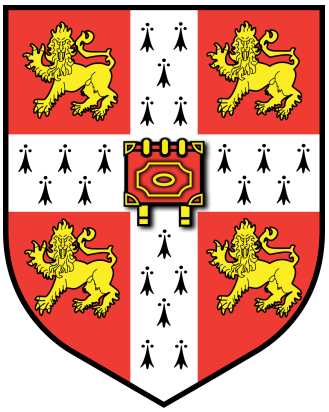


University of Cambridge

MLMI MPhil 2021-22

MLMI 4 - Advanced Machine Learning - Group 7

Weight Uncertainty in Neural Networks

Authors:	*CRSid:*
Alan Clark	ajc348
Max Bronckers	mojb2
Yufan Wang	yw575

Word Count: 4,991

March 31, 2022

Contents

1 Introduction **2**

2 Theory **3**
2.1 Neural Network Point Estimates of Weights 3
2.2 Probability Distributions over Weights 3
2.3 Algorithm 5
2.4 Priors 5
2.5 Local Reparameterisation Trick 6
2.6 Log Pointwise Predictive Density 7
2.7 Contextual Bandits 8
2.8 Adversarial Examples 8

3 Implementation **8**

4 Experiments **9**
4.1 Regression 9
4.2 MNIST Classification 14
4.3 Contextual Bandits 19

5 Conclusion **22**

6 References **23**

1 Introduction

The standard approach to training Deep Neural networks (DNNs) involves learning point estimates of weights. As exemplified in Figure 1, this tends to lead networks to overfit to training data, and exhibit overconfidence in regions of the input space not seen during training. A further issue is one of robustness: as discussed in [1], applying small perturbations to the input causes networks to misclassify images, whilst remaining overconfident. There are various techniques which can be applied during training to combat these issues, including early stopping, weight decay and dropout. As part of our analysis, we trained both regular DNNs and DNNs with dropout as baselines for comparisons.

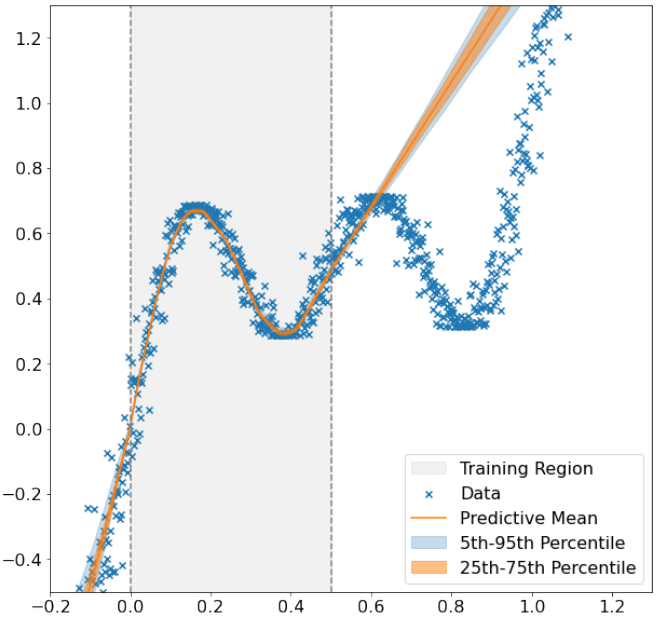


Figure 1: Inference performed using an ensemble of 25 feed-forward DNNs trained using data provided within the grey shaded region. The mean prediction and interquartile ranges are shown. The collection of models shows very limited variability in their predictions.

In their 2015 paper, Blundell, Cornebise, Kavukcuoglu, *et al.* [2] introduce the *Bayes by Backprop* algorithm for performing Bayesian inference on the weights of a Neural Network. Rather than point estimates, posterior distributions are learned for the weights. The goal of training is thus to learn distributions that make correct and confident decisions in regions of input space observed during training, whilst expressing appropriate uncertainty in unfamiliar regions. The authors suggest three motivations for the introduction of uncertainty in the weights:

1. Regularisation via a compression cost on the weights; achieved through the choice of prior distribution
2. Richer representations and predictions from cheap model averaging
3. Automatic exploration in simple reinforcement learning problems

In this work, we replicated all of the experiments performed by the original paper's authors to verify that their approach achieved the above goals. We additionally extended their experiments to assess whether the learned models showed superior robustness to *adversarial examples*[3]. We expected this to be the case, given that regularisation and model ensembling should reduce the propensity to overfit to training data. Further, we investigated the *local reparameterisation trick*, proposed by Kingma, Salimans, and Welling [3], which is purported to aid convergence. A summary of the pertinent background theory is provided in Section 2, and our implementation is detailed in

Section 3. We then describe the experiments conducted and discuss the results in Section 4, with reference to the results obtained by Blundell, *et al.*.

We chose to replicate this paper as we were interested in learning how Bayesian Neural Networks express uncertainty. Many people view DNNs as magic black boxes whose results are to be believed with little critical evaluation, in-part due to networks' confidence in their predictions. More experienced practitioners are aware of their pitfalls, and having a reliable method of obtaining sensible uncertainties in predictions adds a valuable tool to our toolbox. We wanted to gain hands-on experience of implementing BNNs on typical tasks, and perform an unbiased evaluation of their proclaimed benefits and drawbacks.

2 Theory

In this section we provide an overview of the theory necessary to appreciate the description and discussion of our implementation and the experiments conducted. This primarily summarises what is presented by Blundell, *et al.* in [2], and additionally describes the *local reparameterisation trick* and generation of *adversarial examples*.

2.1 Neural Network Point Estimates of Weights

Neural Networks can be thought of as probabilistic models $P(\mathbf{y}|\mathbf{x}, \mathbf{w})$ which assign probabilities to each possible output $\mathbf{y} \in \mathcal{Y}$ given input $\mathbf{x} \in \mathbb{R}^n$ and a learned set of weights $\mathbf{w} \in \mathbb{R}^d$. In standard frequentist approaches a point estimate for the weights, $\hat{\mathbf{w}}$, is learned and predictions for input $\hat{\mathbf{x}}$ determined as $\hat{\mathbf{y}} = g(\hat{\mathbf{x}}, \hat{\mathbf{w}})$ - the alternating linear and non-linear transformations through the layers of the network.

Equation 1 shows the approach by which the *maximum likelihood estimate (MLE)* of the weights, $\hat{\mathbf{w}}^{\mathbf{MLE}}$, is obtained using a training dataset $\mathcal{D} = (\mathbf{x}_i, \mathbf{y}_i)$. Assuming $P(\mathcal{D}|\mathbf{w})$ is differentiable, the *MLE* point estimate is typically found using gradient descent via back-propagation through the Neural Network.

$$\hat{\mathbf{w}}^{\mathbf{MLE}} = \arg\max_{\mathbf{w}} \log P(\mathcal{D}|\mathbf{w}) = \arg\max_{\mathbf{w}} \sum_i \log P(\mathbf{y}_i|\mathbf{x}_i, \mathbf{w}) \tag{1}$$

By introducing a prior distribution, $P(\mathbf{w})$, over the weights regularisation can be achieved and the *maximum a-posterior (MAP)* estimate of the weights, $\hat{\mathbf{w}}^{\mathbf{MAP}}$, found using Equation 2. Again, back-propagation is used to learn the estimate.

$$\hat{\mathbf{w}}^{\mathbf{MAP}} = \arg\max_{\mathbf{w}} \log P(\mathbf{w}|\mathcal{D}) = \arg\max_{\mathbf{w}} \log P(\mathcal{D}|\mathbf{w}) + \log P(\mathbf{w}) \tag{2}$$

If $P(\mathbf{w})$ is a Gaussian distribution then L2 regularisation is performed, whereas if it is a Laplace distribution then L1 regularisation occurs.

2.2 Probability Distributions over Weights

In the Bayesian approach, rather than learning a point estimate, we seek a posterior distribution over the weights, $P(\mathbf{w}|\mathcal{D})$. At inference time, we marginalise over this distribution to obtain values from the posterior predictive density, $P(\hat{\mathbf{y}}|\hat{\mathbf{x}})$, as shown by Equation 3.

$$P(\hat{\mathbf{y}}|\hat{\mathbf{x}}) = \int_{\mathbf{w}} P(\mathbf{w}|\mathcal{D}) P(\hat{\mathbf{y}}|\hat{\mathbf{x}}, \mathbf{w}) d\mathbf{w} = \mathbb{E}_{P(\mathbf{w}|\mathcal{D})}[P(\hat{\mathbf{y}}|\hat{\mathbf{x}}, \mathbf{w})] \tag{3}$$

The true posterior distribution is typically intractable due to the complex functional form of the Neural Network and the number of parameters involved. Instead, we aim to approximate the posterior by introducing a *variational posterior* parameterised by θ, $q(\mathbf{w}|\theta)$, and learning the

parameters θ^* that minimise the Kullback-Leibler (KL) divergence between the variational and true posteriors, as shown by Equation 4. From the KL divergence, we arrive at the expression in 5, which is referred to as the *variational free energy* or *expected lower bound (ELBO)*, and is denoted by $\mathcal{F}(\mathcal{D}, \theta)$, as shown in Equation 6. It can be seen that the free energy is comprised of a complexity cost term that's dependent on the prior and a likelihood cost term that's dependent on the data. As stated by Blundell, *et al.*: *"The cost embodies a trade-off between satisfying the complexity of the data and satisfying the simplicity of the prior."*

$$\begin{aligned}
\theta^* &= \arg\min_\theta \mathrm{KL}[q(\mathbf{w}|\theta)||P(\mathbf{w}|\mathcal{D})] && (4)\\
&= \arg\min_\theta \int_{\mathbf{w}} q(\mathbf{w}|\theta) \log \frac{q(\mathbf{w}|\theta)}{P(\mathbf{w}|\mathcal{D})} d\mathbf{w} \\
&= \arg\min_\theta \int_{\mathbf{w}} q(\mathbf{w}|\theta) \log \frac{q(\mathbf{w}|\theta)P(\mathcal{D})}{P(\mathcal{D}|\mathbf{w})P(\mathbf{w})} d\mathbf{w} && \text{(Bayes' Rule)} \\
&= \arg\min_\theta \int_{\mathbf{w}} q(\mathbf{w}|\theta) \log \frac{q(\mathbf{w}|\theta)}{P(\mathcal{D}|\mathbf{w})P(\mathbf{w})} d\mathbf{w} + \log P(\mathcal{D}) \\
&= \arg\min_\theta \int_{\mathbf{w}} q(\mathbf{w}|\theta) \log \frac{q(\mathbf{w}|\theta)}{P(\mathcal{D}|\mathbf{w})P(\mathbf{w})} d\mathbf{w} && (P(\mathcal{D}) \perp\!\!\!\perp \theta) \\
&= \arg\min_\theta \int_{\mathbf{w}} q(\mathbf{w}|\theta) \log \frac{q(\mathbf{w}|\theta)}{P(\mathbf{w})} d\mathbf{w} - \int_{\mathbf{w}} q(\mathbf{w}|\theta) \log P(\mathcal{D}|\mathbf{w}) d\mathbf{w} \\
&= \arg\min_\theta \mathrm{KL}[q(\mathbf{w}|\theta)||P(\mathbf{w})] - \mathbb{E}_{q(\mathbf{w}|\theta)}[\log P(\mathcal{D}|\mathbf{w})] && (5)
\end{aligned}$$

$$\mathcal{F}(\mathcal{D}, \theta) \coloneqq \mathrm{KL}[q(\mathbf{w}|\theta)||P(\mathbf{w})] - \mathbb{E}_{q(\mathbf{w}|\theta)}[\log P(\mathcal{D}|\mathbf{w})] \quad (6)$$

We face an issue in minimising the *ELBO* with respect to θ in that $\mathbf{w}$ is a random variable, and we cannot take derivatives with respect to random variables. If we cannot take derivatives, we cannot use back-propagation to learn the optimal parameters θ^* of the variational posterior.

Blundell, *et al.* prove that, under certain conditions, the derivative of an expectation can be expressed as the expectation of a derivative, as shown in Equation 7. We can let $f(\mathbf{w}, \theta) = \log q(\mathbf{w}|\theta) - \log P(\mathcal{D}|\mathbf{w}) - P(\mathbf{w})$ and apply Equation 7 to the optimisation problem in Equation 4.

$$\frac{\partial}{\partial\theta}\mathbb{E}_{q(\mathbf{w}|\theta)}[f(\mathbf{w}, \theta)] = \mathbb{E}_{q(\epsilon)}[\frac{\partial f(\mathbf{w}, \theta)}{\partial \mathbf{w}}\frac{\partial \mathbf{w}}{\partial \theta} + \frac{\partial f(\mathbf{w}, \theta)}{\partial \theta}] \quad (7)$$

The authors allow ϵ to be a random variable with probability density $q(\epsilon)$ and let $\mathbf{w} = t(\theta, \epsilon)$, where $t(\theta, \epsilon)$ is a deterministic function. Additionally, it is assumed that $q(\epsilon)d\epsilon = q(\mathbf{w}|\epsilon)d\mathbf{w}$. Thus, $t(\theta, \epsilon)$ transforms a sample of parameter-free noise ϵ and the variational posterior parameters θ into a sample from the variational posterior. That is, if we assume that the variational posterior is a Gaussian distribution parameterised by $\theta = (\mu, \sigma)$, such that $\mathbf{w} \sim \mathcal{N}_\theta(\mu, \sigma)$, we can draw a sample of $\epsilon \sim \mathcal{N}(0, 1)$ and use this to obtain a sample from the variational posterior using Equation 8. This is referred to as the *reparameterisation trick*, and enables us to perform back-propagation through the neural network.

$$w = \mu + \sigma\epsilon \quad (8)$$

Equation 9 is used to approximate the *ELBO*, where $\mathbf{w^{(i)}}$ is the i-th Monte Carlo sample drawn from the variational posterior $q(\mathbf{w}^{(i)}|\theta)$. As highlighted by Blundell, *et al.*, by avoiding closed-form calculation of the KL divergence (as well as the expectation), any combination of prior and variational posterior families could be used.

$$\mathcal{F}(\mathcal{D}, \theta) \approx \sum_{i=1}^{n} \log q(\mathbf{w}^{(i)}|\theta) - \log P(\mathbf{w}^{(i)}) - \log P(\mathcal{D}|\mathbf{w}^{(i)}) \quad (9)$$

2.3 Algorithm

As per the original paper, we set the variational posterior to be a diagonal Gaussian parameterised by $\theta = (\mu, \rho)$, such that the standard deviation $\sigma = \log(1 + \exp(\rho))$. Optimisation proceeded according to the algorithm proposed by Blundell, *et al.*:

1. Randomly initialise the parameters θ
2. Sample $\epsilon \sim \mathcal{N}(0, I)$
3. Obtain a sample of the variation posterior as: $\mathbf{w} = \mu + \log(1+\exp(\rho)) \circ \epsilon$, where $\circ$ is point-wise multiplication
4. Let $f(\mathbf{w}, \theta) = \log q(\mathbf{w}|\theta) - \log P(\mathcal{D}|\mathbf{w}) - P(\mathbf{w})$
5. Calculate the gradient of $f(\mathbf{w}, \theta)$ with respect to μ and ρ, as shown by Equations 10 and 11 respectively.

$$\Delta_\mu = \frac{\partial f(\mathbf{w}, \theta)}{\partial \mathbf{w}} \cdot 1 + \frac{\partial f(\mathbf{w}, \theta)}{\partial \mu} \tag{10}$$

$$\Delta_\rho = \frac{\partial f(\mathbf{w}, \theta)}{\partial \mathbf{w}} \frac{\epsilon}{1 + exp(-\rho)} + \frac{\partial f(\mathbf{w}, \theta)}{\partial \rho} \tag{11}$$

6. Update μ and ρ by taking a step in the direction of the derivative, scaled by learning-rate α, as shown by Equations 12 and 13 respectively.

$$\mu \leftarrow \mu + \alpha \Delta_\mu \tag{12}$$

$$\rho \leftarrow \rho + \alpha \Delta_\rho \tag{13}$$

2.4 Priors

Given we were not restricted to Gaussians, we investigated a range of priors. The first was a single Gaussian and acted as a baseline. The second was the mixture-of-Gaussians, spike-and-slab prior given by Equation 14, as proposed by Blundell, *et al.*, where $\mathbf{w}_j$ is the j-th weight of the network, σ_1^2 and σ_2^2 are the variances of the mixture components, and π is the mixture weighting. The first mixture component was given a larger variance than the second, $\sigma_1^2 > \sigma_2^2$, and the second mixture was given a small variance $\sigma_2^2 << 1$. This combination meant that the mixture-distribution had heavier tails than a regular Guassian density, but also set an a-priori assumption that many of the weights are tightly concentrated around zero. Blundell, *et al.* considered $-\log \sigma_1 \in \{0, 1, 2\}$ and $-\log \sigma_2 \in \{6, 7, 8\}$, as well as $-\pi \in \{\frac{1}{4}, \frac{1}{2}, \frac{3}{4}\}$. Figure 2 illustrates the distributions for $\pi = 0.5$.

$$P(\mathbf{w}) = \prod_j \pi \mathcal{N}(\mathbf{w}_j|0, \sigma_1^2) + (1 - \pi)\mathcal{N}(\mathbf{w}_j|0, \sigma_2^2) \tag{14}$$

An alternative prior that we considered was the Laplace distribution, parameterised by μ and b, as shown in Equation 15. Blundell, *et al.* highlight the ability of the Bayesian Neural Network to regularise weights, and show that a significant fraction of weights can be turned-off without detrimental performance. Our theory was that a Laplace distribution, performing a function akin to L1 regularisation, may perform such pruning automatically. Figure 3 compares the Gaussian, mixture-of-Gaussian and Laplace distributions.

$$P(\mathbf{w}) = \frac{1}{2b} \exp(-\frac{|x - \mu|}{b}) \tag{15}$$

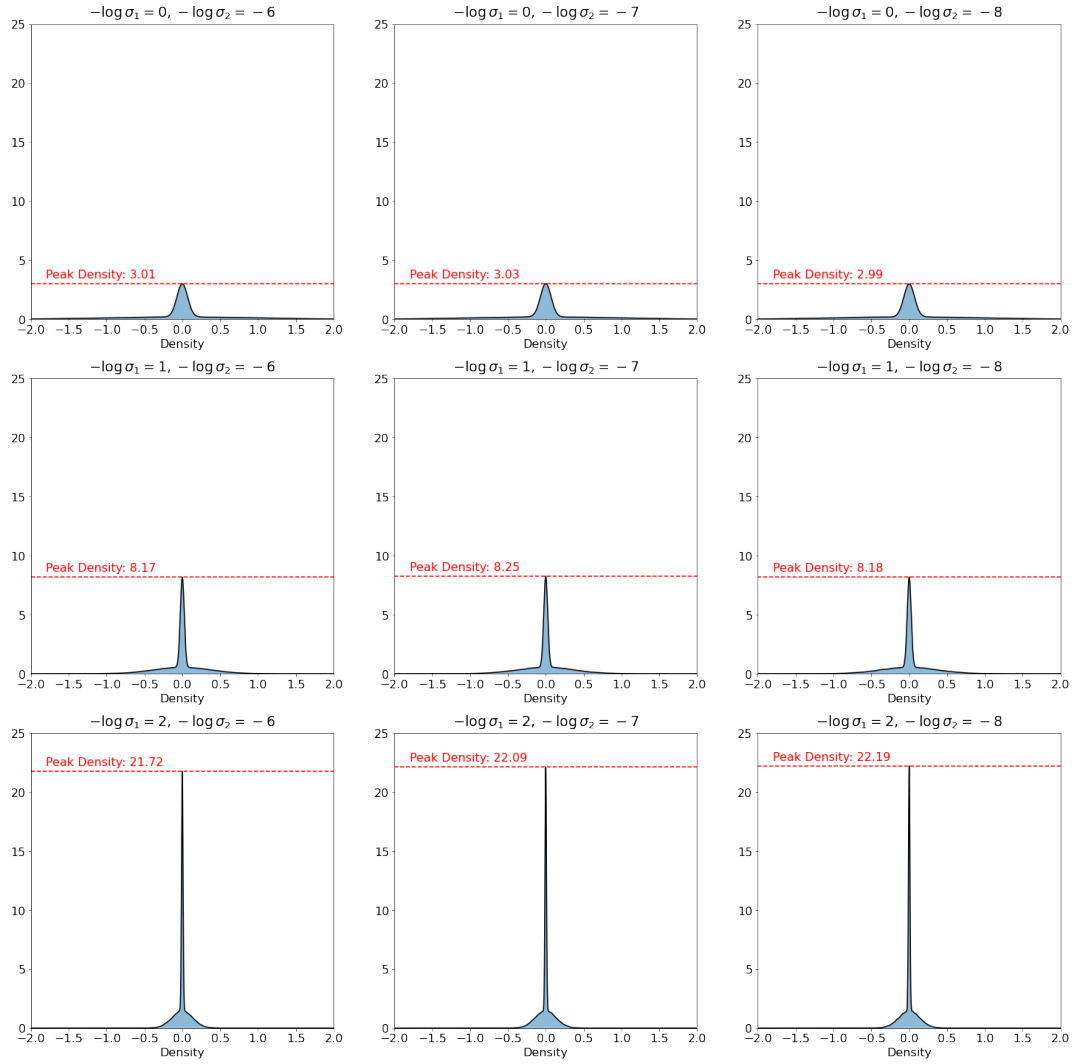


Figure 2: Spike-and-slab mixture-of-Gaussian distributions with $-\log\sigma_1 \in \{0, 1, 2\}$, $-\log\sigma_2 \in \{6, 7, 8\}$ and $\pi = 0.5$. The impact of σ_2 is subtle, and so the densities at the peaks of the distributions have been highlighted.

2.5 Local Reparameterisation Trick

As an extension, we investigated the *local reparameterisation trick* proposed by Kingma, Salimans, and Welling, 2015 in [3]. The trick *"translates uncertainty about global parameters into local noise that is independent across datapoints in the mini-batch,"* and in doing so results in the variance of the free energy estimates being inversely proportional to the mini-batch size. This results in gradients with lower variance, which should lead to faster convergence. The central proposal for the gain in *statistical* efficiency is to sample a separate weight matrix for every training point in the mini-batch, such that that the covariances between the free energy estimates for each data-point are zero. However, assuming there were N datapoints in each mini-batch, and a given layer had an $n \times m$ weight matrix, we would require $N \times n \times m$ samples for that layer alone. The *local* aspect of the trick offers improved *computational* efficiency, by sampling the activations directly, rather than sampling the weights and then computing activations. The resulting reparameterisation equation is given by Equation 16, where $\gamma_{n,j}$ and $\delta_{n,j}$ are determined via Equations 17 and 18 respectively, and $\zeta_{n,j} \sim \mathcal{N}(0, 1)$. Thus, instead of requiring $N \times n \times m$ samples, we now only need $N \times m$. As with the regular BBB algorithm, at inference time a single set of weight samples was drawn from the learned variational posterior distribution and used to make predictions.

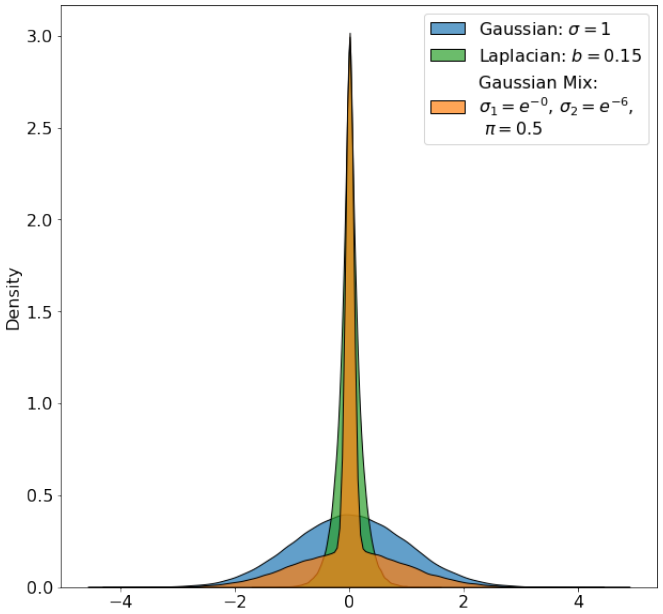


Figure 3: Comparison of prior distributions. Laplace distribution has greater probability mass around zero than the spike-and-slab mixture-of-Gaussian, with less mass in the tails.

$$b_{m,j} = \gamma_{n,j} + \sqrt{\delta_{i,j}}\zeta_{n,j} \tag{16}$$

$$\gamma_{n,j} = \sum_{i=1} a_{n,i}\mu_{i,i} \tag{17}$$

$$\delta_{n,j} = \sum_{i=1} a_{n,i}^2\sigma_{i,i}^2 \tag{18}$$

2.6 Log Pointwise Predictive Density

For regression models, while methods such as the *root mean squared error* can be used for model evaluation, ideally we want to take into account the uncertainty the model has about a predicted value $\hat{y}_i$ and choose the model with the highest posterior predictive probability, $P(\hat{y}_i|\mathbf{y}, \mathbf{x})$ [4], [5]. To this end, the *log pointwise predictive density (lppd)* can be calculated with Equation 19, using N new datapoints drawn from the true data-generating process [4].

$$lppd = \log \prod_{i=1}^{N} P(\hat{y}_i|\mathbf{y}, \mathbf{x}) = \sum_{i=1}^{N} \log \int_{\mathbf{w}} P(\hat{y}_i|\mathbf{y}, \mathbf{x}, \mathbf{w}) P(\mathbf{w}|\mathbf{y}, \mathbf{x}) d\mathbf{w} \tag{19}$$

Given we do not know the posterior distribution, we instead draw S samples from it and estimate the *lppd* using these samples, as shown in Equation 20 ($\mathbf{w}^s \sim P(\mathbf{w}|\mathbf{y}, \mathbf{x})$) [4].

$$\widehat{lppd} = \frac{1}{N} \sum_{i=1}^{N} \log \frac{1}{S} \sum_{s=1}^{S} P(\hat{y}_i|\mathbf{y}, \mathbf{x}, \mathbf{w}^s) \tag{20}$$

2.7 Contextual Bandits

In contextual bandit problems, an agent is presented with some context $x \in X$ from the environment and must choose an action a out of K possible actions. After the agent takes an action it receives a reward r. The overall goal of the agent is to maximise the cumulative reward it obtains over time. The agent must learn a model for $P(r|\mathbf{x}, \mathbf{a}, \mathbf{w})$ from observed $(\mathbf{x}, \mathbf{a}, r)$ triplets, where $\mathbf{w}$ are the parameters of the model. In our case, the model will take the form of a Neural Network, where $\mathbf{w}$ are the weights of the network. As the agent only knows the triplets involving actions a that it has taken, it must have the ability to trade-off between exploration and exploitation.

Two popular methods for trading-off exploration and exploitation are epsilon-greedy exploration and Thompson sampling. In epsilon-greedy exploration the agent has a probability ϵ of taking a random action, and so picks the action with highest anticipated reward $1 - \epsilon$ percent of the time. In Thompson sampling [6], at each step a set of weights is drawn from the posterior function, i.e. sample $\tilde{\mathbf{w}} \sim P(\mathbf{w}|\{(\mathbf{x}, \mathbf{a}, r)\})$. The action with highest expected reward based on the likelihood $P(r|\mathbf{x}, \mathbf{a}, \mathbf{w})$ is then selected. We replicate the work of Blundell, *et al.* to evaluate the use of a Bayesian Neural Network, and compare this with the performance of ϵ-greedy agents.

2.8 Adversarial Examples

As an extension to Blundell, *et al.*'s experiments, we evaluated BNN performance against adversarial MNIST example images. These were generated using the *fast gradient sign method* proposed by Goodfellow, Shlens, and Szegedy in [1]. Given model parameters $\boldsymbol{\theta}$, inputs $\mathbf{x}$, targets y, and a cost function $J(\boldsymbol{\theta}, \mathbf{x}, y)$ used to train a Neural Network, an optimal perturbation, $\boldsymbol{\eta}$, was calculated using Equation 21. The gradients were determined using *PyTorch* by training a simple CNN on the MNIST dataset [7]–[9]. We created perturbed images as $\tilde{\mathbf{x}} = \mathbf{x} + \epsilon \cdot \boldsymbol{\eta}$, where ϵ controls the degree of perturbation applied. Examples are shown in Figure 4.

$$\boldsymbol{\eta} = \text{sign}\nabla_{\mathbf{x}} J(\boldsymbol{\theta}, \mathbf{x}, y) \tag{21}$$

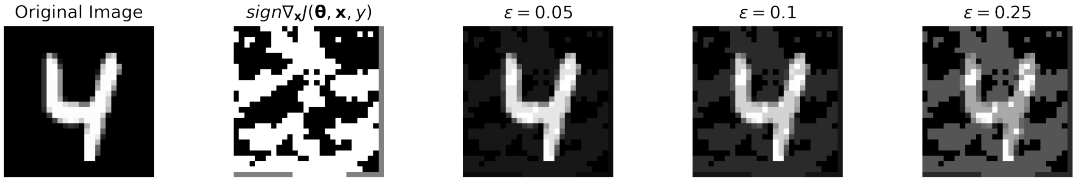


Figure 4: Adversarial examples for an image in the MNIST dataset. The original image and sign of the gradient of the cost function with respect to the input are also shown.

3 Implementation

We utilised the open-source library *PyTorch* for our implementation [7]. Critically, the automatic differentiation provided by the *autograd* functionality meant we were not required to write code to calculate the derivatives presented in Equations 10 and 11 ourselves.

While our implementation was grown from an empty repository, several publicly available implementations of the *Bayes-by-Backprop* algorithm were exploited when designing, writing and debugging our version [10]–[14]. The repositories of Antoran, Markou, and Qing [10] and Liu, Murray, and Persky [14] were the most extensively drawn upon. There are, however, still many unique aspects to our own implementation. For example, we feel our class inheritance structure, shown in Figure 5, is far more robust and extendable than those of other implementations, and certainly aided in accelerating development. Further, the fact that certain implementations did not face issues came as a surprise to us. When implementing the *local reparameterisation trick* we discovered the practical requirement to add an imperceptible amount of noise (of the order

1^{-32}) to Equation 18 to avoid vanishing gradients during back-propagation. We found this was also present in Shridhar, Raikwar, Mehta, *et al.*'s implementation [13], but did not appear in many others. Differences in data and library versions may have had an impact.

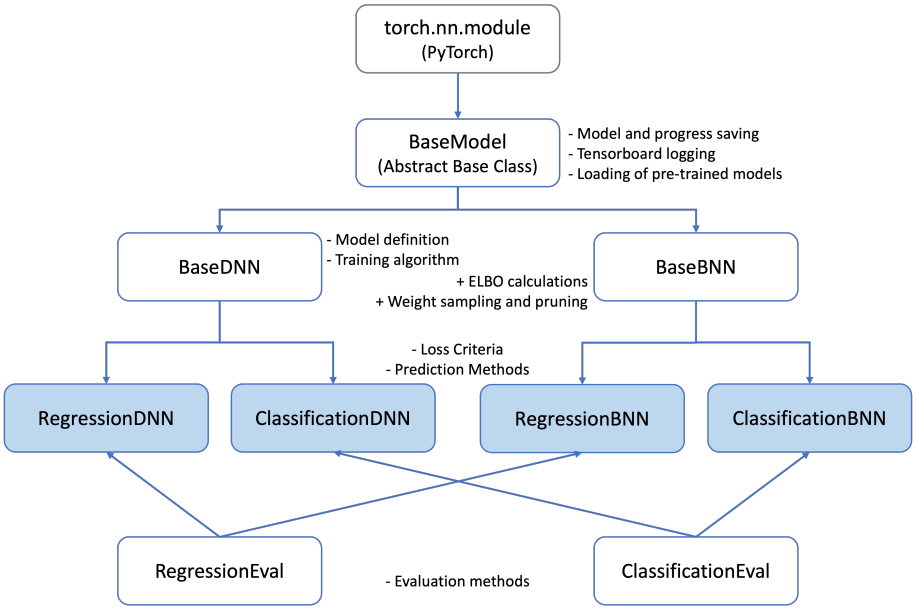


Figure 5: Class inheritance structure designed and coded in our implementation. The enabled significant code reuse, accelerating development and simplifying debugging.

We implemented a `GaussianVarPost` class to represent the variational posterior of an entire layer of weights, from which we could sample, obtain the log probability, and back-propagate through. We initialised the weights uniformly across a tight range around a pre-specified mean. For the variational posterior, we found a weight initialisation of $\mu \sim U(-0.2, 0.2)$ and $\rho \sim U(-5, -4)$, where $\sigma = log(1 + exp(\rho))$, to work well. Initialisation to a constant value across all the weights led to back-propagation issues as the update was consequently identical for each weight.

For our gradient-based optimisation algorithm, we opted for *Adam* [15] instead of *Stochastic Gradient Descent (SGD)*. Training time and convergence were potential issues for BNNs and Adam generally converges faster than SGD.

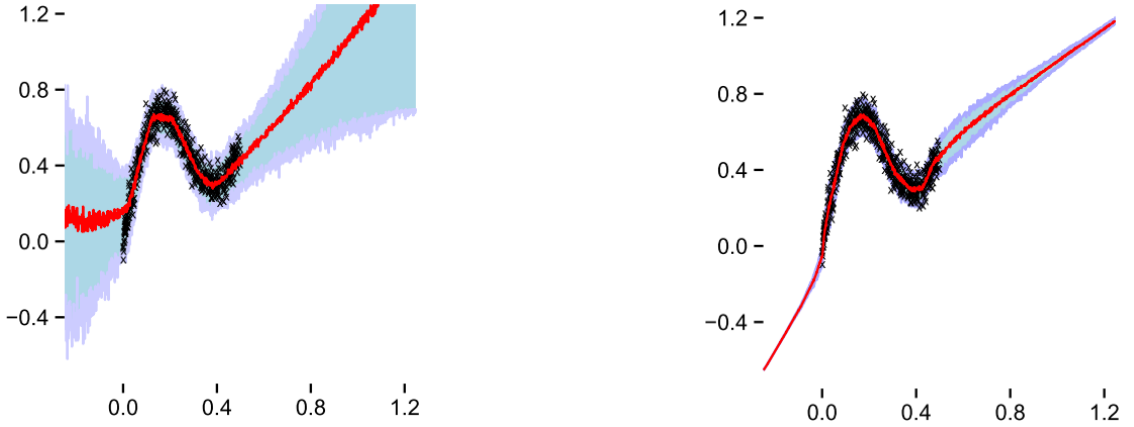
4 Experiments

In the following regression and classification experiments, each model had three layers (including input and output) with linearly rectified (ReLU) units. For regression, we used 400 hidden units per layer. For classification, we evaluated 400, 800 and 1200 hidden units. For the RL experiment, we used two layers with 100 ReLU units each. Other (hyper-)parameter specifications for each experiment are detailed in the appropriate sections.

4.1 Regression

In alignment with Blundell, *et al.* [2], regression data was generated according to Equation 22, where $\epsilon \sim \mathcal{N}(0, 0.02)$. Eight batches of 128 datapoints were generated in the region $x \in [0, 0.5]$ for training. For evaluation, 1000 datapoints in the region $x \in [-0.5, 1.5]$ were used. During training 5 weight samples were drawn when estimating the *ELBO* using Equation 9. For reference, Figure 6 shows the results presented by Blundell, *et al.*.

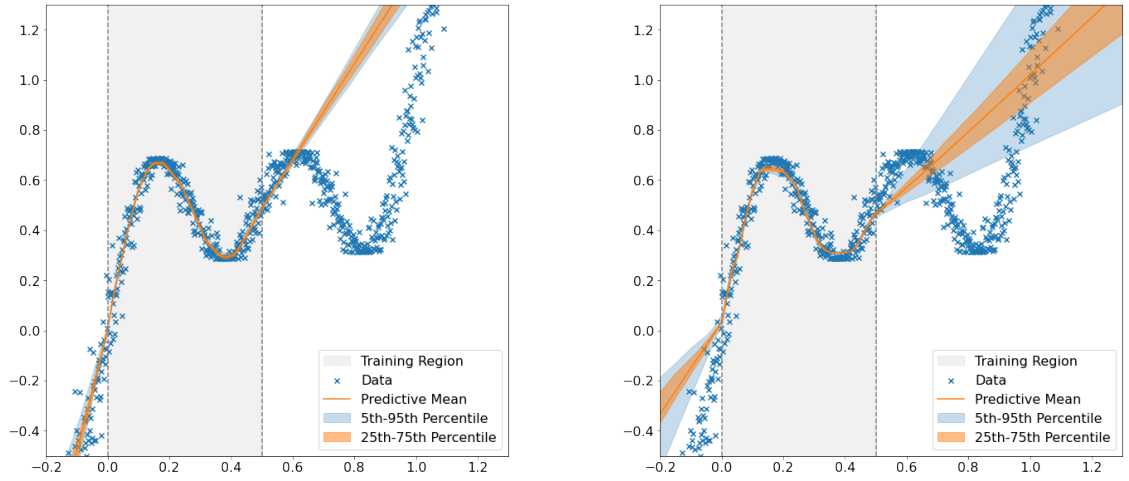
$$y = x + 0.3\sin(2\pi(x + \epsilon)) + 0.3\sin(4\pi(x + \epsilon)) + \epsilon \tag{22}$$



(a) Bayes by Back-prop Neural Network (b) Standard Neural Network

Figure 6: Replication of Figure 5 from Blundell, *et al.*[2], showing the authors' regression results. Black crosses are training samples. Red lines are median predictions. Blue/purple regions are (unspecified) interquartile ranges.

As a baseline, Figure 7 shows the mean and interquartile ranges for predictions generated using an ensemble of 25 independently trained standard DNNs and networks with 50 % dropout during training. Aligned with Figure 6, characteristic overconfidence is displayed by the standard networks, which starts to be addressed by training with dropout.

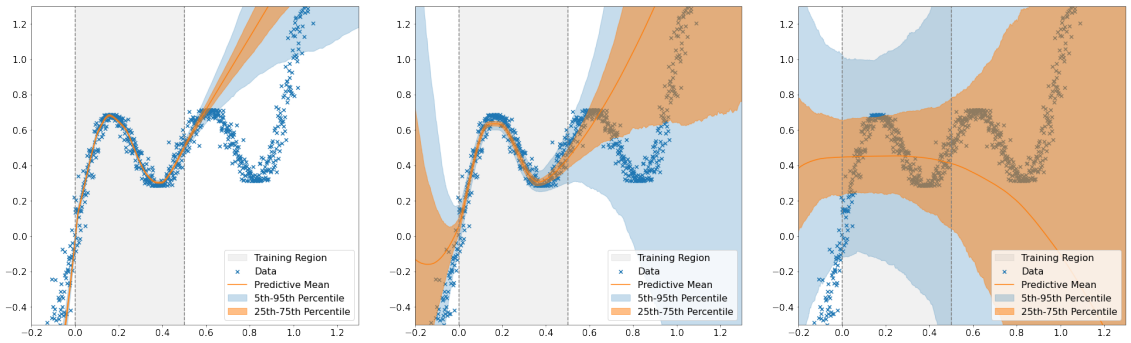


(a) No Dropout (b) 50 % Dropout

Figure 7: Mean and interquartile ranges for inference performed using an ensemble of 25 independently trained Deep Neural Networks, with and without dropout.

For the BBB algorithm, it was assumed that observations $\mathbf{y}$ could be expressed as the result of a deterministic function applied to the input, $g(\mathbf{x})$, perturbed by a source of noise $\epsilon \sim \mathcal{N}(0, \sigma_\epsilon)$. That is: $\mathbf{y} = g(\mathbf{x}) + \epsilon$. The likelihood term in Equation 9 was therefore calculated using $P(\mathcal{D}|\mathbf{w}) = \mathcal{N}(\mathcal{D}; \hat{g}(\mathbf{x}, \mathbf{w}), \sigma_\epsilon)$, where $\hat{g}(\mathbf{x}, \mathbf{w})$ is the approximation of $g(\mathbf{x})$ learned by the Neural Network. Whilst in this case it was known that $\sigma_\epsilon = 0.02$, usually this would be an unknown parameter. Figure 8 shows the results of using $\sigma_\epsilon \in \{0.01, 0.1, 1.0\}$. A value of 0.01 resulted in the model displaying overconfidence. If σ_ϵ is set to a very low value it is potentially difficult to find optimal parameters of the variational posterior to minimise the *ELBO* that can simultaneously describe the observed data well and minimise the divergence with the prior. Conversely, a value of 1.0

resulted in poor inference: too many points have high $P(\mathcal{D}|\mathbf{w})$, and the model is able to explain the data through the noise. $\sigma_\epsilon = 0.1$ offers a middle-ground, and achieved the lowest *ELBO* at the end of 1500 training epochs, as shown in Figure 9. This value was therefore used in the remaining experiments.



(a) $\sigma_\epsilon = 0.01$ (b) $\sigma_\epsilon = 0.1$ (c) $\sigma_\epsilon = 1.0$

Figure 8: Exploration of the likelihood noise, σ_ϵ, used during training. The mean and percentile ranges across 1000 inference samples are shown.

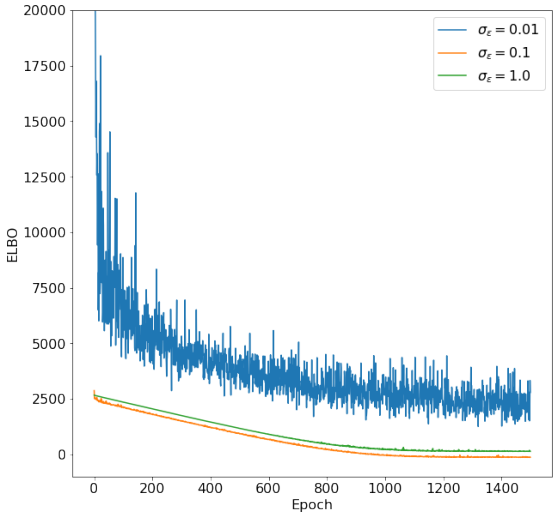
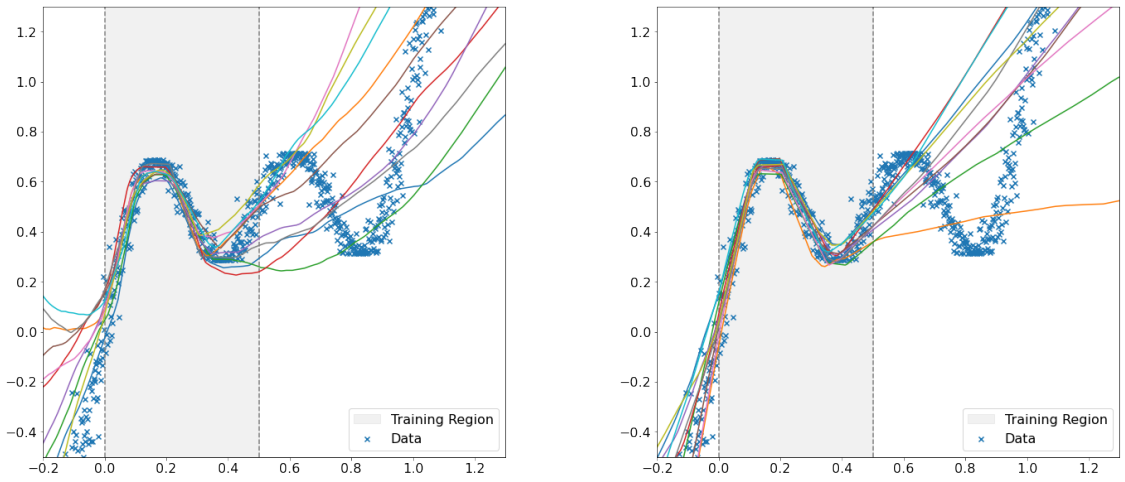


Figure 9: *ELBO* vs. training epochs for a selection of likelihood noise, σ_ϵ, values.

Figure 10 shows example inferences curves, each generated using a different sample of weights drawn from the learned variational posterior, for both a single Gaussian prior (SGP) and mixture-of-Gaussians prior (MoGP). Figure 11 shows the mean and interquartile ranges across 1000 samples. Both express uncertainty in regions of the input space they'd not observed during training. Further, the 5th-95th percentile ranges mostly capture the noise in the training data.

SGPs with standard deviations $-\log\sigma \in \{0, 1, 2\}$ were investigated. Figure 12 shows the *ELBO* and *RMSE* values obtained during training. The rate of convergence increased as σ decreased, however all models achieved similar final *ELBO* values. It is interesting to note from Figure 12b that the *RMSE* initially decreased, but then increased as learning progressed. This is thought to be the result of the model learning weight distributions that allow it to exhibit appropriate uncertainty. Table 1 shows that the *lppd* increased as σ decreased.

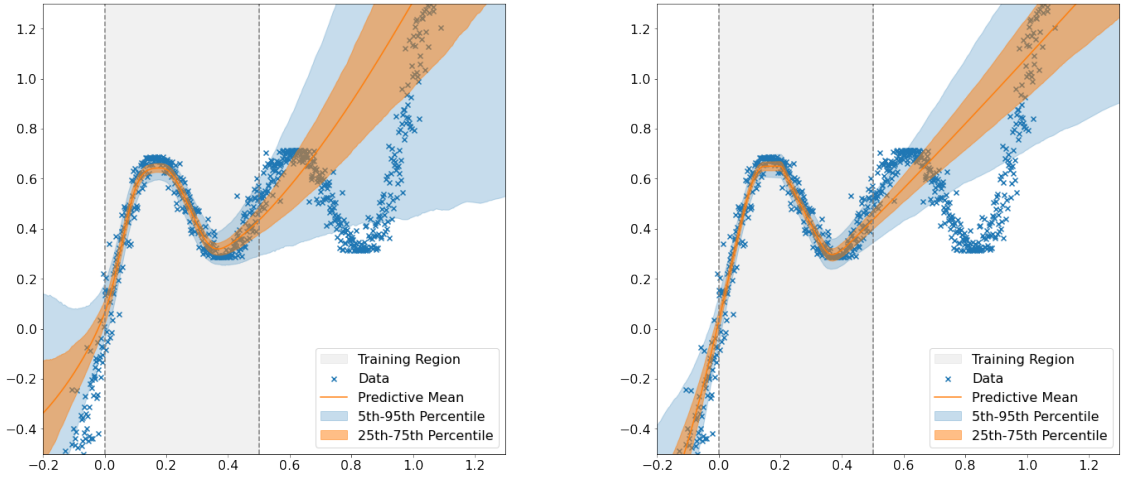
MoGPs with standard deviations $-\log\sigma_1 \in \{0, 1, 2\}$ and $-\log\sigma_2 \in \{6, 7, 8\}$ were also investigated, with the mixture parameter $\pi = 0.5$ held constant. Figure 13 shows the *ELBO* and *RMSE* values obtained during training. In general, it was observed that significantly more training was needed



(a) Single Gaussian Prior
$-\log\sigma = 1$

(b) Mixture-of-Gaussians Prior
$-\log\sigma_1 = 1$, $-\log\sigma_2 = 6$, $m = 0.5$

Figure 10: Ten example inference curves generated from trained Bayesian Neural Networks.



(a) Single Gaussian Prior
$-\log\sigma = 1$

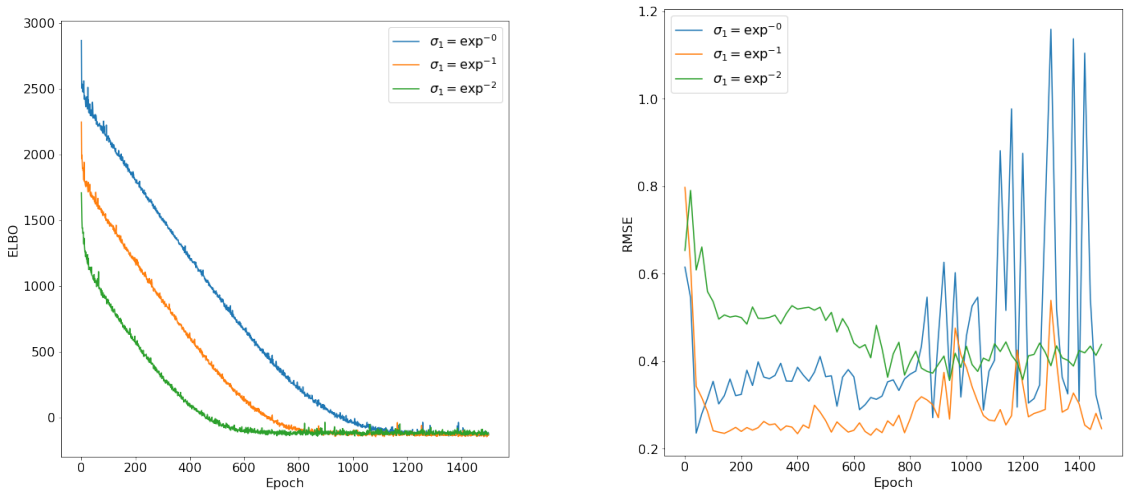
(b) Mixture-of-Gaussians Prior
$-\log\sigma_1 = 1$, $-\log\sigma_2 = 6$, $m = 0.5$

Figure 11: Mean and interquartile ranges for inference performed using trained Bayesian Neural Networks across 1000 inference samples.

$-\log\sigma$	0	1	2
lppd	-120.34	-16.21	-9.69

Table 1: *lppd* for Bayesian Neural Networks trained using single Gaussian priors with standard deviation σ. Averages over 5,000 weight samples were taken.

for convergence than in the SGP case, and that the final *ELBO* values obtained were not as low. The increase will be driven by the complexity cost term in Equation 6: given that the variational posterior is still a single Gaussian, there is an upper limit to the KL divergence that can be achieved between it and the MoGP. Convergence time increased as σ_1 increased and σ_2 decreased—indicating a drive to unite the Gaussian distributions. Figure 13b shows that the variance in the *RMSE* decreased as σ_1 decreased, which is likely the result of the narrower prior



(a) *ELBO* (b) *RMSE*

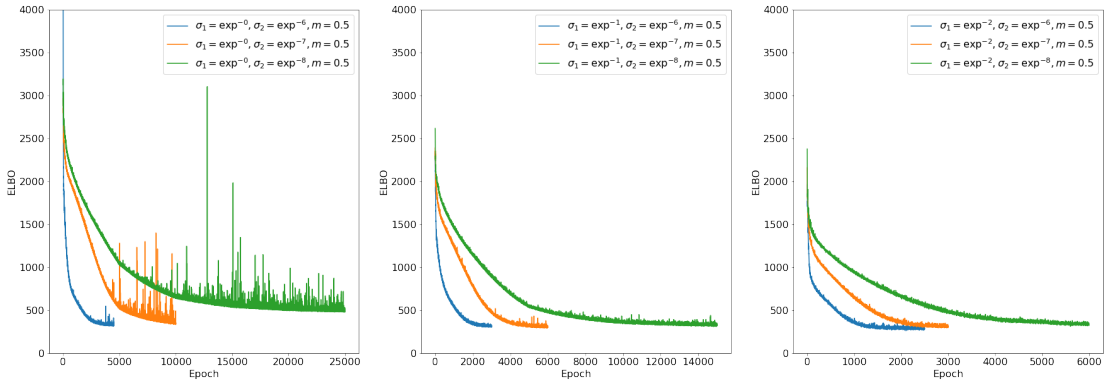
Figure 12: *ELBO* and *RMSE* during training for Bayesian Neural Networks using single Gaussian priors with standard deviation σ. Note that RMSE was determined every 20 epochs.

forcing the weight distributions to be more concentrated around zero. Table 2 shows that the lowest *lppd* was obtained for the $-\log\sigma_1 = 1, -\log\sigma_2 = 6$ prior.

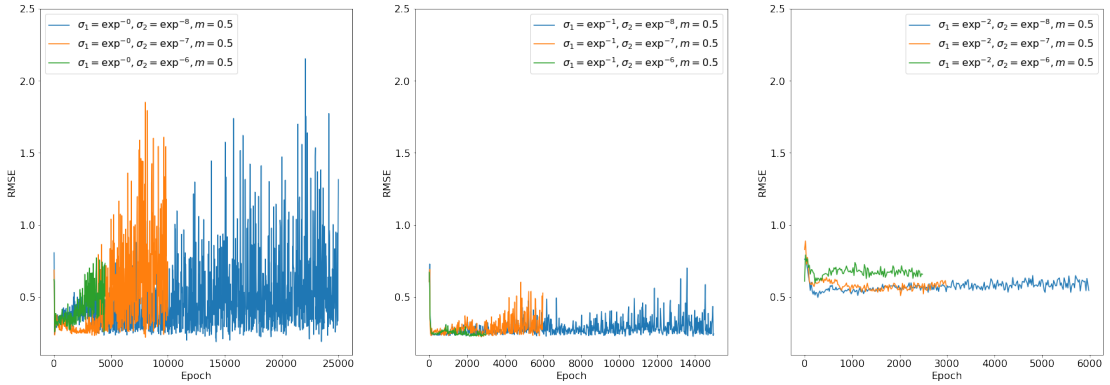
		$-\log\sigma_2$		
		6	7	8
$-\log\sigma_1$	0	-32.40	-412.87	-184.17
	1	-3.21	-22.46	-21.29
	2	-21.58	-16.77	-16.20

Table 2: *lppd* for Bayesian Neural Networks trained using mixture-of-Gaussian priors with standard deviations σ_1 and σ_2. Mixture parameter $\pi = 0.5$ was held constant. Averages over 5,000 weight samples were taken.

Figure 14 shows the impact on inference of applying the *local reparameterisation trick*[16] during training. Compared to 11a, the LRT-trained model had wider interquartile ranges. Consequently, the *lppd* was calculated to be -26.03, compared to -3.21 for the non-LRT approach. Figure 15 shows the *ELBO* and *RMSE* during training. Interestingly, the rate of convergence was not observed to benefit from the LRT, even though the variance of the difference between consecutive *ELBO* values decreased from 444.8 to 244.0. A reduction in the variance of the *RMSE* can also be seen in Figure 15b. A potential reason for the observed lack of improvement was our use of the Adam optimiser, given it performs parameter updates that are invariant to re-scaling of the gradient [15]. Across 1500 epochs, using PyTorch CUDA Event handlers it was found that the non-LRT approach required 317.8 ms/epoch, whereas LRT needed 211.7 ms/epoch—a time saving of 33%.



(a) *ELBO*



(b) *RMSE*

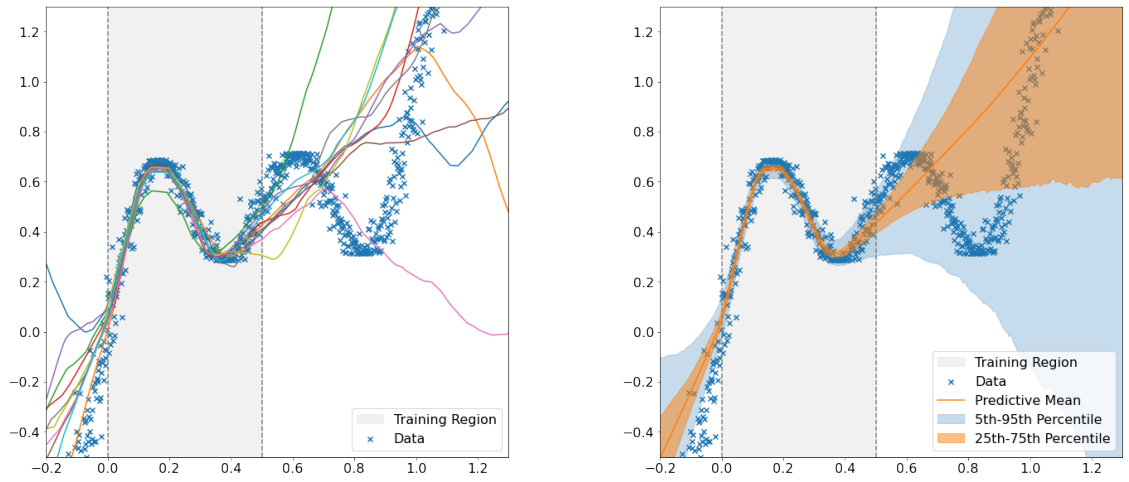
Figure 13: *ELBO* and *RMSE* during training for Bayesian Neural Networks using mixture-of-Gaussian priors with standard deviations σ_1 and σ_2. Mixture parameter $\pi = 0.5$ was held constant. Note that RMSE was determined every 20 epochs.

4.2 MNIST Classification

We replicated Blundell, *et al.*'s classification experiments using the MNIST digits dataset [17]. We evaluated BNNs against standard DNNs, as well as DNNs with dropout applied during training. We assessed their accuracy, learning curves, calibration, and the distributions of the learned weights. Moreover, we considered pruning the BNN based on the signal-to-noise ratios of the weights and inspected the resulting effect on test accuracy. Finally, we extended Blundell, *et al.*'s experiments by evaluating the trained Neural Networks against adversarially perturbed MNIST examples.

Hyperparameters: Blundell, *et al.* specify a grid of hyperparameters they explored, but fail to provide the final configurations which yielded the results they published. We fixed the batch size to 128, the learning rate to 10^{-4}, and trained for 600 epochs. For the BNNs, we drew 2 *ELBO* samples during training and 50 at inference time. We invesigated the use of both a single Gaussian prior (SGP) and a mixture-of-Gaussians prior (MoGP). For the SGP we used a $\mathcal{N}(0, 1)$ distribution. For the MOGP we evaluated a subset of the parameters explored by the paper: $\pi \in \left\{\frac{1}{4}, \frac{1}{2}, \frac{3}{4}\right\}$, $-\log \sigma_1 \in \{0, 1, 2\}$ and $-\log \sigma_2 \in \{6, 7, 8\}$. The best performing MoGP parameters, and so those that our reported results were obtained with, were found to be $\pi = \frac{1}{2}$, $-\log \sigma_1 = 0$, $-\log \sigma_2 = 7$.

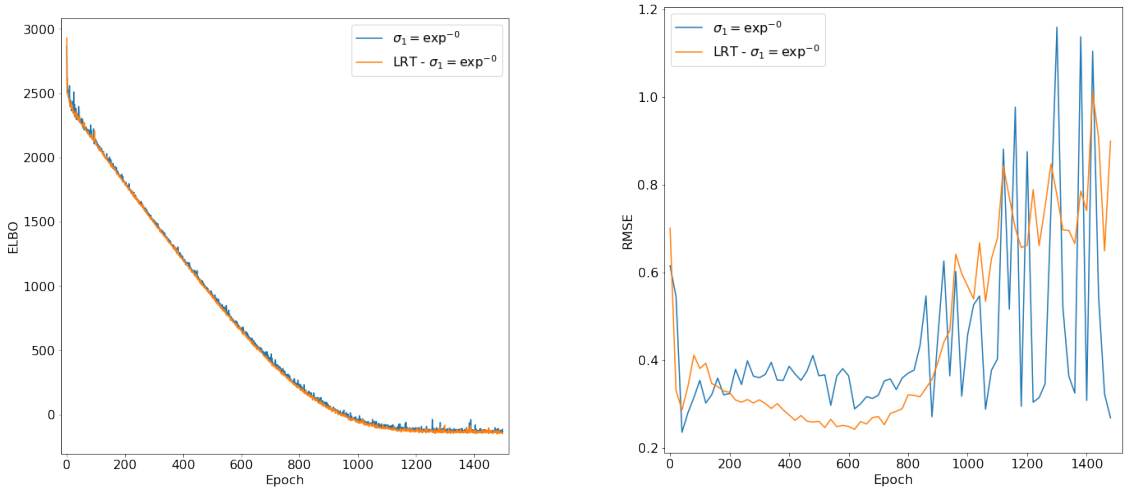
Classification accuracy: We evaluated our trained models against a test set of 10K MNIST images. We report our test error rates against those of Blundell, *et al.* in Table 3. We observed that BNNs outperformed DNNs, and attained similar, if not better, performance to DNNs with Dropout—confirming the paper's observations. However, the results are not in perfect alignment.



(a) 10 Samples

(b) Mean and Interquartile Ranges for 1000 weight samples

Figure 14: Inference performed by Bayesian Neural Network trained using the *local reparameterisation trick (LRT)*. Training used a single Gaussian prior with $\sigma = \exp^{-1}$.



(a) *ELBO*

(b) *RMSE*

Figure 15: *ELBO* and *RMSE* during training for Bayesian Neural Networks using single Gaussian priors with standard deviation $\sigma = 1$. Results are shown for training both with and without the *local reparameterisation trick (LRT)*. Note that *RMSE* was determined every 20 epochs.

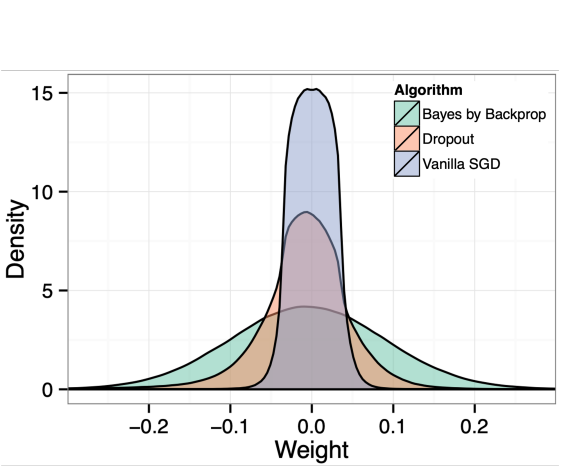
Our BNN with SGP outperformed our BNN with MoGP by 10 to 20 basis points, whereas the paper reported the BNN with MoGP outperformed the SGP by 60 to 70 basis points. Moreover, our BNN with SGP outperformed both sets of the paper's BNN models.

We suspect this is due to differences in the specification of the SGP, given the paper never specifies the mean and variance used. The paper states of its MoGP that "The first mixture component of the prior is given a larger variance than the second, $\sigma_1 > \sigma_2$, *providing a heavier tail in the prior density than a plain Gaussian prior.*" From the latter, we suspect that their SGP had a σ closer to their MoG σ_2, for which they evaluated $-\log \sigma_2 \in \{6, 7, 8\}$. This would lead to an SGP variance $\mathcal{N}(0, \sigma_2)$ much smaller than our $\mathcal{N}(0, 1)$. This hypothesis is supported by the observed differences in weight distributions discussed below.

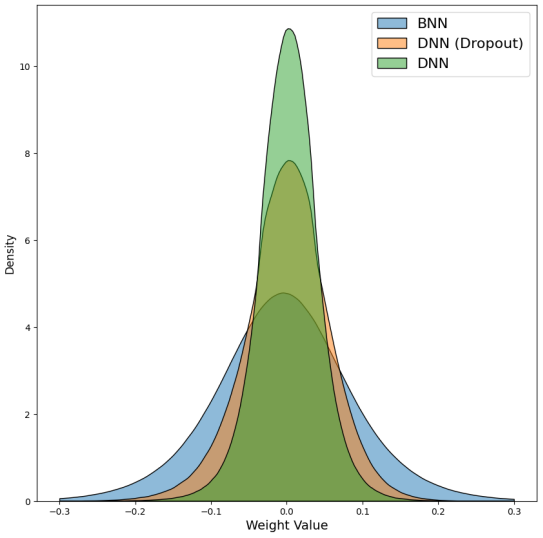
Table 3: Classification Error Rates on MNIST. For BNNs, the results are an average over 50 weight samples. For each network the lowest error rate is bolded.

Model	**# Units/ Layer**	**# Weights**	**Blundell Test Error (%)**	**Our Test Error (%)**
BNN, Single Gaussian Prior	400	500k	1.82	**1.31**
	800	1.3m	1.99	**1.19**
	1200	2.4m	2.02	**1.35**
BNN, Mixture-of-Gaussians Prior	400	500k	**1.36**	1.42
	800	1.3m	**1.34**	1.42
	1200	2.4m	**1.32**	1.45
DNN	400	500k	**1.83**	1.95
	800	1.3m	1.84	**1.68**
	1200	2.4m	1.88	**1.83**
DNN, Dropout	400	500k	1.51	**1.27**
	800	1.3m	**1.33**	1.41
	1200	2.4m	**1.36**	1.37

Weight Distributions: We plot the distribution of weights for the BNN, DNN, and DNN with dropout models in Figure 16. Using a BNN with a $\mathcal{N}(0, 1)$ SGP and 1200 hidden units, our results align with those of the original paper. We found that a BNN with MoGP resulted in a much narrower distribution relative to a SGP. This was particularly noteworthy as the original paper states that "The scale mixture prior used by Bayes by Backprop encourages a broad spread of the weights". However, the MoGPs belonging to their hyperparameter grid are spike-and-slab distributions with significant density at zero (cf. Figure 3), particularly in comparison to our $\mathcal{N}(0, 1)$ SGP. We would therefore expect the resulting variational posterior to be much narrower as well, as can be seen in Figure 17. This discrepancy is explained by our earlier hypothesis regarding the original paper's (unreported) SGP parameters, which, if true, would mean their SGP was much closer to our MoG prior. Figure 17 also shows the weight distribution resulting from using a Laplace prior with $b = 0.15$, which was found to be similar to that of the model trained with the SGP, and so did not provide a significantly higher degree of regularisation.



(a) Original Paper [2]



(b) Our Results

Figure 16: Distributions of trained weights. Our models were trained for 600 epochs. The BNN model used a SGP and 1200 hidden units per layer.

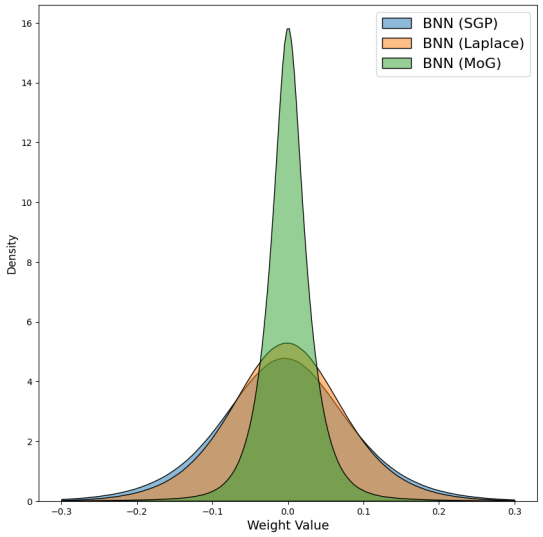
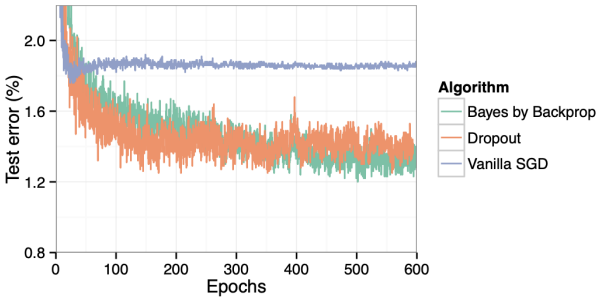
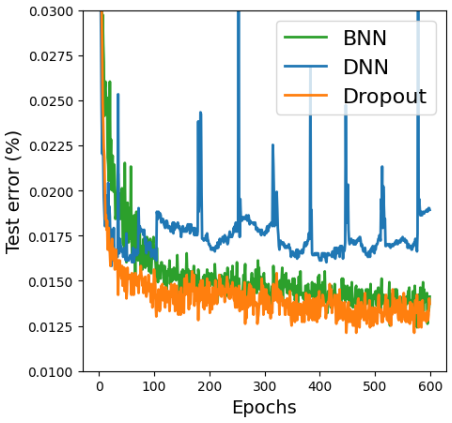


Figure 17: Distributions of weights for BNN models trained using a variety priors. Models were trained for 600 epochs and used 1200 hidden units per layer.

Learning curve: We plot the model's learning curves in Figure 18. We confirm the observation of the original paper that DNNs converge earlier, but with a higher test error rate. The BNN converges more slowly, but to a similar, if not better, accuracy as the DNN with dropout. Oddly, our DNN does not converge as smoothly and has occasional spikes in its test error rate. Our suspicion is that certain mini-batches might affect Adam's optimisation trajectory negatively, given we see the spikes occurring periodically [18].



(a) Original Paper [2]

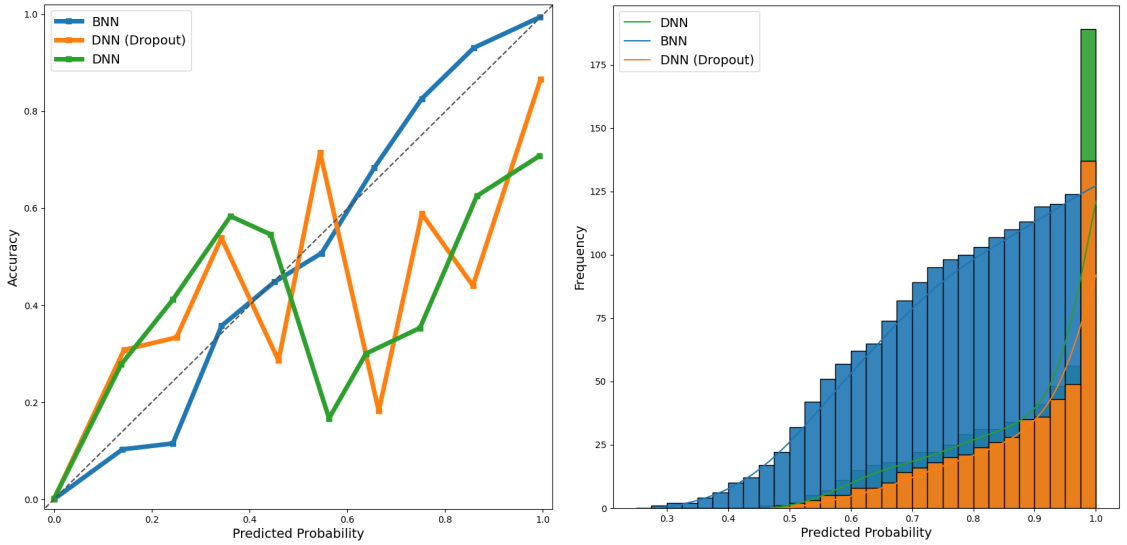


(b) Our Results

Figure 18: Test error rate during training for a regular DNN, a DNN trained with 0.5 dropout rate, and a BNN (SGP, 1200 hidden units).

Calibration: We plot the model's calibration curves in Figure 19. The curves were created by bucketing the model's prediction scores and then mapping those buckets to their percentage accuracy (i.e., the percentage of labels that are correct in each bucket). A perfectly calibrated model means that its output score is in exact alignment with the accuracy of that prediction, indicated by the dashed diagonal line. As one can see, the BNN was very close to this line. In contrast, the DNN and DNN with dropout's calibration lines reflect the characteristic overconfidence these models are known for. For prediction scores around 1.00, the DNN was only correct around 70% of the time

and the DNN with dropout only around 90%, whereas the BNN obtained perfect accuracy.



(a) Calibration line

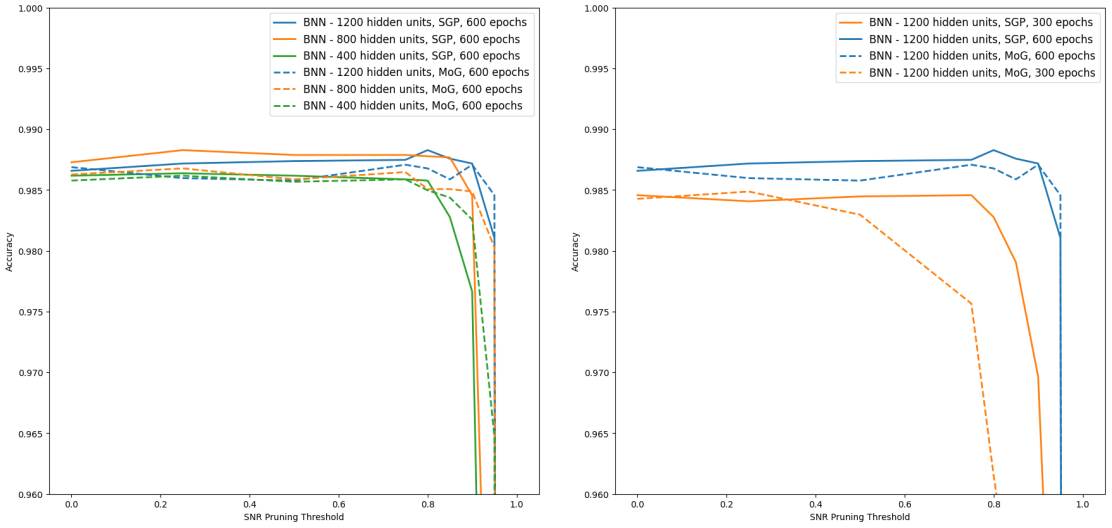
(b) Incorrect predictions histogram.

Figure 19: Model Calibration. All models were trained for 600 epochs. BNN used SGP with 1200 hidden units

Weight pruning: For our BNN weight pruning experiments, weights with a Signal-to-Noise ratio ($\frac{|\mu_i|}{\sigma_i}$, SNR) below a threshold had their variational posterior replaced with a Dirac delta function. The threshold $t \in [0, 1]$ was mapped to a SNR number threshold that captured the t% lowest SNRs of all our weights. Our results are provided in Figure 20, which shows that the accuracy remained at around 98.5 % until 95% of the weights were pruned, in agreement with the original paper's observations. The original paper only showed the results of a BNN with a MoG prior and 1200 hidden units per layer, but we found that pruning's effect on accuracy is approximately constant across the number of hidden units and priors. Moreover, we found that the robustness of the network to pruning significantly improves when the BNN is trained for 600 epochs instead of 300, as is evidenced by the horizontal jump between curves in 20b.

We also investigated the distribution of pruned weights across the individual layers of the network, as shown in Figure 21. Interestingly, we found that the output layer's weights had a higher SNR and were most critical to performance. We compare this with the weight distributions in each layer, highlighting how the output layer had the broadest weight distribution.

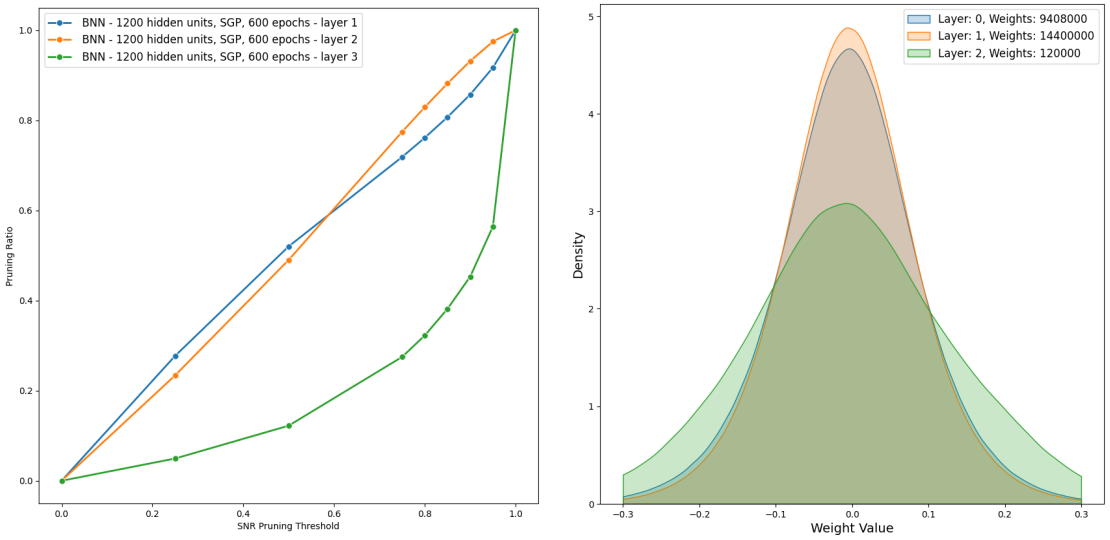
Adversarial examples: We investigated the robustness of the trained models against a set of perturbed MNIST images, generated as described in Section 2.8. We show the models' accuracy against the size of the perturbation in Figure 22. We note that the BNN is more robust to adversarial examples than the DNN and DNN with dropout for a wide range of ϵ values. Specifically, at $\epsilon = 0.05$, the BNN only dropped about 2 % in accuracy, whereas the DNN and DNN with dropout dropped 12 % and 6 % respectively. At $\epsilon = 0.10$, the DNNs were severely impacted whereas the BNN still had an accuracy of 84%. This highlights the robustness of the BNN's learned feature representations by having a distribution over its weights.



(a) All BNNs trained for 600 epochs

(b) BNNs with 1200 hidden units

Figure 20: Effect of weight pruning on accuracy.



(a) Percentage of weights pruned per layer.

(b) Weight distribution per layer.

Figure 21: Pruning ratio per layer.

4.3 Contextual Bandits

In this section, we repeat Blundell, *et al.*'s contextual bandit experiment on the UCI Mushroom dataset [19], which contains 8124 mushrooms, of which 4208 are edible and 3916 are poisonous. Each mushroom has 22 features, including cap shape, cap colour etc. At each step, the agent is presented with a mushroom and must decide whether to eat it or not. If the agent eats an edible mushroom it receives a reward of 5, whereas if eats a poisonous mushroom it has a 50/50 chance of receiving a reward of either -35 or 5 (i.e., there is a 50 % survival rate for eating poisonous mushrooms). The agent gets a reward of 0 if it decides not to eat. In their paper, Blundell, *et al.* use cumulative regret to measure model performance. Regret is measured against an oracle which knows if mushrooms are edible or not, and so always eats edible mushrooms (receiving a reward of 5 each time) and never eats those that are poisonous.

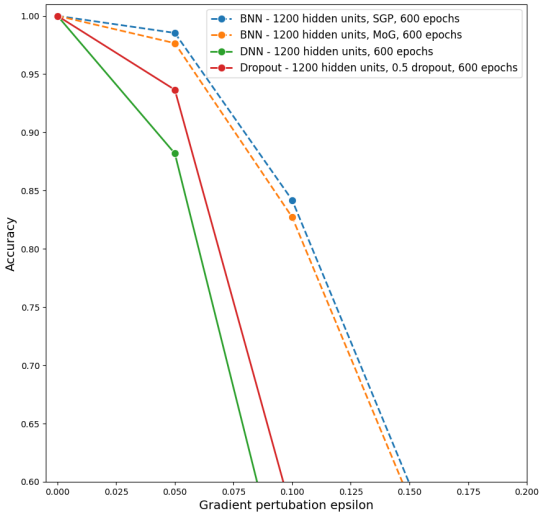
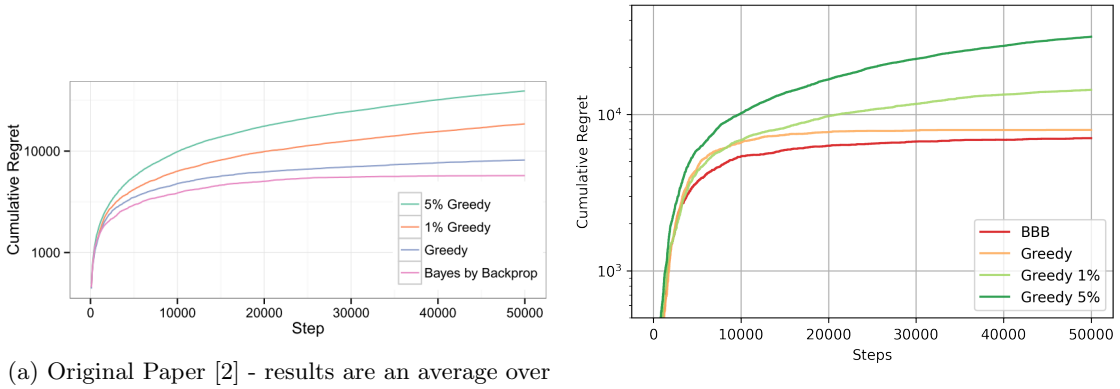


Figure 22: Adversarial robustness.



(a) Original Paper [2] - results are an average over 5 runs

(b) Our Results - results are an average over 3 runs

Figure 23: Cumulative regret for various agents on the mushroom bandit task.

The input to the network is a one-hot representation of the mushroom's 22 features (resulting in 95 dimensions) combined with a one-hot encoded representation of the action taken (eaten or not eaten). The Neural Network used to learn $P(r|\mathbf{x}, \mathbf{a}, \mathbf{w})$ has two hidden layers with 100 hidden units each. The output of the network is a single scalar: the predicted reward for the action taken given the mushroom's features and using the current weights. At each step, the agent takes the action that is associated the largest predicted reward. For the Bayesian Neural Network, we sample the weights twice and average the outputs. Before model training begins, the agent has preliminary knowledge of 4096 mushroom contexts, actions, and rewards, which are the result of taking random actions. In each step the agent is presented with a random mushroom, and has access to a buffer of the most recent 4096 rewards, contexts, and actions.

Figure 23 compares the cumulative regret for the *Bayes by Backprop* agent with three ϵ-greedy agents: $\epsilon = 0\%$ (pure Greedy), $\epsilon = 1\%$, and $\epsilon = 5\%$. The BNN was trained using a mixture-of-Gaussians prior with $-\log \sigma_1 = 0$ and $-\log \sigma_2 = 6$. The ϵ-greedy agents were trained using a standard DNN. The initial learning rate in our experiments was 10^{-4}. A learning rate scheduler multiplied the learning rate by a factor of 0.5 every 5000 steps. All agents were presented with the same mushrooms during training. Our results have similar trends to those of the original paper: the BBB agent converges to the optimal policy faster than the pure Greedy agent. The Greedy agents with $\epsilon = 1\%$, and $\epsilon = 5\%$ accumulate regret linearly due to their taking random actions 1%

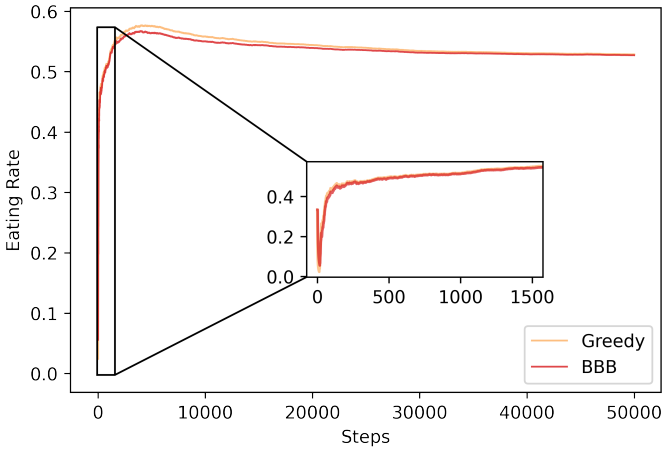


Figure 24: Eating rate of BBB agent and pure Greedy agent averaged over 3 independent runs.

and 5% of the time respectively. Blundell, *et al.* state that the Greedy agent elects to eat nothing in the first 1000 steps before suddenly deciding to start eating. However, Figure 24 shows that in our experiments the Greedy agent had a similar overall eating rate to the BBB agent. As shown in Figure 25, we found that the increase in cumulative regret was primarily due to the higher consumption of poisonous mushrooms by the Greedy agent.

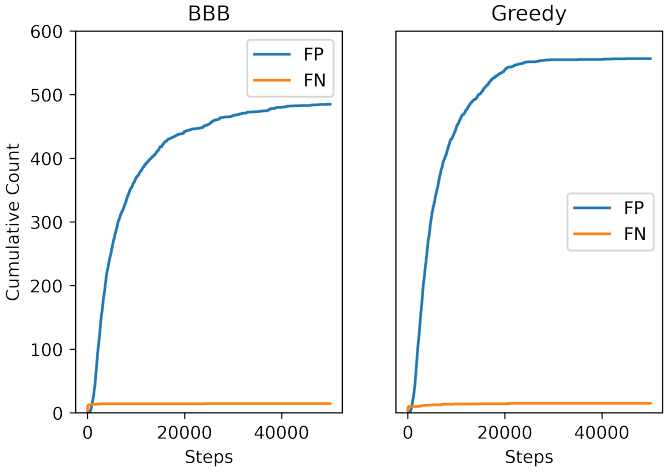
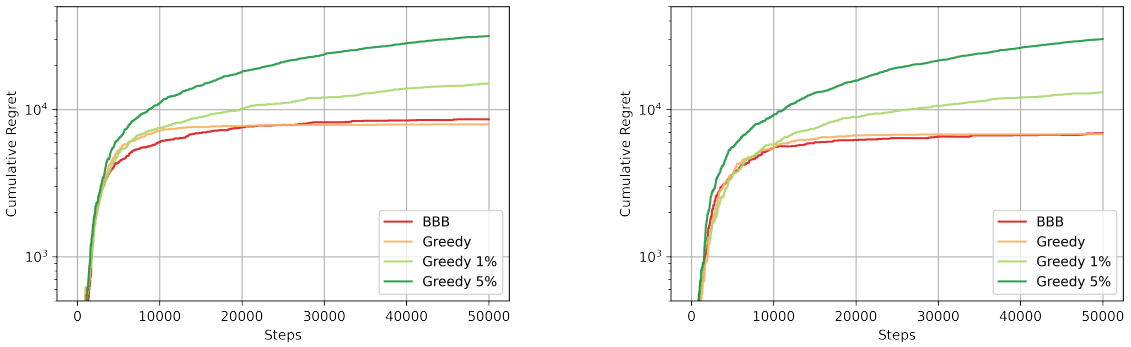


Figure 25: False Positive (FP) (i.e., eating poisonous mushrooms) and False Negative (FN) (i.e., not eating edible mushrooms) counts averaged over 3 independent runs.

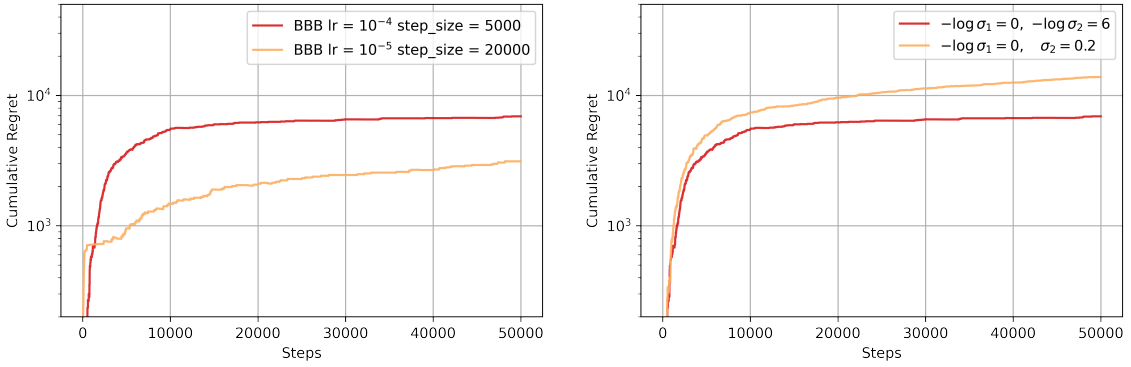
However, we did not find that the BBB agent consistently outperformed its Greedy counterpart. For example, Figure 26 shows individual runs for $\sigma_1 = 0.75$ and $\sigma_1 = 1.0$, with $-\log \sigma_2 = 6$ held constant, where the BBB agent converged more slowly and incurred higher cumulative regret. This may, in part, be due to the stochastic nature of the environment and training process. Further, we found that the learning rate and prior had a significant impact. Figure 27a shows that 50,000 steps were not enough for the agent to converge using a learning rate of 10^{-5}. Figure 27b demonstrates that a large prior variance may induce the agent to explore for longer than necessary, delaying its rate of convergence. Conversely, when the prior variance is too small the BBB agent may be prevented from exploring, as a result of a narrower weight distribution leading to lower variance in the predicted returns.



(a) BNN with MoGP($\sigma_1 = 0.75$, $\sigma_2 = e^{-6}$)

(b) BNN with MoGP($\sigma_1 = 1.00$, $\sigma_2 = e^{-6}$)

Figure 26: Example individual runs where the Greedy agent outperformed the BBB agent.



(a) Different learning rates and step sizes

(b) Different priors

Figure 27: Comparison of BBB agents' cumulative regret across different parameterisations.

5 Conclusion

In this report, we implemented the *Bayes by Backprop* algorithm proposed by Blundell, *et al.* in [2] and replicated their experiments. Our results were largely in agreement with the original paper's, finding that BNNs: appropriately express uncertainty in unseen parts of the input space; outperform regular DNNs under the same hyperparameter configurations; can be pruned heavily without affecting accuracy; and make better choices in the early stages of RL based tasks. Through our extensions, we found that BNNs are near-perfectly calibrated and significantly more robust than regular DNNs to adversarial inputs. Additionally, whilst the *local reparameterisation trick* was observed to reduce variance and improve computational efficiency, it was not observed to accelerate convergence in a simple regression task.

We believe that the clean and professional structuring of our implementation significantly aided development and debugging, and reduced collaboration overhead. We did, however, make the time consuming mistake of assuming there was a bug in our code when, in fact, we simply had not trained our models for long enough. The training time of the BNN also lengthened the feedback cycle, particularly in RL and classification tasks, making it impossible to run as many experiments as we would have liked.

In turn, if we had more time, we would seek to further explore and exploit the efficiencies of the LRT. Moreover, we would evaluate the BNN on more complex datasets, assessing its computational and accuracy trade-offs on larger input spaces with sparse signals and/or imbalanced inputs. Finally, we would evaluate different variational posterior families, assessing their impact on confidence, calibration, accuracy, and robustness.

6 References

[1] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and Harnessing Adversarial Examples," *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, Dec. 2014. arXiv: 1412.6572. [Online]. Available: https://arxiv.org/abs/1412.6572v3.

[2] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight Uncertainty in Neural Networks," *32nd International Conference on Machine Learning, ICML 2015*, vol. 2, pp. 1613–1622, May 2015. arXiv: 1505.05424. [Online]. Available: https://arxiv.org/abs/1505.05424v2.

[3] D. P. Kingma, T. Salimans, and M. Welling, "Variational Dropout and the Local Reparameterization Trick," *Advances in Neural Information Processing Systems*, vol. 2015-January, pp. 2575–2583, Jun. 2015, ISSN: 10495258. arXiv: 1506.02557. [Online]. Available: https://arxiv.org/abs/1506.02557v2.

[4] A. Gelman, J. Carlin, H. Stern, D. Dunson, A. Vehtari, and D. Rubin, *Bayesian Data Analysis, Third Edition*, 3rd editio. CRC Press, 2013.

[5] B. Lambert, *A Student's Guide to Bayesian Statistics*. Los Angeles : SAGE, 2018, 2018, ISBN: 9781473916364.

[6] W. R. Thompson, "On the Likelihood that One Unknown Probability Exceeds Another in View of the Evidence of Two Samples," *Biometrika*, vol. 25, no. 3/4, p. 294, Dec. 1933, ISSN: 00063444. DOI: 10.1093/biomet/25.3-4.285. eprint: https://academic.oup.com/biomet/article-pdf/25/3-4/285/513725/25-3-4-285.pdf. [Online]. Available: https://doi.org/10.1093/biomet/25.3-4.285.

[7] A. Paszke, S. Gross, F. Massa, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[8] PyTorch, *Training a Classifier — PyTorch Tutorials*, (Accessed on 30/03/2022). [Online]. Available: https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html (visited on 03/30/2022).

[9] Nutan, *PyTorch Convolutional Neural Network With MNIST Dataset*, (Accessed on 30/03/2022). [Online]. Available: https://medium.com/@nutanbhogendrasharma/pytorch-convolutional-neural-network-with-mnist-dataset-4e8a4265e118 (visited on 03/30/2022).

[10] J. Antoran, S. Markou, and I. Qing, *Bayesian Neural Networks*, https://github.com/JavierAntoran/Bayesian-Neural-Networks/, 2020.

[11] D. Kelshaw, *Weight Uncertainty*, https://github.com/danielkelshaw/WeightUncertainty/, 2020.

[12] S. Mayur, K. Kollnig, L. Delaney, and G. Couairon, *Weight Uncertainty in Neural Networks*, https://github.com/saxena-mayur/Weight-Uncertainty-in-Neural-Networks/, 2019.

[13] K. Shridhar, P. Raikwar, P. Mehta, *et al.*, *PyTorch Bayesian CNN*, https://github.com/kumar-shridhar/PyTorch-BayesianCNN/, 2021.

[14] T. Liu, C. Murray, and E. Persky, *Bayesian Neural Networks*, https://github.com/tennisonliu/bayesian-neural-network/, 2021.

[15] D. P. Kingma and J. L. Ba, "Adam: A Method for Stochastic Optimization," *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, Dec. 2014. DOI: 10.48550/arxiv.1412.6980. arXiv: 1412.6980. [Online]. Available: https://arxiv.org/abs/1412.6980v9.

[16] D. P. Kingma and M. Welling, "Auto-Encoding Variational Bayes," *2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings*, Dec. 2013. arXiv: 1312.6114. [Online]. Available: https://arxiv.org/abs/1312.6114v10.

[17] Y. LeCun and C. Cortes, "Mnist handwritten digit database," *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, vol. 7, 2010.

[18] *Neural networks - explanation of spikes in training loss vs. iterations with adam optimizer*, `https://stats.stackexchange.com/questions/303857/explanation-of-spikes-in-training-loss-vs-iterations-with-adam-optimizer`, (Accessed on 30/03/2022).

[19] D. Dua and C. Graff, *UCI machine learning repository*, 2017. [Online]. Available: `http://archive.ics.uci.edu/ml`.