Compromise-free Bayesian neural networks

Kamran Javid*,†‡, Will Handley*,†§, Mike Hobson*, and Anthony Lasenby*,†

Abstract. We conduct a thorough analysis of the relationship between the out-of-sample performance and the Bayesian evidence of Bayesian neural networks (BNNs) using the Boston housing dataset, as well as looking at the performance of ensembles of BNNs. We numerically sample without compromise the full network posterior and obtain estimates of the Bayesian evidence using the publicly available version of the nested sampling algorithm PolyChord¶ (Handley et al., 2015a,b), considering network models with up to 156 trainable parameters*(Javid and Handley, 2020). The networks have between zero and four hidden layers, either tanh or ReLU activation functions, and with and without hierarchical priors (MacKay, 1992c; Neal, 2012). The ensembles of BNNs are obtained by determining the posterior distribution over networks, from the posterior samples of individual BNNs re-weighted by the associated Bayesian evidence values. From the out-of-sample performance of the BNNs with ReLU activations, it is clear that they outperform BNNs of the same architecture with tanh activations, and evidence values corresponding to the former reflect this in their relatively high values. Looking at the models with hierarchical priors, there is a good correlation between out-of-sample performance and evidence, as was found in MacKay (1992c), as well as a remarkable symmetry between the evidence versus model size and out-of-sample performance versus model size planes. The BNNs predictively outperform the equivalent neural networks trained with a traditional backpropagation approach, and Bayesian marginalising/ensembling over architectures acts to further improve performance.

MSC 2010 subject classifications: 62, 65, 68.

Keywords: Bayesian Neural Networks, Nested sampling, Boston Housing dataset.

1 Introduction

In an age where machine learning models are being applied to scenarios where the associated decisions can have significant consequences, quantifying model uncertainty is becoming more and more crucial (Krzywinski and Altman, 2013; Ghahramani, 2015). Bayesian neural networks (BNNs) are one example of a model which provides its own uncertainty quantification, and have gained popularity in-part due to the successes of the conventional backward propagation trained neural networks (Rumelhart et al., 1985).

BNNs have a history stretching back almost as far as research on traditional neural networks (TNNs), with Denker et al. (1987); Tishby et al. (1989); Buntine and
Weigend (1991); Denker and Lecun (1991) laying the foundations for the application of BNNs. Major breakthroughs occurred thanks to the work of MacKay and his success with BNNs in prediction competitions (MacKay et al., 1994). Prior to this, MacKay published several papers (MacKay, 1992a,c,b) detailing his methods and highlighting several important aspects of BNNs. He trained the networks by using quadratic approximations of the posteriors, and incorporated low-level hierarchical Bayesian inference into his modelling to learn the prior distribution variances. From this he obtained estimates for the Bayesian evidence, and found a positive correlation between the evidence and the BNNs’ ability to make predictions on out-of-sample data well. Neal (1992, 1993) focused on improving the predictive power of BNNs, by relaxing the Gaussian approximation by instead sampling the posterior using Hamiltonian Markov chain Monte Carlo (MCMC) methods. He also incorporated more complex hierarchical Bayesian modelling into his methods by using Gibbs sampling to sample the variances associated with both the priors and likelihood. He found that predictions with these BNNs consistently outperformed the equivalently sized networks trained using backward propagation techniques. More recently de Freitas et al. (2000); de Freitas (2003); Andrieu et al. (2003) used reversible jump MCMC and sequential MC methods to train BNNs and do model selection without calculating Bayesian evidences, by parameterising the different networks as a random variable, and sampling from the resultant posterior. In general they found that their methods produce networks which perform well, but were much slower to train than expectation maximisation-based networks. Indeed, methods which are more efficient in the training of networks have gained popularity in the recent years, due to the success of deep learning. Two of the most commonly used methods for training scalable networks are variational inference (see e.g. Hinton and Van Camp, 1993; Barber and Bishop, 1998; Graves, 2011) and dropout training (Gal and Ghahramani, 2016a). The latter has proved to be particularly popular, due to the fact that dropout BNNs can be trained using standard backward propagation techniques, and can be applied to recurrent and convolutional networks (Gal and Ghahramani, 2016b; Kendall and Gal, 2017).

In this paper we present what we refer to as compromise-free BNNs as a proof-of-concept idea for training BNNs, conditional on computational resources. As in Higson et al. (2018), we make no assumptions about the functional form of the posteriors, and fully numerically sample the posterior distributions using the nested sampling algorithm PolyChord (Handley et al., 2015a,b) to train the BNNs. Furthermore we obtain numerical estimates for the Bayesian evidences associated with each BNN, with which we analyse the relationship between the evidence and out-of-sample performance as first done in MacKay (1992c). We consider a wide array of different networks, with between zero and four hidden layers, tanh or ReLU activation functions, and varying complexities of hierarchical priors following MacKay (1992c); Neal (2012), to obtain a thorough understanding of the evidence and out-of-sample performance through various cross sections of the BNN architecture space. Similar to de Freitas (2003) we look at the posterior over networks as a form of model selection. However in our case, the posteriors are obtained from the samples of the individual network posteriors re-weighted according to the evidences associated with a given run. We then marginalise over these network posteriors to obtain predictions from ensembles of BNNs. A preliminary analysis along these lines was conducted by Higson et al. (2018), who also explored using an
adaptive method akin to de Freitas (2003). The adaptive approach has the feature of severely undersampling less preferred models and taking (often substantially) less time than sampling models individually, and will be explored in a future work.

In Sections 2 and 3 we briefly review the theory of Bayesian neural networks and establish notation. In Section 4 we describe the Boston housing dataset, before summarising our methods in Section 5. Results for individual networks are presented in Section 6 and for ensembled networks in Section 7. A comparison with traditional networks is discussed in Section 8 before conclusions are drawn in Section 9.

2 Neural networks

A neural network is a parameterised function $f$ which maps a real input $x$ to a real output $y$, both of which can have arbitrary (finite) dimension. A fully connected multi-layer perceptron (MLP) is parameterised by network weights $w$ and biases $b$, and can be represented recursively using intermediate variables $z$ as

$$ f = z^{(L)}, \quad z^{(l)}_i = g(b^{(l)}_i + \sum_k w^{(l)}_{ik} z^{(l-1)}_k), \quad z^{(0)} = x, \quad (2.1) $$

where $g$ is an activation function. The activation function can take many forms\(^1\), but for the purposes of this paper we shall consider either $\text{tanh}$ ($g(x) = \text{tanh} x$) or ReLU ($g(x) = \max\{x, 0\}$) as the activation in the hidden layers, with a linear activation in the output layer. Equation (2.1) may be represented graphically as in Figure 1. The activation function(s), number of layers $L$, and the number of nodes within each layer $l_i$ together determine the architecture of the network. Using the notation of Equation (2.1), we can say the input $x$ has dimension (number of features) $l_0$ while the output $y$ has dimension $l_L$.

Traditional neural networks are trained in a supervised fashion on a set of example input data paired with the corresponding outputs. A misfit function is defined between the true values of the outputs and the values predicted by the network (for regression problems this is usually proportional to the sum of squared differences). It is also common to define a regularisation function $R(\theta)$ which seeks to penalise overly complex models. The network parameters $\theta = (b, w)$ are chosen such that

$$ \theta_* = \min_\theta \lambda_m \frac{\chi_{\text{train}}^2(\theta)}{n_{\text{train}}} + \lambda_r R(\theta), \quad \chi_{\text{train}}^2(\theta) = \sum_{i\in\text{train}} |y^{(i)} - f(x^{(i)}; \theta)|^2, \quad (2.2) $$

where $\lambda_m$ and $\lambda_r$ are hyperparameters dictating the relative weighting of the misfit versus the regularisation in the optimisation.\(^2\) $n_{\text{train}}$ is the number of training instances.

---

\(^1\)And indeed can in general vary between layers and nodes $g = g^{(l)}$.

\(^2\)It should be noted that in a minimisation context, there is a redundancy in including two regularisation parameters $\lambda_m$ and $\lambda_r$, so people usually therefore without loss of generality set $\lambda_m = 1$. In a Bayesian context this redundancy is removed since the full posterior is composed of likelihood and prior terms whose widths are each controlled by a separate regularisation parameter. We thus follow MacKay et al. (1994) and retain both $\lambda_m$ and $\lambda_r$ here.
Compromise-free Bayesian neural networks

of the input/output pair \((x, y)\) (number of training records). These network trained parameters \(\theta_*\) can then be tested on further data not used in the training procedure to test out-of-sample performance using \(\chi^2\) test \((\theta_*\).

There are two key issues that immediately arise from the traditional approach. The first is that the network gives no indication of the confidence in its prediction \(y_{\text{pred}} \equiv f(x_{\text{new}}; \theta_*)\) on unseen inputs \(x_{\text{new}}\). It would be preferable if the network were to provide an error bar \(y_{\text{pred}} \pm \sigma_{y_{\text{pred}}}\) for its confidence, and that this error bar should become larger as the network extrapolates beyond the domain of the original training data.

The second issue arises from the arbitrariness in the choice of hyperparameters. There is little guidance from the formalism above as to how to choose the architecture of the network defining Equation (2.1) and represented in Figure 1, and the hyperparameters in Equation (2.2). Networks that are too large/complex may overfit the data, while networks too small/simple will likely underfit. The most common method of finding a ‘happy medium’ is through the use of cross validation (Stone, 1977; Efron, 1979; Li et al., 1985), but searching the associated hyperparameter space can be time consuming and arbitrary unless one resorts to techniques such as Bayesian optimisation (Snoek et al., 2012).

3 Bayesian neural networks . . .

3.1 . . . in theory

The Bayesian approach to neural networks aims to ameliorate the two difficulties discussed in Section 2 by sampling the parameter space rather than optimising over it. Here we consider the misfit function \(\chi^2_{\text{train}}(\theta)\) in terms of an independent Gaussian likelihood

\[
P(D_{\text{train}}|\theta, M) = \prod_{i \in \text{train}} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - f(x^{(i)}, \theta))^2}{2\sigma^2}\right) = \frac{\exp\left(-\frac{\chi^2_{\text{train}}(\theta)}{2\sigma^2}\right)}{(\sqrt{2\pi}\sigma)^{N_{\text{train}}}},
\]

where \(D_{\text{train}}\) are the training data, \(\sigma^2\) is a misfit variance (which plays a similar role to the parameter \(\lambda_m\) in Equation (2.2)), and \(M\) is the network architecture (or Bayesian model). The likelihood \(L\) can be related to a posterior \(P\) on the parameters \(\theta\) using a prior measure on the network parameter space \(P(\theta|M) \equiv \pi\) (which draws parallels with \(R(\theta)\), see Section 3.3 and Higson et al. 2018) via Bayes theorem

\[
P(\theta|D_{\text{train}}, M) = \frac{P(D_{\text{train}}|\theta, M)P(\theta|M)}{P(D_{\text{train}}|M)} \equiv \frac{L\pi}{Z}.
\]

The process of determining the posterior is termed parameter estimation. Instead of having a single best-fit set of network parameters \(\theta_*\), one now has a distribution over \(\theta\). A predictive distribution for \(y_{\text{pred}}\), from unseen inputs \(x_{\text{new}}\) by the network is induced by the posterior, and may be computed by marginalisation (Handley and Millea, 2019)

\[
P(y_{\text{pred}}|x_{\text{new}}, D_{\text{train}}, M) = \int \delta(y_{\text{pred}} - f(x_{\text{new}}; \theta))P(\theta|D_{\text{train}}, M)d\theta.
\]
Figure 1: The neural network architectures considered in this paper. These are graphical representations of Equation (2.1) with inputs \( x (l_0 = 13) \) and outputs \( y (l_L = 1) \) illustrated as squares on the left and right hand sides, intermediate node values \( \nu \) as circles arranged in vertical layers \( \ell \) and weights \( w \) and biases \( b \) as solid lines. Each architecture is summarised by a list of numbers in parentheses, giving the number of nodes in each hidden layer.
Compromise-free Bayesian neural networks

\[ d \frac{\partial}{\partial y_{\text{pred}}} \int_{f(x_{\text{new}}; \theta) < y_{\text{pred}}} P(\theta | D_{\text{train}}, M) d\theta. \]  

(3.4)

If desired, this above distribution can be compressed into summary statistics such as a mean prediction and error bar

\[ \hat{y}_{\text{pred}} = \int P(\theta | D_{\text{train}}, M) f(x_{\text{new}}; \theta) d\theta = \langle f(x_{\text{new}}; \theta) \rangle \]  

(3.5)

\[ \sigma_{y, \text{pred}}^2 = \int P(\theta | D_{\text{train}}, M) (f(x_{\text{new}}; \theta) - \hat{y}_{\text{pred}})^2 d\theta = \text{var}(f(x_{\text{new}}; \theta)). \]  

(3.6)

The denominator of Equation (3.2) is termed the Bayesian evidence, and is computed from the likelihood and prior as a normalisation constant

\[ Z \equiv P(D_{\text{train}} | M) = \int P(D_{\text{train}} | \theta, M) P(\theta | M) d\theta \equiv \int \mathcal{L} \pi d\theta. \]  

(3.7)

The evidence is critical in the upper level of Bayesian inference, termed model comparison whereby one’s confidence in the network as supported by the data is given by

\[ P(M | D_{\text{train}}) = \frac{P(D_{\text{train}} | M) P(M)}{\sum_m Z_m P(M)} = \frac{Z_M P(M)}{\sum_m Z_m P(m)}. \]  

(3.8)

In the above posterior over models, \( m \) is a categorical variable ranging over all architectures considered, and \( P(m) \) is the assigned prior probability to each network (typically taken to be uniform over all \( m \)). The evidence is therefore a measure of the quality of a network as viewed by the data and can be used to compare architectures.

One can of course use evidences to marginalise out network dependence completely and obtain values for \( \hat{y}_{\text{pred}} \) and \( \sigma_{y, \text{pred}} \) corresponding to the posterior distribution over network architectures. The equivalent of Equation (3.3) becomes

\[ P(y | x, D_{\text{train}}) = \sum_m P(y | x, D_{\text{train}}, m) P(m | D_{\text{train}}), \]  

(3.9)

with corresponding marginal summary statistics such as:

\[ \hat{y}_{\text{pred}} = \sum_m \hat{y}_{\text{pred}}(m) P(m | D_{\text{train}}), \]  

(3.10)

where \( \hat{y}_{\text{pred}}(m) \) are the corresponding means from Equation (3.5) conditioned on model \( m \). In using Equation (3.9), one is effectively marginalising over an ensemble of networks, weighted by the quality of the fit as viewed by the data versus the model complexity.

The Bayesian evidence \( Z \) in Equation (3.7) is the average of the likelihood function over the sampling parameter space, weighted by the prior distribution. A larger parameter space, either in the form of higher dimensionality or a larger domain results in a lower evidence value, all other things being equal. Thus the evidence automatically
implements Occam’s razor: when you have two competing theories that make similar predictions, the one with fewer active (i.e. constrained) parameters should be preferred.

This naturally has useful implications in the context of machine learning: for two models which fit the training data equally well, one would expect the simpler model (i.e. the one with the higher Bayesian evidence) to generalise to out-of-sample data better, due to it overfitting the training data less. Thus one can postulate that the Bayesian evidence can be used as a proxy for out-of-sample data performance of a model relative to alternative models. Indeed MacKay (1992c) finds a good correlation between \( Z \) and generalisation ability for small neural networks applied to regression problems. Furthermore MacKay found that for models where this correlation did not exist, the models generally performed poorly on test set data, but when they were improved in some way (in the particular example presented, MacKay improved performance by increasing the granularity of the variable prior hyperparameters, see the next Section) then the correlation was found to exist. Thus in this instance the correlation (or lack thereof) between the evidence and out of sample performance can also be interpreted as a tool for determining when a model’s performance on out-of-sample data can be improved.

3.2 ... in practice

Whilst the likelihood in Equation (3.1) is Gaussian with respect to the data, it is highly non-Gaussian with respect to the parameters \( \theta \). We must therefore turn to numerical Bayesian inference, for which the critical concept is that of sampling a distribution.

Sampling from a distribution provides a natural compression scheme, encoding the critical information in a general posterior \( P(\theta) \) as a set of weighted samples drawn from it. From samples, one may perform the otherwise challenging but critical operations of marginalisation and transformation of distributions with ease. Marginalisation amounts to ignoring coordinates, and from a set of samples from \( P(\theta) \) one can easily generate samples from an alternative distribution \( P(q) \) where \( q = q(\theta) \) by applying \( q \) to each sample. This is very helpful for producing samples from a predictive distributions such as those given by Equations (3.3) and (3.9).

In the context of Bayesian neural networks, one can consider the traditional predictive procedure of using \( y_{\text{pred}} = f(x_{\text{new}}, \theta_*) \) as being extended to using a set of predictions given by \( f(x_{\text{new}}, \theta) \) for the sampled values of \( \theta \), each weighted by the corresponding \( P(\theta) \) to give the posterior over \( y_{\text{pred}} \) (and with it summary statistics such as \( \hat{y}_{\text{pred}} \) and \( \sigma_{y_{\text{pred}}} \)).

Samples from the full ensemble distribution over different networks can be generated by re-weighting so that the posterior mass of each individual network is proportional to its evidence. This generates a set of samples from the full joint distribution. In the context of MLP neural networks, different models can take several forms, including different numbers of layers, different numbers of nodes within the layers, different activation functions, and in our case, even different granularities on the prior hyperparameters (see Section 3.3). The beauty of using the full joint distribution is it allows the data to decide which models are most important in the fitting process, arguably
putting less onus on the user as they now only have to decide on a suite of models to choose from, and the associated prior probabilities of these models.

Here we use the PolyChord algorithm (Handley et al., 2015a,b), a high-dimensional, high-performance implementation of nested sampling (Skilling, 2006) to obtain samples from the full numerical distribution and simultaneously compute the Bayesian evidence. We note that PolyChord is also capable of sampling directly from the joint posterior of networks and parameters (Hee et al., 2015; Chua et al., 2018; Higson et al., 2018). However we adopted the method of combining individual runs for parallelisation efficiency, since each run could be done completely independently on separate models.

### 3.3 The prior

In order to begin the process of Bayesian inference, we must specify the prior. Throughout, we take the prior on the network parameters and bias terms to be independent normal distributions with width $\sigma_i$ so each component $\theta_i$ has

$$P(\theta_i|\sigma_i, M) = \frac{1}{\sqrt{2\pi \sigma_i}} \exp \left( -\frac{\theta_i^2}{2\sigma_i^2} \right).$$

(3.11)

Note that in the traditional approach denoted by Equation (2.2) defining $R$ to be a function of the $l_2$ norm of $\theta$ was inspired from the use of a Gaussian prior over the network parameters. In order to fully specify the model, one must also specify the individual prior and likelihood widths $\sigma_i$ and $\sigma$, which play an analogous role to the regularisation parameters $\lambda_r$ and $\lambda_m$ in Equation (2.2) respectively. Since the data $x$ and $y$ can be (and are generally) whitened to have zero mean and unit variance (Section 4), a not unreasonable choice is to set the likelihood variance to one. Setting $\sigma_i = 1$ for all $i$ is equivalent to setting $\lambda_r = \frac{1}{2}$.

In the spirit of model comparison however, an extended strategy is to check whether treating $\sigma_i$ and $\sigma$ as free hyperparameters alongside the network parameters provides better performance and to see how the Bayesian evidence is affected. This necessitates specifying a further hyperprior on the widths, which we will in general take to be a gamma prior on the precision $\tau_i = \sigma_i^{-2}$

$$P(\sigma_i|\alpha_i, \beta_i, M) = \frac{2^{\alpha_i} \beta_i^{\alpha_i}}{\Gamma(\alpha_i)} \sigma_i^{-2\alpha_i-1} e^{-\beta_i/\sigma_i^2},$$

(3.12)

with an equivalent expression for a distribution on $\sigma$ with hyperparameters $\alpha$ and $\beta$. This procedure of treating the widths $\sigma_i$ as additional parameters is common in hierarchical Bayesian inference, and is equivalent to letting the data decide the width of the distributions from which the network parameters are sampled. In the language of traditional neural network training, this is equivalent to letting the data decide the regularisation factor. For the likelihood function, treating the width $\sigma$ as a random variable essentially lets us estimate the data noise from the data themselves.

The downside to this hierarchical approach is that we still have to assign the values of the hyperparameters $\alpha_i, \beta_i, \alpha, \beta$ for the hierarchical priors (discussed in Section 5.2). An
appropriately chosen hyperprior will be designed so that the choice of hyperparameters for the hierarchical prior has a diluted effect compared to choosing the hyperparameters for the base prior (i.e. \( \sigma = \sigma_i = 1 \)). We note that Neal (2012) used two-level hierarchical Bayesian inference to push this problem deeper: the hyperparameters of the hierarchical prior are themselves assigned a prior distribution, and it is this second-level hierarchical prior which has to have hyperparameters assigned deterministically. Ideally one would continue down the hierarchy until the Bayesian evidence tells us that we do not need to go any further, but in this work we only consider up to one-level hierarchical models, leaving a deeper analysis for future research.

### 3.4 Weight space symmetry

For the hidden layers in a neural network, a degeneracy between the weights/biases in different nodes exists within a given layer (Higson et al., 2018). In deep learning this is known as weight space symmetry (Goodfellow et al., 2016). For a fully-connected feed-forward neural network, the degeneracy arises due to the fact that any node is just a linear combination of the outputs of the previous (usually followed by a non-linear activation). Thus no node within a layer is unique, and so is degenerate with all other nodes in that layer. This means that for a neural network with \( L \) hidden layers, where the number of nodes in the layers is given by \((l_1, \ldots, l_L)\), then the total degeneracy of the network is \( \prod_{i=1}^{L} l_i! \). This degeneracy exponentially increases the size of parameter space to be explored without providing a better fit, and so should be avoided for computational efficiency whenever possible.

This problem may be resolved by using a forced identifiability prior (Handley et al., 2019; Buscicchio et al., 2019), which enforces an artificial ordering on degenerate parameters. When applied to the bias terms in a layer, this provides a labelling on the nodes and breaks the degeneracy between them.

Usually when using a sampling algorithm which samples from the unit hypercube such as PolyChord, one obtains a sequence of parameters \((\theta_1, \ldots, \theta_{\text{pars}})\) in physical space from their representations in the unit hypercube \((u_1, \ldots, u_{\text{pars}})\) using the inverse CDF function of the prior. When using a forced identifiability prior, an intermediate step enforces an ordering on the unit hypercube values, involving a reversed recurrence relation, starting by updating \(u_{\text{pars}} \rightarrow u_{\text{pars}}^{1/\text{pars}}\) and then \(u_i \rightarrow u_i^{-1} u_{i+1}\). This enforces an ordering \(u_1 < u_2 \ldots < u_{\text{pars}}\), thus breaking any switching degeneracy in \(\theta_i\) values.

### 4 Boston housing dataset

The Boston housing dataset\(^3\) has long been the focus of many analyses in supervised machine learning. We focus on this dataset mainly due to its small sample size, where Bayesian neural networks should be effective due to their robustness to overfitting relative to traditionally trained neural networks. Furthermore the small size of the data

---

\(^3\)https://archive.ics.uci.edu/ml/machine-learning-databases/housing/
means that a full Bayesian treatment of the problem is computationally feasible, including accurate calculation of the Bayesian evidence, a key facet of this paper. The dataset consists of 506 records, with 13 inputs (features) and one output. All 14 of these are continuous variables, and so the task is a regression problem. We transform the input and output variables so that they all have zero mean and unit variance in order to aid the deep learning process.

5 Methodology

5.1 Training and test set splits

To see how the Bayesian evidence \( Z \) correlates with out-of-sample performance we split the entire data in half so that both the training and test sets contain \( n_{\text{test}} = n_{\text{train}} = 253 \) records. We attribute a large split to the test data as it is vital to be confident in our out-of-sample performance. We evaluate the test set performance by looking at the mean squared error \( E = \frac{\chi^2_{\text{test}}}{n_{\text{test}}} \) between \( y_{\text{test}} \) and the mean BNN prediction \( \hat{y}_{\text{new}} \) (Equation (3.3)), as well as the error on \( E \) derived from \( \sigma_{y,\text{pred}} \).

To cross-validate and reduce scatter in the visualisation of our results we repeat the analysis with ten different random splits of the training/test data, meaning for a given setup, we train a BNN on ten different training sets, and measure their performance on the ten corresponding test sets. For these ten different data splits, we look at the average value of the mean squared errors on the test sets, and also take the mean value of \( Z \) obtained from the ten analyses. The values quoted in the rest of this analysis are the logarithm of the values of these average values of the evidence, and the average values of the mean squared errors. For a given BNN, there is no correlation between \( Z \) and test loss over different data randomisations (i.e. between the ten different splits of the data).

5.2 Neural network setups

Figure 1 details the neural network architectures we consider. We also consider a network with no hidden layer (i.e. Bayesian linear regression). For each architecture, we first consider networks with either a tanh or ReLU activation function for all hidden layers, with prior and likelihood widths fixed to \( \sigma_i = \sigma = 1 \).

For the tanh activation functions, we also consider the impact of hierarchical priors as discussed in Section 3.3. In this case we allow the likelihood precision \( \tau \equiv \sigma^{-2} \) to vary by setting a Gamma prior with \( \alpha = \beta = 1 \). For the hierarchical priors on the network parameter prior widths \( \sigma_i \), we consider three granularities of prior:

**Single granularity** One global precision hyperparameter \( \sigma_1 \) controlling the precision of all weights and biases in the network, with a Gamma prior on the precision with \( \alpha_1 = \beta_1 = 1 \).

---

\(^4\)For more information on the dataset see https://www.cs.toronto.edu/~delve/data/boston/bostonDetail.html.
Layer granularity Two precision hyperparameters $\sigma_i$ per layer ($2(L + 1)$ in total), one controlling the precision of the weights within a layer, the other of the bias parameters within a layer. A Gamma prior with $\alpha_i = 1$, $\beta_i = 1$ is used for $i$ in the first hidden layer and for the bias nodes in each layer. For subsequent hidden layers and the output layer, the weights of the $j^{th}$ layer were assigned a Gamma distribution with $\alpha_i = 1$, $\beta_i = 1/l_j$, following the scaling arguments of Neal (2012) (see Appendix A for more detail).

Input size granularity All weights within a layer multiplying the same activation $z$ share a precision hyperparameter $\sigma_i$. Bias hyperparameters are shared per-layer as in layer granularity. The total number of variable prior hyperparameters for the network is therefore $\sum_{j=0}^{L-1}(l_j + 1)$. The same scaling of the Gamma distributions as for layer granularity is adopted.

As an example of input size granularity consider the network with $(l_1, l_2) = (4, 4)$ (n.b. the input and output layers are $l_0 = 13$ and $l_3 = 1$ respectively). The three sets of biases for each layer each are assigned a Gamma hyperprior with $\alpha_i = \beta_i = 1$. The weights in the first hidden layer have 13 Gamma hierarchical priors with these same hyperparameters. The second hidden layer has four Gamma hyperpriors with $\alpha_i = 1$ and $\beta_i = 1/l_1 = 1/4$. The output node’s weights have a separate hyperprior assigned to them all with $\alpha_i = 1$ and $\beta_i = 1/l_1 = 1/4$. Thus the number of variable hyperparameters for this setup is $3 + 13 + 4 + 4 = 24$.

In the terminology of Bayesian statistics we shall refer to each combination of architecture, activation function and prior as a model.

5.3 Training the Bayesian networks

To obtain the posterior distributions of all the models considered, the PolyChord algorithm (Handley et al., 2015a,b) was run with 1000 live points $n_{\text{live}}$ and the number of repeats $n_{\text{repeats}}$ set to $5 \times$ the dimensionality of the parameter space (which vary between 14 and 156 dimensions, see Table 2). For nested sampling, $n_{\text{live}}$ acts as a typical resolution parameter, with runtime scaling linearly with $n_{\text{live}}$, and evidence and parameter estimation sampling error decreasing as the square root of the number of live points. Setting $n_{\text{live}} = 1000$ gives a good balance between computational feasibility and evidence accuracy. For PolyChord, the number of repeats $n_{\text{repeats}}$ serves as a reliability parameter, in that setting it too low is liable to generate algorithm-dependent biases, but there is a point beyond which setting it arbitrarily high brings no further gain. We followed the recommended procedure of checking our results are stable with respect to varying $n_{\text{repeats}}$. An example posterior produced by PolyChord is plotted in Figure 2.

6 Results for individual models

The individual BNNs were trained for all the models detailed in Section 5.2, meaning that a total of 49 models were considered (enumerated in Table 2 within Appendix B).

---

5 In analogy with the inverse of MultiNest’s efficiency $\text{efr}$ (Feroz et al., 2009)
Figure 2: Example posterior produced by PolyChord (Handley et al., 2015a,b) for the neural network shown in the bottom right corner with “single" granularity hyperpriors (Section 5.2). The full 45-dimensional posterior is sampled numerically. In the top panels the 1-dimensional marginal distributions for the network weights $w$ and biases $b$ in the first layer are shown. The three square plots show the pairwise 2-dimensional marginal posteriors for the next three layers, with 1-d marginals on the diagonals, representative samples drawn from the posterior in the upper triangle and histograms of the posterior in the lower triangle. The bottom centre plots show the posterior on the $a$-priori unknown Gaussian noise level $\sigma$ in the likelihood and single prior hyperparameter $\sigma_i$. It should be noted that the posteriors are highly non-Gaussian and multi-modal, necessitating the use of a full compromise-free sampler. Plot produced using anesthetic (Handley, 2019).
Furthermore, each of these was trained on ten different randomisations of the training/test split, meaning in total 490 runs were completed. Throughout this analysis we focus on the average values obtained over the ten instances of each BNN as mentioned in Section 5.1, but note that when we inspected the results from single instances of the randomisations, the same trends appeared albeit less coherently. Table 2 in Appendix B gives a summary of the average results obtained for the 49 different models, giving the test set loss values (mean squared error), the BNN’s estimate of the error of the test losses, propagated through the error on the model predictions, $\sigma_y$, as well as the log evidences and their errors obtained from inferences. The dimensionalities include any variable hyperparameters involved in the analyses. Note that the four models with no hidden layers points consistently have test losses around 0.35, much higher than the other models considered. This emphasises the importance of deep networks, even when variable hyperparameters are used. For the rest of the analyses we do not consider these results obtained from the no hidden layer networks.

Figure 3 shows the evidence versus the test loss for the 45 BNNs with hidden layers, from which one sees a clear separation between the BNNs with no variable hyperparameter (with either tanh or ReLU activations) and the variable hyperparameter BNNs (with tanh activations). Thus, straight away it is clear that the added complexity associated with variable hyperparameters is captured in the evidence values, but also provides an increase in performance.

Much of the improvement in evidence value can be understood by the fact that when hyperparameters are not varied the likelihood variance $\sigma$ is set to unity, in fact as shown in Figure 2 its desired value of $\sigma \sim 3.4 \pm 0.2$ is statistically significantly larger than this.

We now focus on different cross-sections of the set of models considered to get more insight into the evidence–test set loss relation for different subsets of the models.

### 6.1 tanh and ReLU models

We first focus on the models with no variable hyperparameters in Figure 3. Figure 4 shows that the models with ReLU activations consistently outperform the tanh models, and almost always provide a higher value of $Z$. This is quite a surprising result since a-priori, one may expect the opposite to occur, since tanh is a more non-linear function, one may expect it to perform well for the small networks considered here. In traditional neural network training, two of the key reasons why ReLU is a popular choice are that: 1) its derivative is fast and easy to calculate which is crucial for backward propagation training and 2) the fact that non-positive weights are shut out (their value and derivative are both zero) means that a side-effect of using ReLU is that it provides a form of regularisation, which can be helpful in large networks. Neither of these considerations are applicable to the BNNs we consider here, however, since PolyChord does not use derivative information, and only small networks are considered. The only obvious potential benefit of using ReLU for these BNNs is that the function does not saturate for input values large in magnitude as tanh does. Regardless of why ReLU so consistently outperforms the tanh models, the evidence clearly picks up on its superiority.
Figure 3: Log Bayesian evidence versus test loss values averaged over the ten different data randomisations, for the BNNs with tanh activation functions (blue), ReLU activations (orange), and variable hyperparameter models with tanh activations (green).

Figure 4: Detail of Figure 3 focussing on tanh (blue points) or ReLU (orange points) activation functions, and no variable hyperparameters.
Figure 5 shows the evidence versus the BNN dimensionality (top plot) and test loss versus the BNN dimensionality (bottom plot), for the fixed hyperparameter BNNs. The evidence seems to behave similarly (minus an offset) for both activations as a function of the model dimensionality. Looking at the results for the two different activations separately, referring back to Figure 4 there is no trend between evidence and test set performance for the different sized networks. From Figure 5 it appears that the ReLU models get better as they grow in size, but no such trend exists for the tanh models.

6.2 Variable hyperparameter models

We now focus on the variable hyperparameter models in Figure 3. There is a marked increase in performance (i.e. decrease in test loss) in comparison with the fixed hyperparameter models, and a corresponding increase in Bayesian evidence. As can be seen in Figure 2, this is predominantly driven by the fact that the likelihood variance $\sigma$ preferred by the data is significantly different from the value chosen (unity) when the hyperparameters are fixed.

Figure 6 shows the evidence versus test loss, and Figure 7 shows the evidence versus the BNN dimensionality (top plot), and test loss versus the BNN dimensionality (bottom plot), for models with different granularities of variable prior hyperparameters. Looking at all the variable hyperparameter models as one (i.e. ignoring the colour-coding in Figure 7), the correlation and symmetries remain, but the peaks/troughs are much less apparent in comparison with the same patterns within a colour class.

Single random variable hyperparameter runs

Focusing first on the single random variable hyperparameter models, we see some correlation between Bayesian evidence and test set performance. We also see a remarkable symmetry between the log evidence as a function of NN dimensionality and the test set performance as a function of the same parameter. Furthermore, there appears to be a peak in the Bayesian evidence, which is fast to increase (with the BNN dimension), but relatively slow in decline. This is a well-known trend in model selection, called a Bayesian cliff, and was observed in MacKay (1992c). The corresponding dip in test set performance is less well-pronounced, but still arguably there and could be interpreted as a plateau.

Layer granularity runs

For the layer random variable hyperparameter models, more of a correlation between test set performance and Bayesian evidence is present. Similar to the single random variable hyperparameter case, the $Z$–BNN dimensionality and test set performance–BNN dimensionality symmetry also appears. The corresponding peaks and troughs are also there, but are less clear-cut (more plateau-like). For layer granularity models the test set performance was better than the equivalent models with single granularity in most cases. The evidence values were also higher for the former in general, indicating that it correctly captured the superior performance.
Figure 5: Log Bayesian evidence (top plot) and test loss values (bottom plot) versus NN model dimensionality, for all individual BNNs with tanh (blue points) or ReLU (orange points) activation functions, with fixed hyperparameters. Note that the mapping between architecture and dimensionality is not one-to-one, as some models by chance have the same dimensionality. In these cases, the evidence may still be used to select between architectures of the same dimensionality. For clarity, reading right-to-left and then top-to-bottom the network architectures are (2) (2,2) (2,2,2) (2,2,2,2), (4), (4,4), (4,4,4), (4,4,4,4), (8).
Input size random variable hyperparameter runs

Surprisingly, the input size random variable hyperparameter models consistently underperform the layer models (of the same model dimensionality). Since this model is more granular than the layer models, one may expect it to perform at least as well. Thus, one can only attribute this to the (more) complex parameter space not being explored as well, or, the model overfitting to the training data. Nevertheless, the input size results show a good correlation between test performance and evidence, a strong symmetry in evidence and test performance when plotted against BNN dimensionality, and arguably, the most well-formed peaks/troughs in the corresponding Figures.

Further investigation into the layer and input size granularity performances is warranted, since the latter has been shown to do better in the past (Neal, 2012) and other recent works (Javid et al., 2020). We first checked the training set performance, which was similar between the two granularities in terms of mean squared errors (see Table 1). Next we looked at the values of the prior standard deviations ($\alpha 1/\sqrt{\lambda_r}$ where $\lambda_r$ is the regularisation constant in the traditional terminology). In all but one case, the input size standard deviations were smaller than those obtained from the layer granularity models, suggesting the former is a more regularised model. This is perhaps surprising, as the underperformance on the test data of the input size models would suggest the training data is being overfit. The most plausible explanation therefore is that the input size models are on average shutting off nodes important in making accurate predictions on the test data, and more so than the layer granularity models, while doing a similarly good job on the training data.

As mentioned previously, MacKay (1992c) found that when a correlation between $Z$
Figure 7: Log Bayesian evidence (top plot) and test loss values (bottom plot) versus BNN model dimensionality focusing on the cases with variable hyperparameters with single (blue points), layer (orange points), or input size granularity. Note that, as in Figure 5, model dimensionality is the total number of model parameters, including network and prior hyperparameters.
and test performance was not present, then the model could be improved in some way, and that the improved model exhibited this correlation more. One could argue the same trend has appeared in this work. In the previous section, we considered models with no variable hyperparameters, which for a given activation, showed no trend between \( Z \) and test performance for different sized models. However once variable hyperparameters were included, correlations appeared, and so did an overall increase in the performance of the models.

**Analysis of results by model size**

Looking at different cross sections of the set of models by aggregating the models in terms of their size, we first look at all the \( n \)-node architectures, i.e. all the models which have \( n \)-nodes in their hidden layers. For the two-node models, Figure 8 shows that the correlation between \( Z \) and test performance is present for this subset, with clear modes corresponding to tanh models with fixed hyperparameters, ReLU models, and the variable hyperparameter models. The top and bottom plots of Figure 9 shows that \( Z \) and test set performance symmetries are also present, but the peaks/troughs/plateaus are more convoluted than the ones seen in Section 6.2. The four and eight-node architectures show similar results.

When considering models grouped together by how many hidden layers they contain (one, two, three or four), the same overall patterns seem to exist, but there seems to be mainly two modes of separation: fixed hyperparameter versus variable hyperparameter models, as was the case in Figure 3. For one layer, all but the one-layer model with eight nodes show prominent peaks/troughs associated with Occam’s hill and test set

| names   | \( E_{tr,1} \) | \( E_{tr,i} \) | \( n_l \) | \( n_i \) | \( \sigma_{p,1} \) | \( \sigma_{p,i} \) error | \( \sigma_{p,1} \) | \( \sigma_{p,i} \) error |
|---------|---------------|---------------|---------|---------|----------------|-----------------|----------------|----------------|
| (2)     | 0.1120        | 0.1270        | 4       | 17      | 2.696         | 1.967           | 1.650           | 1.991           |
| (4)     | 0.0508        | 0.0593        | 4       | 19      | 4.763         | 1.036           | 2.682           | 0.432           |
| (8)     | 0.0326        | 0.0312        | 4       | 23      | 4.698         | 0.740           | 3.314           | 0.758           |
| (2, 2)  | 0.0926        | 0.0901        | 6       | 20      | 2.793         | 2.818           | 1.829           | 0.583           |
| (4, 4)  | 0.0459        | 0.0502        | 6       | 24      | 2.529         | 0.677           | 1.846           | 0.495           |
| (2, 2, 2) | 0.0946    | 0.0911        | 8       | 23      | 3.103         | 2.366           | 1.836           | 0.507           |
| (4, 4, 4) | 0.0476    | 0.0512        | 8       | 29      | 2.465         | 0.541           | 2.566           | 1.262           |
| (2, 2, 2, 2) | 0.0924 | 0.0895        | 10      | 26      | 3.470         | 1.947           | 1.910           | 0.623           |
| (4, 4, 4, 4) | 0.0493 | 0.0525        | 10      | 34      | 5.551         | 2.113           | 1.787           | 0.318           |

Table 1: Comparison of layer and input size granularity random variable prior hyperparameter models on training data. The training losses (\( E_{tr,\cdot} \)) are averages obtained over the ten different data randomisations. \( \sigma_p \) denotes the standard deviation of the prior (i.e. the variable prior hyperparameters), and hats denote their average values. The quoted errors are the standard deviations of the mean estimates. The numbers in parentheses in the left-hand column specify the number of nodes in each hidden layer of the network, while \( n_l \) and \( n_i \) denote the number of variable hyperparameters in layer and input size models respectively.
Compromise-free Bayesian neural networks

performance versus BNN dimensionality. The same can be said for the 2, 3, and 4 layer models, but said relations are less clear-cut.

7 Results for ensembled models

As mentioned in Section 3.2, the posterior distributions of different models can be combined (ensamed) such that one gets a posterior corresponding to a model in which one considers all of these models at once when training the data. We assign a uniform prior over models for all combined analyses, showing no preference for any particular model a-priori. We obtain the evidences and predictions associated with these combined models and analyse the $Z - E$ relation for these ensembles, and compare the quality of their predictions relative to the individual models. We consider many different combinations of models, across various cross sections of the wide array of models used in the individual analyses. Table 3 gives a full breakdown of the different ensembles considered, but to summarise, the combinations of models we consider broadly cover the following cross sections (and their supersets): different sized models with either the same or different activation functions; models with the same variable hyperparameter granularity; models with the same number of nodes per layer; and models with the same number of layers.

The $Z$-test set performance trends were very similar to the results of the models which the respective combined runs comprised of, as can be seen in Figure 10 which shows the $Z$ and test set losses for both individual and combined models. The evidences of the combined runs generally lie in the middle of values of the individual ones which
Figure 9: Log Bayesian evidence versus NN dimensionality (top plot) and test loss versus NN dimensionality (bottom plot), for all individual BNNs with two node-wide layers (blue points), four node-wide layers (orange points), or eight node-wide layers (green points).
is to be expected, since the combined evidences are just a linear combination of the
evidences of the individual runs, weighted by their priors. Test set performance was also
very similar on average, and so the separation in the $Z$–test loss plane corresponding to
varying and fixed hyperparameter models persist with the combined models. Though
the performances were on average very similar, the lowest test loss was obtained using a
combined model; the ensemble which combined all models trained with layer granularity
hyperparameters achieved a test loss of 0.1732, while the best individual model was the
$(l_1, l_2) = (4, 4)$ model with layer hyperparameter granularity which obtained a test loss
value of 0.1791.

Figure 11 compares various combined models. The top plot shows all models with
fixed prior hyperparameters, while the bottom plot compares models with all three
different granularities of hyperparameters. The different ensembles considered for each
type are listed in Table 3.

8 Comparison with traditional network training

As a means of providing a baseline for the previous analyses, we trained the same net-
work architectures (with tanh or ReLU activations) using traditional back propagation
methods (Rumelhart et al., 1985; MacKay, 2003) in their simplest form (i.e. no regu-
larisation, no hyperparameter tuning). We refer to these networks as traditional neural
networks (TNNs). The networks were trained for 1000 epochs each using the Adam
optimisation algorithm (Kingma and Ba, 2014), on the same training/test data splits
as considered for the BNNs. Once again we used 10 different data randomisations and
Figure 11: Top plot: Log Bayesian evidence versus test loss values averaged over the ten different data randomisations, for all the tanh activation function BNNs combined (blue points), the ReLU activation function BNNs combined (orange points), and combinations of BNNs with both types of activation function (green points). For each of these, the seven combinations (datapoints) are as follows: all one layer models combined, all two layer models combined, all three layer models combined, all four layer models combined, all two node-wide models combined, all four node-wide models combined, and all models combined. Bottom plot: Same as above but for single granularity random variable prior hyperparameter BNNs combined (blue points), layer granularity random variable prior hyperparameter BNNs combined (orange points), and input size granularity random variable prior hyperparameter BNNs combined (green points).
Compromise-free Bayesian neural networks averaged the results. We note in passing that the maximum likelihood parameters found by the two different methods of training the networks showed no correlation, suggesting the optimisation and sampling methods are exploring the parameter spaces in quite different ways. For networks with tanh activations, six of the TNN test set estimates estimates were inferior to the no variable hyperparameter BNN estimates, while four were superior. When variable hyperparameters were used, all 10 BNNs performed better than the TNN equivalent architectures. For ReLU networks the simplest BNNs outperformed the TNNs in all 10 cases, and the tanh TNNs outperformed the ReLU TNNs in all but one case, again emphasising the significance of the ReLU BNNs outperforming their tanh equivalents. Table 4 in Appendix B summarises the performance of the TNNs.

9 Discussion & conclusions

In this paper we applied compromise-free Bayesian neural networks (BNNs) to the Boston housing dataset in order to explore the relationship between out-of-sample performance and the Bayesian evidence as first studied in MacKay (1992c), and how combining the predictions of different networks in a statistically sound way affects their performance. To obtain fully numerical posterior samples and evidences associated with the training data and the networks, we used the nested sampling algorithm PolyChord (Handley et al., 2015a,b), to train models with up to 156 trainable parameters. We considered a wide variety of models in our analysis: networks with either tanh or ReLU activation functions; networks with between zero and four hidden layers; and models trained using hierarchical Bayesian inference, i.e. models where the prior and likelihood standard deviations are modelled as random variables, as in MacKay (1992c); Neal (2012); Javid et al. (2020). For the hierarchical priors, three different variants were considered, corresponding to three different levels of granularity on the priors: single, layer (both used in MacKay, 1992c) and input size (similar to the automatic relevance determination method in Neal, 2012). Single granularity has one hyperparameter controlling the standard deviations on the priors of all weights and biases in the network. Layer granularity has two hyperparameters per layer (one for the bias, one for the weights in each layer). For input size granularity models, the number of hyperparameters for each layer depends on the number of inputs to that layer. Following Neal (2012) we used zero mean Gaussian distributions with an ordering enforced to prevent weight space degeneracy (Goodfellow et al., 2016) (as discussed in Handley et al., 2019; Buscicchio et al., 2019) for the priors and Gamma distributions for the hyperpriors. For layer and input size random variable hyperparameter models we scaled the Gamma distribution hyperparameters so that the prior over functions converges to Gaussian processes as discussed in Neal (2012).

In total we trained 49 different models, and for each we performed 10 different randomisations of the training–test split (in each case 50% of the data is used for training and the remaining 50% is used for testing), and took averages of these 10 results. From inspecting the mean squared error of the BNN estimates on the test data, the Bayesian evidences, and the size of the parameter space dimensionalities for the different models, we found the following:
• Looking at all of the models with at least one hidden layer collectively, in the test set performance–evidence plane, there appears to be two distinct clusters: models which used fixed hyperparameters, and models which used variable hyperparameters. In all cases for a given architecture, the latter had higher evidence values and better performance, adhering to the trend found in MacKay (1992c).

• For models with fixed hyperparameters, the models which used the ReLU activation function consistently outperformed the equivalent architectures with tanh activations. One may find this result rather surprising, since tanh is a more non-linear function, one may expect it to perform relatively well for the small models considered here.

• Considering models with single random variable prior hyperparameters, there is a definite positive correlation between test set accuracy and Bayesian evidence.

• For the same subset of models, there appears to be a Bayesian cliff present when looking at the evidence as a function of parameter space dimensionality (as found in MacKay, 1992c), as well as a striking symmetry between the evidence–dimensionality and the test set performance–dimensionality planes.

• A stronger correlation between performance and evidence is present for the layer random variable prior hyperparameter models. The symmetry mentioned in the last item is also present, but in this case the Bayesian cliff is more of a plateau. The similar can be said for the models with input size granularity.

• For layer granularity models the test set performance was better than the equivalent models with single granularity in most cases. The evidence values were also higher for the former in general, indicating that it correctly captured the superior performance. But perhaps surprisingly, the input size granularity models consistently underperform the layer granularity models, in contrast to Neal (2012); Javid et al. (2020). Further investigation into the training and test set losses suggests that the input size models are on average shutting off nodes important for the test data, more than the layer granularity models are, while performing similarly well on the training data.

• The jump in performance and evidence, and increase in correlation between the two, associated with switching from fixed to variable hyperparameters was also found in MacKay (1992c), but in effectively switching from single to layer granularity models. MacKay attributed this to the idea that when a correlation between evidence and performance was not present, then the model could be improved in some way, and that the improved model showed this correlation more.

We next combined the predictions of an ensemble of BNNs by considering the posterior distribution over the corresponding models, parameterised by a categorical variable representing a given BNN. These posteriors are obtained from the samples of the individual model posteriors which are re-weighted according to the evidences associated with a given run, and the ensemble prior. For simplicity we assume a uniform prior over all the models, and we consider a wide array of different ensembles of these BNNs,
including: ensembles of models with the same variable hyperparameter granularity; en-
sembles of models with the same number of nodes per layer; and ensembles of models
with the same number of layers. Looking at the predictions and the evidences associated
with the ensembles we found the following:

- The evidence–test set performance relations were very much the same as those
  found for the individual model results.

- The best performance on the test set data overall was obtained with an ensemble
  of models: the combination of all models with layer granularity random variable
  hyperparameters achieved a test loss of 0.1732. The best performing individual
  model obtained a loss of 0.1791.

Finally, we trained the same network architectures and activation functions using
traditional backward propagation techniques (Rumelhart et al., 1985), which we refer
to as traditional neural networks (TNNs). We considered the most basic form of TNNs,
in the sense that no regularisation or hyperparameter tuning was used, to give a lower
bound on performance for these networks. Looking at the performance of these TNNs
and comparing with the BNN results we found:

- The BNNs with fixed hyperparameters outperformed the corresponding TNNs
  with tanh in the majority of cases, while the BNNs with variable hyperparameters
  performed superior to the TNNs in all cases.

- For networks with ReLU activations, the fixed hyperparameter BNNs outper-
  formed the TNNs in all cases. Furthermore the tanh TNNs outperformed the
  ReLU TNNs in all but one case. Both of these results emphasise the significance
  of the superior performance of BNNs with ReLU models relative to those with
  tanh activations.

Final comments

A compromise-free approach is only ever intended as an initial step in a wider analysis.
The purpose of solving the full numerical problem without approximation is twofold:
First to see how far one can get with current computing resources, and hence to fore-
cast how things may scale both now and in the future with more compute, time or
money. Second, to use the full solution to examine what approximations may be safely
made in practice in order to create a more performant algorithm whilst preserving the
characteristics of the correct, fully-numerical answer. It is hoped that this work will
be a springboard and inspiration for a profitable line of research into Bayesian neural
networks.

Acknowledgements

This work was performed using resources provided by the Cambridge Service for Data
Driven Discovery (CSD3) operated by the University of Cambridge Research Computing
Service, provided by Dell EMC and Intel using Tier-2 funding from the Engineering and Physical Sciences Research Council (capital grant EP/P020259/1), and DiRAC funding from the Science and Technology Facilities Council. K.J. was supported by an STFC Impact Acceleration Account 2018 grant. W.H. was supported by a Gonville & Caius College Research fellowship and STFC IPS grant number G102229.

References
Andrieu, C., de Freitas, N., Doucet, A., and Jordan, M. I. (2003). “An introduction to MCMC for machine learning.” *Machine learning*, 50(1-2): 5–43. 2
Barber, D. and Bishop, C. M. (1998). “Ensemble learning in Bayesian neural networks.” *Nato ASI Series F Computer and Systems Sciences*, 168: 215–238. 2
Buntine, W. L. and Weigend, A. S. (1991). “Bayesian back-propagation.” *Complex systems*, 5(6): 603–643. 1
Buscicchio, R., Roebber, E., Goldstein, J. M., and Moore, C. J. (2019). “The Label Switching Problem in Bayesian Analysis for Gravitational Wave Astronomy.” *arXiv preprint arXiv:1907.11631*. 9, 24
Chua, A. J., Hee, S., Handley, W. J., Higson, E., Moore, C. J., Gair, J. R., Hobson, M. P., and Lasenby, A. N. (2018). “Towards a framework for testing general relativity with extreme-mass-ratio-inspiral observations.” *Monthly Notices of the Royal Astronomical Society*, 478(1): 28–40. 8
de Freitas, J. d., Niranjan, M., Gee, A. H., and Doucet, A. (2000). “Sequential Monte Carlo methods to train neural network models.” *Neural computation*, 12(4): 955–993. 2
de Freitas, J. F. G. (2003). “Bayesian methods for neural networks.” Ph.D. thesis, University of Cambridge. 2, 3
Denker, J., Schwartz, D., Wittner, B., Solla, S., Howard, R., Jackel, L., and Hopfield, J. (1987). “Large automatic learning, rule extraction, and generalization.” *Complex systems*, 1(5): 877–922. 1
Denker, J. S. and Lecun, Y. (1991). “Transforming neural-net output levels to probability distributions.” In *Advances in neural information processing systems*, 853–859. 2
Efron, B. (1979). “Computers and the theory of statistics: thinking the unthinkable.” *SIAM review*, 21(4): 460–480. 4
Feroz, F., Hobson, M., and Bridges, M. (2009). “MultiNest: an efficient and robust Bayesian inference tool for cosmology and particle physics.” *Monthly Notices of the Royal Astronomical Society*, 398(4): 1601–1614. 11
Gal, Y. and Ghahramani, Z. (2016a). “Dropout as a bayesian approximation: Representing model uncertainty in deep learning.” In *international conference on machine learning*, 1050–1059. 2
Compromise-free Bayesian neural networks

— (2016b). “A theoretically grounded application of dropout in recurrent neural networks.” In Advances in neural information processing systems, 1019–1027.

Ghahramani, Z. (2015). “Probabilistic machine learning and artificial intelligence.” Nature, 521(7553): 452.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). Deep learning. MIT press.

Graves, A. (2011). “Practical variational inference for neural networks.” In Advances in neural information processing systems, 2348–2356.

Handley, W. (2019). “anesthetic: nested sampling visualisation.” The Journal of Open Source Software, 4: 1414.

Handley, W., Hobson, M., and Lasenby, A. (2015a). “PolyChord: nested sampling for cosmology.” Monthly Notices of the Royal Astronomical Society: Letters, 450(1): L61–L65.

— (2015b). “POLYCHORD: next-generation nested sampling.” Monthly Notices of the Royal Astronomical Society, 453(4): 4384–4398.

Handley, W. and Millea, M. (2019). “Maximum-Entropy Priors with Derived Parameters in a Specified Distribution.” Entropy, 21(3): 272.

Handley, W. J., Lasenby, A. N., Peiris, H. V., and Hobson, M. P. (2019). “Bayesian inflationary reconstructions from Planck 2018 data.” arXiv preprint arXiv:1908.00906.

Hee, S., Handley, W., Hobson, M. P., and Lasenby, A. N. (2015). “Bayesian model selection without evidences: application to the dark energy equation-of-state.” Monthly Notices of the Royal Astronomical Society, 455(3): 2461–2473.

Higson, E., Handley, W., Hobson, M., and Lasenby, A. (2018). “Bayesian sparse reconstruction: a brute-force approach to astronomical imaging and machine learning.” Monthly Notices of the Royal Astronomical Society, 483(4): 4828–4846.

Hinton, G. and Van Camp, D. (1993). “Keeping neural networks simple by minimizing the description length of the weights.” In in Proc. of the 6th Ann. ACM Conf. on Computational Learning Theory. Citeseer.

Javid, K. and Handley, W. (2020). “SuperKam91/bnn: First public release.”

URL https://doi.org/10.5281/zenodo.3758553

Javid, K., Handley, W., Hobson, M., and Lasenby, A. (2020). “TwentyoneFlow: accelerated global 21cm signal emulation with TensorFlow and Bayesian neural networks.” MNRAS (In preparation).

Kendall, A. and Gal, Y. (2017). “What uncertainties do we need in bayesian deep learning for computer vision?” In Advances in neural information processing systems, 5574–5584.

Kingma, D. P. and Ba, J. (2014). “Adam: A method for stochastic optimization.” arXiv preprint arXiv:1412.6980.
Krzywinski, M. and Altman, N. (2013). “Points of significance: Importance of being uncertain.”

Li, K.-C. et al. (1985). “From Stein’s unbiased risk estimates to the method of generalized cross validation.” *The Annals of Statistics*, 13(4): 1352–1377.

MacKay, D. J. (1992a). “Bayesian interpolation.” *Neural computation*, 4(3): 415–447.

— (1992b). “The evidence framework applied to classification networks.” *Neural computation*, 4(5): 720–736.

— (1992c). “A practical Bayesian framework for backpropagation networks.” *Neural computation*, 4(3): 448–472.

— (2003). *Information theory, inference and learning algorithms*. Cambridge university press.

MacKay, D. J. et al. (1994). “Bayesian nonlinear modeling for the prediction competition.” *ASHRAE transactions*, 100(2): 1053–1062.

Neal, R. M. (1992). “Bayesian training of backpropagation networks by the hybrid Monte Carlo method.” Technical report, Citeseer.

— (1993). “Bayesian learning via stochastic dynamics.” In *Advances in neural information processing systems*, 475–482.

— (2012). *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1985). “Learning internal representations by error propagation.” Technical report, California Univ San Diego La Jolla Inst for Cognitive Science.

Skilling, J. (2006). “Nested sampling for general Bayesian computation.” *Bayesian analysis*, 1(4): 833–859.

Snoek, J., Larochelle, H., and Adams, R. P. (2012). “Practical bayesian optimization of machine learning algorithms.” In *Advances in neural information processing systems*, 2951–2959.

Stone, M. (1977). “Asymptotics for and against cross-validation.” *Biometrika*, 29–35.

Tishby, N., Levin, E., and Solla, S. A. (1989). “Consistent inference of probabilities in layered networks: Predictions and generalization.” In *International Joint Conference on Neural Networks*, volume 2, 403–409.
Appendix A: Granularities in context

Implementing hierarchical Bayesian inference

The single and layer granularities were used in MacKay (1992c), where he found that the former lead to a poor correlation between $Z$ and test set performance, when comparing models of different sizes. He argued that this was due to the fact that the input, outputs and hidden units had no reason to take the same scale of values, and thus scaling the weights associated with the different layers by the same factor (hyperparameter) was not the right thing to do. Thus he assigned one hyperparameter to the hidden unit weights, one to the hidden unit bias, and one for the output weights and biases (note this is slightly less granular than our implementation, which assigns separate hyperparameters for the output weights and bias). With this model he finds a much stronger correlation between $Z$ and test set performance, and an overall improvement in the models’ performance. This is an example of the evidence not only being used as a proxy for out-of-sample performance, but also as an indicator that model performance can be improved in some way (though in general it does not give any indication of how to improve the model). The input size granularity takes inspiration from the automatic relevance determination (ARD) methodology introduced by MacKay et al. (1994); Neal (2012). The idea is to block out any inputs to a given layer which are not being used in learning the function mapping which the model represents. This is accomplished by inferring a large value for the precision (small value for the variance) associated with that input. Note that to the authors’ knowledge, an in-depth analysis of the Bayesian evidence when considering granularities such as input size/ARD has not been done previously.

Gaussian processes as a prior over networks

Neal (2012) introduces further insight into prior hyperparameters through his analysis of Gaussian processes and their relation to Bayesian neural networks. Neal finds that in the limit of an infinitely wide neural network, the network prior converges to a Gaussian process when the priors of the network weights are appropriately scaled. Neal shows that to prevent overfitting the data when building an arbitrarily large network, one must scale the variance of the weight priors according to the size of the previous layer (this argument does not apply to the first hidden layer, as the input layer nodes have no such restriction on their contribution to the subsequent layers). This also ensures the prior over functions has a finite variance. Thus in the models we consider with layer or input size variable prior hyperparameters, for layer $i+1$ we scale the scale parameters of all the Gamma (hierarchical) priors for the weights, by the size of the previous hidden layer $l_i$, i.e. $\beta_i \rightarrow \beta_i/l_i$. 
## Appendix B: Tabled results

| names   | test loss | test loss error | log($\mathcal{Z}$) | log($\mathcal{Z}$) error | dimensionality |
|---------|-----------|-----------------|---------------------|--------------------------|----------------|
| br      | 0.3415    | 0.0070          | -294.16             | 0.11                     | 14             |
| sh sv   | 0.3416    | 0.0036          | -201.16             | 0.14                     | 16             |
| lh sv   | 0.3418    | 0.0036          | -202.12             | 0.14                     | 17             |
| ih sv   | 0.3416    | 0.0036          | -209.32             | 0.15                     | 28             |
| (2)     | 0.2746    | 0.0088          | -286.75             | 0.12                     | 31             |
| r (2)   | 0.2526    | 0.0080          | -278.73             | 0.13                     | 31             |
| sh sv (2) | 0.2181  | 0.0059          | -149.09             | 0.70                     | 33             |
| lh sv (2) | 0.2090  | 0.0049          | -141.35             | 0.33                     | 36             |
| ih sv (2) | 0.2214  | 0.0045          | -150.62             | 0.39                     | 49             |
| (4)     | 0.2711    | 0.0090          | -287.61             | 0.11                     | 61             |
| r (4)   | 0.2336    | 0.0085          | -281.28             | 0.13                     | 61             |
| sh sv (4) | 0.1999  | 0.0074          | -135.05             | 0.75                     | 63             |
| lh sv (4) | 0.1982  | 0.0086          | -121.86             | 1.06                     | 66             |
| ih sv (4) | 0.2021  | 0.0073          | -124.91             | 1.29                     | 81             |
| (8)     | 0.2688    | 0.0092          | -291.26             | 0.11                     | 121            |
| r (8)   | 0.2216    | 0.0086          | -287.74             | 0.14                     | 121            |
| sh sv (8) | 0.2082  | 0.0068          | -127.67             | 2.57                     | 123            |
| lh sv (8) | 0.1864  | 0.0069          | -108.84             | 1.16                     | 126            |
| ih sv (8) | 0.1913  | 0.0066          | -104.99             | 1.71                     | 145            |
| (2, 2)  | 0.2773    | 0.0095          | -289.02             | 0.13                     | 37             |
| r (2, 2) | 0.2509    | 0.0085          | -279.56             | 0.12                     | 37             |
| sh sv (2, 2) | 0.1983 | 0.0072          | -138.37             | 0.70                     | 39             |
| lh sv (2, 2) | 0.2042  | 0.0063          | -130.96             | 0.48                     | 44             |
| ih sv (2, 2) | 0.2152  | 0.0067          | -139.92             | 0.75                     | 58             |
| (4, 4)  | 0.2722    | 0.0101          | -289.18             | 0.10                     | 81             |
| r (4, 4) | 0.2311    | 0.0098          | -281.92             | 0.13                     | 81             |
| sh sv (4, 4) | 0.1877  | 0.0084          | -116.07             | 1.23                     | 83             |
| lh sv (4, 4) | 0.1791  | 0.0081          | -115.37             | 1.74                     | 88             |
| ih sv (4, 4) | 0.2033  | 0.0078          | -115.78             | 1.38                     | 106            |
| (2, 2, 2) | 0.2779  | 0.0096          | -290.47             | 0.13                     | 43             |
| r (2, 2, 2) | 0.2558  | 0.0086          | -280.53             | 0.15                     | 43             |
| sh sv (2, 2, 2) | 0.2032 | 0.0077          | -137.25             | 0.76                     | 45             |
| lh sv (2, 2, 2) | 0.1995  | 0.0056          | -129.39             | 0.57                     | 52             |
| ih sv (2, 2, 2) | 0.2078  | 0.0061          | -141.26             | 0.92                     | 67             |
| (4, 4, 4) | 0.2730  | 0.0104          | -289.93             | 0.13                     | 101            |
| r (4, 4, 4) | 0.2335  | 0.0100          | -282.17             | 0.13                     | 101            |
| sh sv (4, 4, 4) | 0.2108 | 0.0081          | -116.89             | 1.57                     | 103            |
| lh sv (4, 4, 4) | 0.1801  | 0.0068          | -116.27             | 2.08                     | 110            |
| ih sv (4, 4, 4) | 0.1923  | 0.0080          | -117.12             | 1.01                     | 131            |
| (2, 2, 2, 2) | 0.2749  | 0.0094          | -291.49             | 0.13                     | 49             |
| r (2, 2, 2, 2) | 0.2589  | 0.0084          | -281.74             | 0.13                     | 49             |
Compromise-free Bayesian neural networks

Table 2: Test loss and evidence comparison for all the individual BNNs. The losses and evidences ($Z$) are averages obtained over the ten different data randomisations. The quoted errors are the standard deviations of the mean estimates. The key for the model names is as follows: r denotes ReLU activation functions were used in the hidden layers of the network (tanh otherwise). sh denotes a single random variable prior hyperparameter model, lh denotes layer granularity, while ih represents input size granularity. sv means the likelihood hyperparameter (variance) was also treated as variable. br (i.e. the first row) denotes Bayesian linear regression i.e. no hidden layers and fixed hyperparameters. The numbers in parentheses denote the number of nodes per hidden layer.

| names      | test loss | test loss error | log($Z$) | log($Z$) error | dimensionality |
|------------|-----------|-----------------|----------|----------------|----------------|
| sh sv (2, 2, 2) | 0.2056    | 0.0068          | -137.83  | 0.91           | 51             |
| lh sv (2, 2, 2) | 0.1963    | 0.0057          | -132.31  | 0.47           | 60             |
| ih sv (2, 2, 2) | 0.2103    | 0.0068          | -141.68  | 0.73           | 76             |
| (4, 4, 4, 4) | 0.2724    | 0.0105          | -290.34  | 0.21           | 121            |
| r (4, 4, 4, 4) | 0.2387    | 0.0098          | -282.29  | 0.14           | 121            |
| sh sv (4, 4, 4, 4) | 0.2158    | 0.0077          | -118.75  | 1.11           | 123            |
| lh sv (4, 4, 4, 4) | 0.2035    | 0.0074          | -120.24  | 2.02           | 132            |
| ih sv (4, 4, 4, 4) | 0.2197    | 0.0073          | -125.18  | 1.93           | 156            |

| names      | test loss | test loss error | log($Z$) | log($Z$) error |
|------------|-----------|-----------------|----------|----------------|
| 1l         | 0.2731    | 0.0089          | -287.49  | 0.09           |
| 2l         | 0.2745    | 0.0098          | -289.09  | 0.08           |
| 3l         | 0.2742    | 0.0101          | -290.16  | 0.10           |
| 4l         | 0.2718    | 0.0101          | -290.76  | 0.16           |
| 2n         | 0.2739    | 0.0091          | -288.01  | 0.10           |
| 4n         | 0.2706    | 0.0096          | -288.68  | 0.08           |
| an         | 0.2724    | 0.0093          | -288.40  | 0.07           |
| 1l r       | 0.2491    | 0.0081          | -279.75  | 0.12           |
| 2l r       | 0.2459    | 0.0089          | -280.17  | 0.11           |
| 3l r       | 0.2452    | 0.0093          | -281.05  | 0.13           |
| 4l r       | 0.2479    | 0.0092          | -281.97  | 0.10           |
| 2n r       | 0.2511    | 0.0083          | -279.61  | 0.09           |
| 4n r       | 0.2325    | 0.0094          | -281.83  | 0.07           |
| an r       | 0.2463    | 0.0086          | -280.32  | 0.08           |
| 1l aa      | 0.2491    | 0.0081          | -280.44  | 0.12           |
| 2l aa      | 0.2459    | 0.0089          | -280.86  | 0.11           |
| 3l aa      | 0.2452    | 0.0093          | -281.74  | 0.13           |
| 4l aa      | 0.2479    | 0.0092          | -282.66  | 0.10           |
| Architectures | (2)  | (4)  | (8)  | (2, 2) | (4, 4) | (2, 2, 2) | (4, 4, 4) | (2, 2, 2, 2) | (4, 4, 4, 4) |
|--------------|------|------|------|--------|--------|-----------|-----------|-------------|-------------|
| tanh loss    | 0.2892 | 0.2942 | 0.2374 | 0.2776 | 0.2511 | 0.2884 | 0.2284 | 0.3085 | 0.2363 |
| ReLU loss    | 0.4025 | 0.2782 | 0.2609 | 0.5472 | 0.2957 | 0.7637 | 0.2877 | 0.7375 | 0.3068 |

Table 4: Test losses obtained from networks trained with traditional maximum likelihood, backward propagation optimisation techniques.

Table 3: Test loss and evidence comparisons for all the combined BNNs. The key for the model names is as follows (including the notation in Table 2: Xl denotes the combination of all networks with X layers. Xn models combine all networks with X nodes in the hidden layers. Note an denotes all number of nodes i.e. all model sizes combined, and aa denotes all activation functions, i.e. combining models with either tanh or ReLU activations in their hidden layers.)

| names     | test loss | test loss error | log(Z) | log(Z) error |
|-----------|-----------|-----------------|--------|--------------|
| 2n aa     | 0.2523    | 0.0082          | -280.61| 0.11         |
| 4n aa     | 0.2326    | 0.0094          | -282.52| 0.07         |
| an aa     | 0.2456    | 0.0086          | -281.28| 0.09         |
| 1l sh sv  | 0.2104    | 0.0072          | -128.76| 2.57         |
| 2l sh sv  | 0.1877    | 0.0084          | -116.77| 1.23         |
| 3l sh sv  | 0.2108    | 0.0081          | -117.58| 1.57         |
| 4l sh sv  | 0.2158    | 0.0077          | -119.44| 1.11         |
| 2n sh sv  | 0.1985    | 0.0074          | -137.83| 0.55         |
| 4n sh sv  | 0.2140    | 0.0076          | -116.81| 1.16         |
| an sh sv  | 0.2116    | 0.0075          | -117.62| 1.16         |
| 1l lh sv  | 0.1862    | 0.0070          | -109.94| 1.16         |
| 2l lh sv  | 0.1791    | 0.0081          | -116.07| 1.74         |
| 3l lh sv  | 0.1801    | 0.0068          | -116.96| 2.08         |
| 4l lh sv  | 0.2035    | 0.0074          | -120.93| 2.02         |
| 2n lh sv  | 0.2228    | 0.0111          | -130.51| 0.47         |
| 4n lh sv  | 0.1797    | 0.0069          | -116.19| 1.75         |
| an lh sv  | 0.1732    | 0.0075          | -111.03| 1.16         |
| 1l ih sv  | 0.1913    | 0.0066          | -106.09| 1.71         |
| 2l ih sv  | 0.2033    | 0.0078          | -116.47| 1.38         |
| 3l ih sv  | 0.1923    | 0.0080          | -117.82| 1.01         |
| 4l ih sv  | 0.2197    | 0.0073          | -125.87| 1.93         |
| 2n ih sv  | 0.2074    | 0.0073          | -140.81| 0.59         |
| 4n ih sv  | 0.2052    | 0.0070          | -116.88| 1.28         |
| an ih sv  | 0.2012    | 0.0064          | -107.19| 1.71         |