0.001, b_1 variance = 20, W_1 variance = 20, H = 100, optimiser = adam, epochs = 4,000, learning rate = 0.005.

KL Convergence Results

Experimental details for figure 4.9 are as follows.

Training was done on 50% of the data, with KL computed over the other 50%. Results were averaged over ten runs. The 'ideal' line shows the metric when posterior samples from the GP itself, rather than anchored NNs, were used.

The Boston Housing dataset was used, with 50% of data used for training, and testing on the other 50%.

Hyperparameters: activation = ReLU, σ_ϵ^2 = 0.1, b_1 variance = 2, W_1 variance = 2, H = [4, 16, 64, 256, 1024], M = [3,5,10,20,40], optimiser = adam, no. runs = 10, epochs = 1,000, learning rate = 0.001 when $H < 20$ else learning rate = 0.0002.

Regression Benchmarking Experiments

We complied with the established protocol (Hernández-Lobato and Adams, 2015). Single-layer NNs of 50 nodes were used, experiments repeated 20 times with random train/test splits of 90%/10%. The larger Protein and Song datasets allow 100 node NNs, and were repeated five and one time respectively.

The hyperparameter tuning process and final settings for experiments in table 4.1, A.1 & A.2 are as follows.

Hyperparameter Tuning

Hyperparameter tuning was done on a single train/validation split of 80%/20%. We found it convenient to begin by tuning data noise variance and prior variances. We restricted the prior variance search space by enforcing, $\sigma^2_{W_1} = \sigma^2_{b_1}/D$, and $\sigma^2_{W_2} = 1/H$. We therefore had only two hyperparameters to optimise initially: $\sigma^2_{b_1}$ and σ^2_{ϵ}. We did this with the GP model, using grid search, maximising marginal log likelihood over the training portion, and minimising NLL of the validation portion. For the larger datasets, when inference over the 80% training portion was too slow, we reduced the training split to 2,000 data points.

Hyperparameters for priors and data noise estimates were shared between the GP and anchored ensembles. Hyperparameters requiring tuning specifically for anchored ensembles were batch size, learning rate, number of epochs and decay rate. This was done on the same 80%/20% split used to select data noise and prior variance. We used random search, directed by our knowledge of the optimisation process (e.g. a lower learning rate requires more epochs to converge), minimising NLL on the validation portion.

We did not retune hyperparameters from scratch for the double layer NN (5x 50-50 NNs). We used settings as for the single-layer NNs (5x 50 NNs), but divided learning rate by 4, and multiplied epochs by 1.5.

For the single regularised NN with constant noise, we again used hyperparameters as for the single-layer ensemble (5x 50 NNs), tuning only the constant amount of variance to be added on the same 80%/20% split.

Hyperparameter Settings

Table A.5 provides the key hyperparameters used. The adam optimiser was used for all experiments. ReLU activations were used for all except the ERF GP (prior variance was separately tuned for this, values aren't given in the table).

Out-of-Distribution Classification

Fashion MNIST

We trained a three-layer NN on eight of ten classes of Fashion MNIST. We trained on 48,000 examples, tested on 8,000.

Experiments were repeated 5 times with a different random seed for each run.

Data categories were created as suggested by their name in table A.4. Examples are shown in figure A.1.

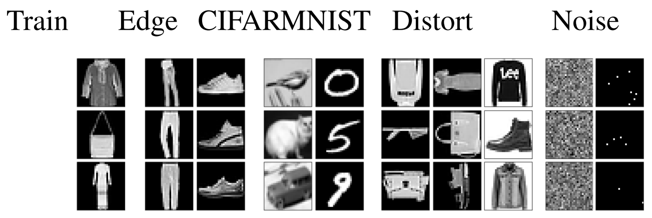


Figure A.1 Fashion MNIST OOD data examples.

- **Distort** comprised of rotations, vertical flips, and pixel value inversions.
- **Noise** comprised of iid Gaussian noise, mean = 0.0, standard deviation = 2.0.
- **Sparse** comprised of iid Bernoulli noise, pixles were given a value of 50.0 with p = 0.005, else 0.0.

Hyperparameters: activation = ReLU, optimiser = adam, epochs = 30, learning rate = 0.005, batch size = 256, hidden layers = 3, hidden units = 100

Table A.5 Hyperparameters used for regression benchmark results.

	N	Batch Size	Learn Rate	$\hat{\sigma}_\epsilon^2$	b_1 variance	W_1 variance	No. Epochs	Decay Rate	Single NN var.
Boston	506	64	0.05	0.06	10	0.77	3000	0.995	0.45
Concrete	1,030	64	0.05	0.05	40	5.00	2000	0.997	0.28
Energy	768	64	0.05	1e-7	12	1.50	2000	0.997	0.03
Kin8nm	8,192	256	0.10	0.02	40	5.00	2000	0.998	0.32
Naval	11,934	256	0.10	1e-7	200	12.50	1000	0.997	0.03
Power	9,568	256	0.20	0.05	4	1.00	1000	0.995	0.24
Protein	45,730	8192	0.10	0.5	50	5.56	3000	0.995	0.71
Wine	1,599	64	0.05	0.5	20	1.82	500	0.997	0.77
Yacht	308	64	0.05	1e-7	15	2.50	3000	0.997	0.10
Song Year	515,345	32768	0.01	0.7	2	0.02	500	0.996	0.84

CIFAR-10

CIFAR-10 contains 50,000 32x32 color training images, labelled over 10 categories, and 10,000 test images.

We removed 2 categories during training (ships, dogs) so trained over 40,000 examples.

OOD data classes are as show in the images in table 4.2.

- **Scramble** permuted each row of pixels in a given image.
- **Invert** took the negative of the pixel values.
- **Noise** sampled pixels from bernoulli distribution (p=0.005) of large magnitude (pixel value=50).

NN architecture: A convolutional NN was used, with the following structure, 64-64-maxpool-128-128-maxpool-256-256-256-maxpool-512-512-512-maxpool-flatten-2048fc-softmax.

All convolutional kernels were [3 x 3 x number of channels in previous layer]. All maxpooling kernels were [2 x 2]. The total number of parameters was 8,689,472.

Hyperparameters: activation = ReLU, optimiser = adam, learning rate = 0.001 decreasing to 0.0005 after 10 epochs and to 0.0001 after 20 epochs, batch size = 300.

In order to bring test accuracies and confidence on the training dataset roughly in line, it was necessary to train for a different number of training epochs for each method (this effectively applies early stopping to the unconstrained case). Anchored eps = 25, Regularise eps = 30, Unconstrained eps = 15.

Experiments were repeated 3 times with a different random seed for each run.

IMDb

Dataset of 25,000 movie reviews, labelled as positive or negative.

Example: *"this movie is the best horror movie bar none i love how stanley just dumps the women into the lake i have been a fan of judd nelson's work for many years and he blew me away its a blend of horror and ... "*

OOD data classes were generated as follows.

- **Reuters** - taken from the Reuters news dataset.

Example: *"said it has started talks on the possible ... of the company with various parties that it did not identify the company said the talks began after it ... "*

- **Random 1** - A single integers sampled uniformly at random from $\{1\ldots\text{vocabulary size}\}$ and converted to a repeated sequence of words.

  Example: *"member member member member member member member member member member member member ... "*

- **Random 2** - One integer per word sampled uniformly at random from $\{1\ldots\text{vocabulary size}\}$ and converted to words.

  Example: *"twists mentally superb finest will dinosaur variety models stands knew refreshing member spock might mode lose leonard resemble began happily names... "*

- **Random 3** - As for Random 2, but now only sample from least commonly used 100 words.

  Example: *"computers towers bondage braveheart threatened rear triangle refuse detectives hangs bondage firmly btw token 1990s mermaid reeves landed dylan remove hum natives insightful demonic... "*

NN architecture: used an embedding layer (outputting 20 dimensions), followed by 1D convolutional layer using 50 filters with kernel size of 3 words. Finally a hidden layer with 200 hidden nodes.

Hyperparameters: activation = ReLU, optimiser = adam, learning rate = 0.001, batch size = 64, max sentence length = 200, vocabulary size = 6000

Experiments were repeated 5 times with a different random seed for each run.

Reinforcement Learning

Uncertainty-Aware Reinforcement Learning

The FetchPush environment from OpenAI Gym was used with the sparse rewards setting. We modified the environment slightly. The goal was positioned at a fixed radius from the block (but at varying angle). Actions were discretised and vertical movements removed so the agent had a choice of moving 0.4 units forward/backwards/left/right. Gaussian noise was added to the actions to make the problem stochastic. Inputs were preprocessed so that relative coordinates of gripper to cube and cube to goal were provided directly to the NNs.

We used fixed target NNs which were updated every 500 episodes.

The simulation was run for 40,000 episodes, with final average rewards around -0.4. Two-layer NNs of 50 nodes were used. Learning rate = 0.001, batch size = 100, episodes in between training = 100, γ = 0.98, buffer size = 100,000.

Appendix B

High-Quality Prediction Intervals

B.1 Experimental details

In this section we give full experimental details of the work described in chapter 5, along with several additional results tables. Code is made available online (https://github.com/TeaPearce).

Qualitative Experiments

Training method: PSO vs. GD

For the qualitative training method comparison, PSO vs. GD (section 5.1), NNs used ReLU activations and 50 nodes in one hidden layer. GD was trained using $Loss_{QD-soft}$ and run for 2,000 epochs, PSO was trained using $Loss_{QD}$ and run for 50 particles over 2,000 iterations, parameters as given in SPSO 2011 were followed Thomas et al. (2012). Data consisted of 200 points sampled uniformly from the interval $[-2, 2]$.

Loss function: QD vs. MVE

For the loss function comparison, QD vs. MVE (section 5.2), NNs used Tanh activations and 50 nodes in one hidden layer. Both methods were trained with GD and results are for an individual NN (not ensembled). Data consisted of 200 points sampled uniformly from the interval $[-2, 2]$.

Model Uncertainty Estimation: Ensembles

For evaluation of ensembling (section 5.3), we sampled 50 points uniformly in the interval $[-4, -1]$, and another 50 from $[1, 4]$. An ensemble of ten QD NNs using ReLU activations and 50 nodes in one hidden layer was trained with GD, using parameter resampling.

Training Method / Loss Function Permutation

We provide quantitative results in table B.1 for the two synthetic datasets described in sections 5.1 & 5.2. These results cover permutations of loss function and training method [LUBE, QD, MVE]x[GD, PSO], using individual NNs (not ensembled).

Again, 200 training data points were sampled uniformly from the interval $[-2, 2]$. Validation was on 2,000 data points sampled from the same interval. Experiments were repeated ten times. PIs targeted 95% coverage.

LUBE relates to the 'softened' version of eq. (16), and QD to $Loss_{QD-soft}$. Softened versions were used for both GD and PSO (note 'soft' and 'hard' versions were contrasted in section 5.1).

For GD, 2,000 epochs were used, for PSO, 10 particles were run over 2,000 epochs. This meant the computational effort for PSO was five times that required by GD - the computational equivalent of 2,000 forward and backward passes for GD is 2 particles at 2,000 epochs for PSO.

NLL & RMSE metrics were computed, however should be viewed with caution for LUBE and QD - see section B.1.

All training methods and loss functions slightly overfitted the training data, producing PICP's lower than 95%. Generally GD-trained NNs outperformed their PSO counterparts in terms of the primary metric of their loss function ($MPIW$ for LUBE and QD, NLL for MVE), although quality of other metrics was similar. QD produced comparable results to LUBE - we note that our contributions to the loss function were from a theoretical and usability perspective and did not expect a large impact on performance. MVE produced PIs of comparable width to LUBE and QD for the case of normal noise, but PIs were significantly wider in the exponential noise case.

Table B.1 Full permutation results of training method and loss function on synthetic data with differing noise distributions. Mean ± one standard error.

	LUBE		MVE		QD	
	GD	PSO	GD	PSO	GD	PSO
			NORMAL NOISE			
PICP	0.90 ± 0.01	0.91 ± 0.01	0.91 ± 0.01	0.93 ± 0.01	0.91 ± 0.01	0.91 ± 0.01
MPIW	1.16 ± 0.05	1.13 ± 0.04	1.11 ± 0.04	1.00 ± 0.02	0.97 ± 0.04	1.07 ± 0.04
RMSE	0.42 ± 0.01	0.43 ± 0.01	0.39 ± 0.00	0.38 ± 0.01	0.40 ± 0.01	0.41 ± 0.01
NLL	0.60 ± 0.07	0.47 ± 0.06	-0.44 ± 0.02	-0.23 ± 0.04	0.13 ± 0.08	0.37 ± 0.08
			EXPONENTIAL NOISE			
PICP	0.91 ± 0.01	0.92 ± 0.01	0.93 ± 0.01	0.95 ± 0.00	0.91 ± 0.01	0.93 ± 0.01
MPIW	0.80 ± 0.02	0.93 ± 0.04	1.07 ± 0.04	0.99 ± 0.03	0.84 ± 0.03	0.98 ± 0.04
RMSE	0.39 ± 0.00	0.40 ± 0.01	0.38 ± 0.01	0.37 ± 0.01	0.39 ± 0.01	0.41 ± 0.01
NLL	0.36 ± 0.09	0.64 ± 0.13	-0.48 ± 0.02	-0.18 ± 0.02	0.40 ± 0.05	0.54 ± 0.13

Benchmarking Experiments

Set up and Hyperparameters

For the benchmarking section, experiments were run across ten open-access datasets, train/test folds were randomly split 90%/10%, with experiments repeated 20 times, input and target variables were normalised to zero mean and unit variance. NNs had 50 neurons in one hidden layer with ReLU activations. The exceptions to this were for experiments with the two largest datasets, *Protein* and *Song Year*, where NNs had 100 neurons in one hidden layer, and were repeated five times and one time respectively.

The softening factor was constant for all datasets, $s = 160.0$. For the majority of the datasets $\lambda = 15.0$, but was set to 4.0 for *naval*, 40.0 for *protein*, 30.0 for *wine*, and 6.0 for *yacht*. The Adam optimiser was used with batch sizes of 100. Five NNs were used in each ensemble, using parameter resampling.

Hyperparameters requiring tuning were learning rate, decay rate, λ, initialising variance, and number of training epochs. Tuning was done on a single 80%/20% train/validation split and using random search.

NLL & RMSE Results

In table B.2 we report NLL & RMSE in unnormalised form to be consistent with previous works. Note that in the main results (section 6) we found it more meaningful to leave $MPIW$ in normalised form so that comparisons could be made across datasets.

To compute NLL & RMSE for QD-Ens, we used the midpoint of the PIs as the point estimate to calculate RMSE. We computed the equivalent Gaussian distribution of the PIs by centering around this midpoint and using a standard deviation of $(y_{Ui} - y_{Li})/3.92$ (since the PI represented 95% coverage), which enabled NLL to be computed. We emphasise that by doing this, we break the distribution-free assumption of the PIs, and include these purely for the purpose of consistency with previous work. Unsurprisingly, NLL & RMSE metrics for QD-Ens are poor. MVE-Ens results are in line with previously reported work Lakshminarayanan et al. (2017).

Table B.2 RMSE and NLL on ten benchmarking regression datasets; mean ± one standard error, best result in bold.

			RMSE		NLL	
	n	D	MVE-Ens	QD-Ens	MVE-Ens	QD-Ens
Boston	506	13	**2.84 ± 0.19**	3.38 ± 0.26	**2.60 ± 0.10**	2.74 ± 0.14
Concrete	1,030	8	**5.20 ± 0.10**	5.76 ± 0.10	**2.95 ± 0.04**	3.10 ± 0.02
Energy	768	8	**1.67 ± 0.05**	2.30 ± 0.04	**1.12 ± 0.05**	1.62 ± 0.06
Kin8nm	8,192	8	**0.08 ± 0.00**	0.09 ± 0.00	**-1.28 ± 0.01**	-1.14 ± 0.01
Naval	11,934	16	**0.00 ± 0.00**	**0.00 ± 0.00**	**-5.67 ± 0.03**	-5.73 ± 0.03
Power plant	9,568	4	**3.94 ± 0.03**	4.10 ± 0.03	**2.77 ± 0.01**	2.83 ± 0.01
Protein	45,730	9	**4.35 ± 0.02**	4.98 ± 0.02	**2.74 ± 0.02**	3.12 ± 0.02
Wine	1,599	11	**0.62 ± 0.01**	0.65 ± 0.01	**1.07 ± 0.06**	1.15 ± 0.03
Yacht	308	6	1.36 ± 0.09	**1.00 ± 0.08**	1.02 ± 0.05	**0.76 ± 0.07**
Song year	515,345	90	**8.88 ± ± NA**	9.30 ± ± NA	**3.37 ± ± NA**	3.58 ± ± NA