Project Machine Learning :
— Milestone 2 —

Denis Semko

February 28, 2023

1 Introduction

This project consists in implementing the methods presented by Blundell et al. in the paper **Weight Uncertainty in Neural Networks** (2015). The algorithm, called *Bayes by Backprop* (**BBB**), is implemented and empirically evaluated on a classification and regression task; the performance of the so-called Bayesian neural networks (**BNN**s) is compared to "vanilla" multilayer perceptrons (**MLP**s).

In Milestone 1, the regression and classification datasets were inspected and preprocessed, baseline MLPs were defined and trained and the BBB algorithm was completely implemented, assuring that first results were already obtained on the BNN. Without any hyperparameter optimization and only having trained for 8 epochs, we managed to drive the accuracy to 83.1% for **classification** on MNIST. Regarding the **regression** task, the same BNN (tweaked for regression) was trained and some uncertainty was showcased as per Figure 1a. Ideally, the data in the training data interval (light grey) should be inferred with very high confidence. On the contrary, as soon as the BNN must predict on data out of its training distribution, it must behave very cautiously with an ever decreasing confidence (Figure 1b).

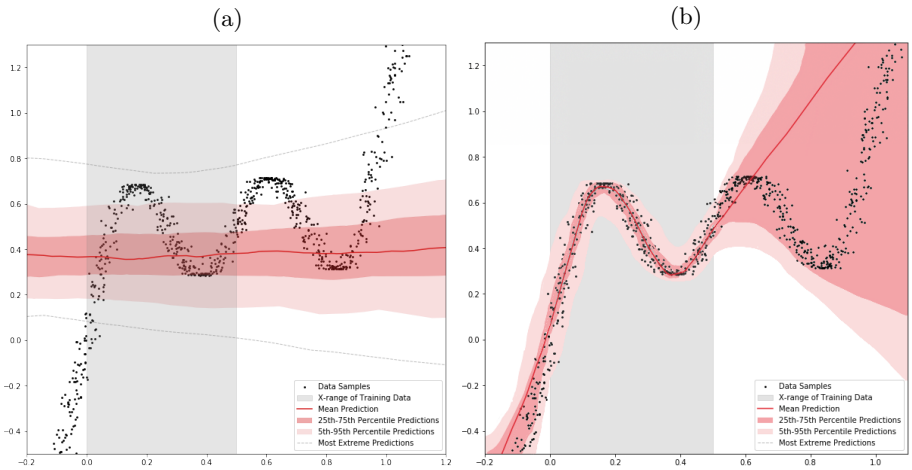


Figure 1: Graph (a) shows the results obtained in MS1 for regression. After replicating all settings mentioned in Blundell et al. and fine-tuning the set of hyper-parameters, we would ideally like to see a result akin to the hand-drawn example in (b).

In Milestone 2, the experiments from the original paper are recreated and the performance of BNNs are compared to MLPs. For this, the provided Grid Engine cluster has been used extensively. The results are then discussed and prospects are being made for Milestone 3.

2 Bayes by Backprop: A quick overview

Before analysing the hyperparameter settings more closely, one must have a clear outline of the model in order to identify which hyperparameters are mutable and also what their impact could be on the model results. At the end of this section, a non-exhaustive list is compiled with all the hyperparameters that should be looked into.

In the BBB algorithm, a neural network is trained not to obtain the optimal set of weights $\boldsymbol{w}$ (and biases), but to model a posterior probability distribution $P(\boldsymbol{w}|D)$ that would characterize every w_i with the distribution parameters θ_i . Through variational Bayesian methods, we approximate the true posterior $P(\boldsymbol{w}|D)$ with $q(\boldsymbol{w}|\theta)$, the variational posterior. In that way, the BNN better represents the uncertainty in the data by replicating it in the weights, as compared to a deterministic MLP.

The most common approach of variational Bayes minimizes the Kullback-Leibler divergence between $P(\boldsymbol{w}|D)$ and $q(\boldsymbol{w}|\theta)$, denoted as $\mathcal{F}(D,\theta)$. $\mathcal{F}(D,\theta)$ is the loss function to our BNN and is later referred to as the Expected Lower Bound (ELBO). Using the Gaussian Reparameterization Trick, we are able to approximate it as follows :

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

Notice that the two first terms are model-dependent (known as the complexity cost) while the third term is data-dependent (known as the likelihood cost). By iteratively minimizing (1), we approximate the optimal set of parameters $\boldsymbol{\theta}$. In order to fully define $\mathcal{F}(D,\theta)$, the prior distribution $P(w)$ needs to be modelled. Blundell et al. present the Spike-and-Slab prior as follows:

$$P(w) = \Pi_j \pi \mathcal{N}(\boldsymbol{w}_j|0,\sigma^2_{spike}) + (1-\pi)\mathcal{N}(\boldsymbol{w}_j|0,\sigma^2_{slab}) \tag{2}$$

At last, the authors opt for a mini-batch optimization associated with a re-weighting scheme. Suppose the optimization step is broken down into M randomly-split, equally-sized subsets, the ELBO (batch cost) is defined as:

$$\mathcal{F}_i(D_i,\theta) \approx \boldsymbol{\pi_i}\left[\sum_{i=1}^{n} \log q(\boldsymbol{w}^{(i)}|\theta) - \log P(\boldsymbol{w}^{(i)})\right] - \log P(D_i|\boldsymbol{w}^{(i)}) \tag{3}$$

for mini-batch $i = 1, 2, ..., M$. This allows for an adaptive weighting scheme with $\boldsymbol{\pi_i} = \frac{2^{M-i}}{2^M-1}$, ascribing more weight on the prior-dependent part of the ELBO at first, and a gradual shift to the data-dependent part as training progresses through the epoch.

Now that the cost $\mathcal{F}(D,\theta)$ is fully defined, we will go over the optimization step that allows to update the variational parameters $\theta = (\mu, \rho)$ in order to reduce the loss.

As proposed by the authors, the weights $\boldsymbol{w}$ are generated in a shift and scale manner, according to the current parameter values θ :

1. Sample $\epsilon \sim \mathcal{N}(O, I)$
2. $\sigma = log(1 + exp(\rho))$
3. Let $\mathbf{w} = \mu + \sigma \cdot \epsilon$

The $\boldsymbol{w}$ can be sampled N times, which results in a sum of N different ELBOs:

$$ELBO = \sum_{i=1}^{N} ELBO_i \tag{4}$$

Ultimately, the gradient of the ELBO $f(\boldsymbol{w},\theta)$ w.r.t μ and ρ can be computed and an update of θ is carried out. This corresponds to a simple gradient update step where the error is backpropagated.

Here follows a concise list of tunable hyperparameters of a BNN:

- Type of prior : Spike-and-Slab, simple Gaussian, ...
- π in the Scale Mixture Prior (Spike-and-Slab)
- σ_{spike} and σ_{slab} in the Scale Mixture Prior (Spike-and-Slab)
- Number/size of mini-batches M
- Number of Monte Carlo samples N (training & validation)
- The optimization algorithm and its hyperparameters (e.g. learning rate α)
- Parameters initialization : $\theta^0 = (\mu^0, \rho^0)$
- Classic MLP parameters: # hidden layers, # neurons, activation functions
- The noise-parameter in the regression likelihood cost function σ_{noise}

3 Experiments

In the following we present our results when recreating the regression and classification experiments by Blundell et al.. Each model consists of an input- and output layer, as well as a hidden layer in between with varying amounts of hidden linearly rectified units (ReLU). The datasets have been described in detail in Milestone 1, as well as the baseline MLP models, how the BNNs have been implemented, and how they employ the BBB algorithm.

3.1 Regression

The regression task at hand is defined as:

$$y = x + 0.3sin(2\pi(x + \epsilon)) + 0.3sin(4\pi(x + \epsilon)) + \epsilon \tag{5}$$

with $\epsilon \sim \mathcal{N}(0, 0.02)$. X-values are randomly drawn from a uniform distribution $X \sim \mathcal{U}(0, 0.5)$ for the training procedure. For evaluation, 1000 datapoints are generated from the same function, except the x-values are drawn from $X \sim \mathcal{U}(-0.2, 1.1)$ (see Figure 2).

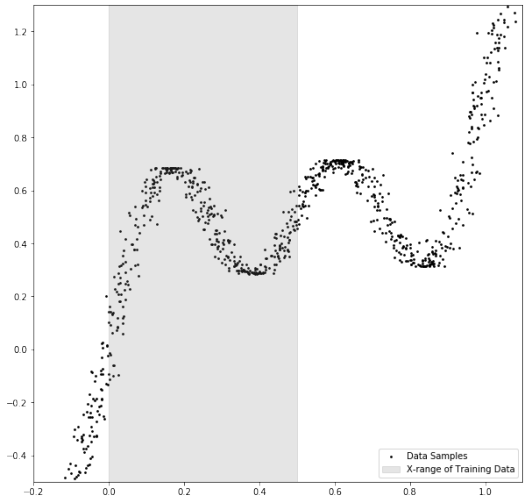


Figure 2: Randomly generated data from our curve function (5).

Figure 3 shows Blundell et al.'s resulting median predictions and interquartile ranges of a BNN (left), and an ensemble of MLPs trained on the data (right).

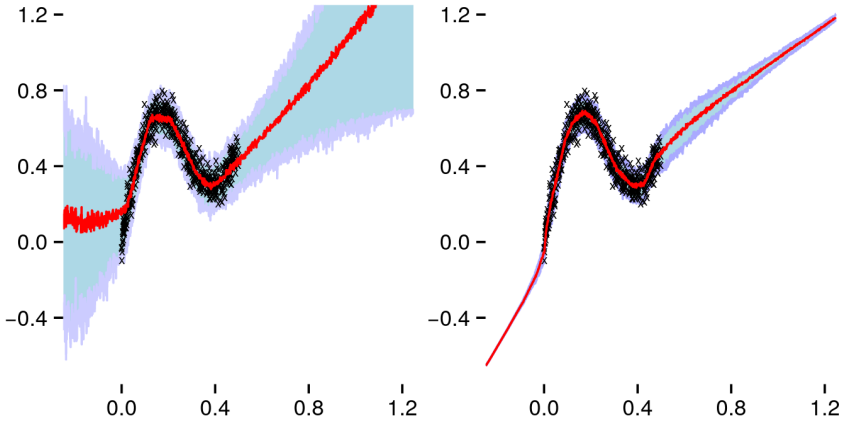


Figure 3: The authors' regression results from the original paper.

In the past Milestone we recreated the plot for ensembles of "vanilla" MLPs with and without dropout, see Figure 4. We were able to observe that dropout begins to address some of the disadvantages of standard MLPs without dropout, by increasing the confidence interval for data points outside of the training distribution, but it still left much to be desired.

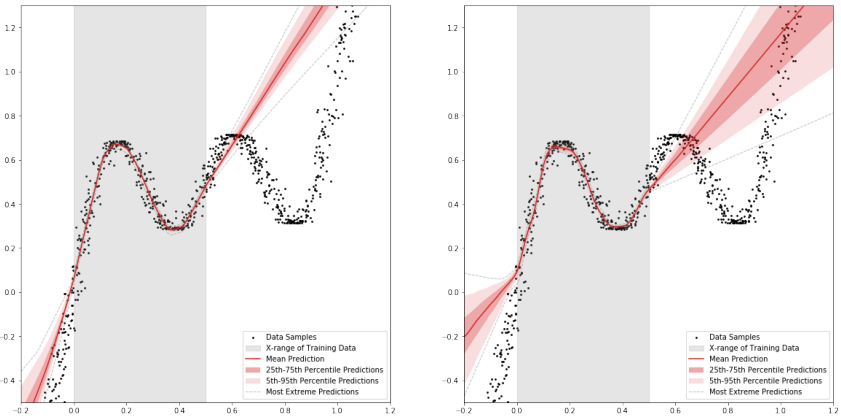


Figure 4: "Vanilla" MLP ensembles from Milestone 1.

Blundell et al. have not specified the hyperparameter configuration which led to the published results, we thus set out to find the hyperparameters which result in a model that best encompasses the points outside of the training data range with its 5th-95th percentile prediction interval. Using our BBB implementation from Milestone 1 we were able to train the model depicted in Figure 5.

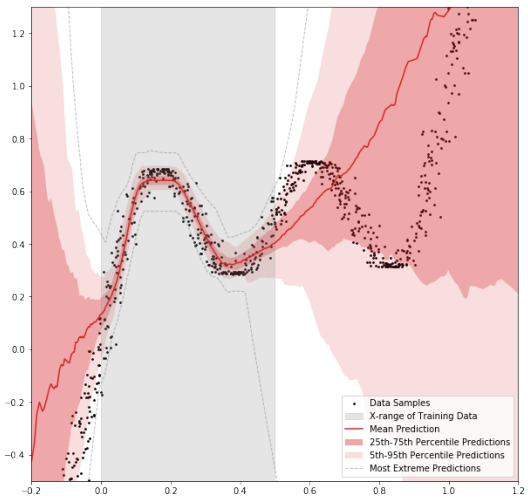


Figure 5: Our final, trained BNN

We trained the model using Adam and a learning rate of 10^{-3} for 1500 epochs. Rather surprisingly for us, a simple Gaussian prior delivered very similar results to the more "fancy" Spike-and-Slab prior proposed by Blundell et al., moreover, even lower ELBO values were reached for the simple Gaussian case. A possible explanation for that can be the fact that all the uncertainty in the data-generating function (5) comes from ϵ, which is normally distributed, and thus a simple Gaussian best describes that uncertainty. Another factor to keep in mind is the fact that the Spike-and-Slab prior has 3 hyperparameters; the standard deviations of the "spike" and the "slab" Gaussians (σ_{spike} and σ_{slab}), as well as the coefficient π which combines both. As a result, finding the best fitting 3 hyperparameters becomes very complex, as they all interdepend on each other as well. Meanwhile the more simple Gaussian prior has only one hyperparameter, its own standard deviation, tuning which didn't result in any major observable differences for us. We thus set the variance of the Gaussian to 1.

One of the most crucial hyperparameters turned out to be the noise-parameter σ_{noise} for the likelihood cost of the ELBO loss function (1) (for regression); described in detail in Milestone 1. As an illustration we have trained the same model with $\sigma_{noise} \in \{0.01, 0.1, 1\}$, the results can be observed in Figure 6. A "low" σ_{noise} of 0.01 displays overconfidence reminiscent of an ensemble of "vanilla" MLPs. A "high" σ_{noise} of 1 leads to poor inference: all points have a high likelihood, and the model explains the variation in the data through this high noise. A σ_{noise} of 0.1 seems to strike a middle ground, which is why we chose this as our final value.

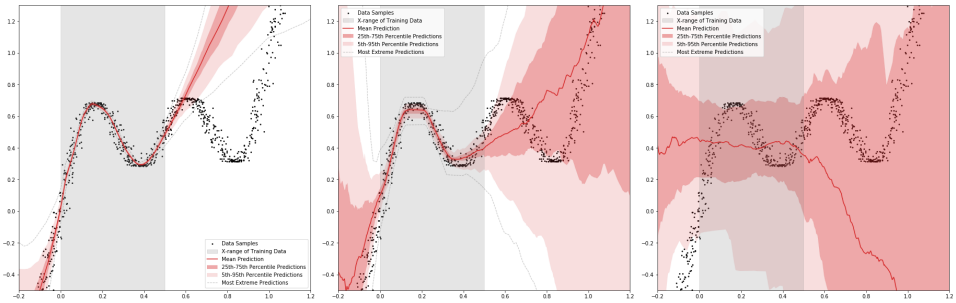


Figure 6: Exploring the effect of σ_{noise} in the likelihood function. Values 0.01, 0.1, 1 from left to right.

3.2 Classification

The classification task at hand consists in training a feed-forward neural network on the flattened MNIST dataset. 784 ($= 28 \times 28$) input features are fed to the model, and 10 output neurons return the probabilities of belonging to a certain class. Milestone 1 looks at the dataset in depth.

While Blundell et al. specify a grid of hyperparameters which they explored, they do not present the exact configuration which resulted in their publication. We reused the batch-size of 128 and Adam with a learning-rate of 10^{-3} for optimization. We found that learning rate decay/scheduling is crucial, because BNNs tend to diverge quickly after reaching its minimum test error. Training was fixed to 600 epochs, as is done in the original paper. Completely training one BNN on the provided cluster took us between 8 and 12 hours (for 2 ELBO training samples), depending on the number of hidden BNN units and the graphics processing unit (GPU) assigned for training (we recall training on following Nvidia GPU models: A100, V100, P100, TITAN X, RTX 2070).

Another very important parameter which impacts training time is the amount of ELBO samples drawn during training; Blundell et al. used 1, 2, 5 and 10 ELBO samples as hyperparameters in their experiments. Each additional ELBO sample essentially results in training one additional neural network, if one considers a BNN to be a special kind of ensemble of "vanilla" MLPs.

We did not find any significant improvement in the test error or in convergence when going up from 2 to 5 or 10 ELBO samples, but training stability improved considerably when going up from 1 to 2 training ELBO samples in our preliminary (not full 600 epoch) experiments. Hence we fixed this hyperparameter to be 2, thus trimming away a very computationally expensive dimension which we would need to examine and iterate over during our grid search.

For the validation step, which takes much less time than the training step, we chose 10 ELBO samples in order to get an accurate prediction of the true accuracy of the model.

Analogously to Blundell et al. we inspected both, a simple Gaussian prior and a more advanced Spike-and-Slab prior. For the Gaussian prior we used the standard normal distribution. For the Spike-and-Slab prior we searched over the grid of parameters explored in the paper: $\pi \in \{0.25, 0.5, 0.75\}$, $-log\,\sigma_{spike} \in \{0, 1, 2\}$, $-log\,\sigma_{slab} \in \{6, 7, 8\}$. In our experiments we found the best performing parameters to be $\pi = 0.5$, $-log\,\sigma_{spike} = 0$, $-log\,\sigma_{slab} = 8$

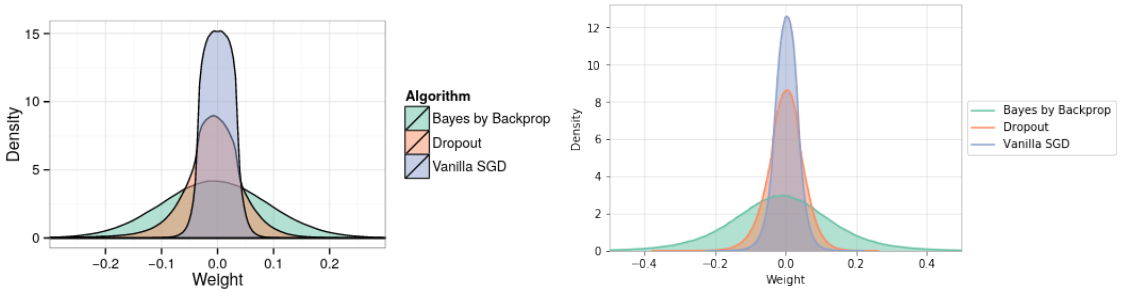
As per the original paper, we evaluated our models on the 10,000 samples from the MNIST test dataset. Our accuracy results compared to Blundell et al.'s results can be observed in Table 1. In alignment with the original paper we observed that in a basic classification task BNNs outperform vanilla MLPs, and perform roughly as good as MLPs with dropout. The attained test errors are not perfectly in line with the author's experiments, though. We attained considerably lower test errors for all experiments - up to 27 basis points for MLPs, 17 basis points for MLPs with dropout, up to 8 basis points for BNNs with Spike-and-Slab prior and most surprisingly up to 75 basis points for BNNs with a Gaussian prior. The authors did not specify their parameters used for the Gaussian prior, an explanation for this large discrepancy may thus be the fact that they did not choose the standard normal $\mathcal{N}(0, 1)$, but a different Gaussian prior.

Method	**# Units/Layer**	**# Weights**	**Blundell et al. Test Error**	**Our Test Error**
MLP	400	500k	1.83%	1.63%
	800	1.3m	1.84%	1.64%
	1200	2.4m	1.88%	1.61%
MLP, dropout	400	500k	1.51%	1.37%
	800	1.3m	1.33%	1.24%
	1200	2.4m	1.36%	**1.19%**
BBB, Gaussian	400	500k	1.82%	1.38%
	800	1.3m	1.99%	1.29%
	1200	2.4m	2.04%	1.29%
BBB, Mixture	400	500k	1.36%	1.30%
	800	1.3m	1.34%	**1.19%**
	1200	2.4m	**1.32%**	1.24%

Table 1: The lowest classification test error rates for each different model we trained on the MNIST dataset.

Given these large improvements over the original paper one also needs to keep in mind the phenomenon of overfitting to the test set, given that we have been optimizing the hyperparameters on the same test data set which we use for final accuracy evaluation. Though we have not done any excessive hyperparameter optimization, having "only" used the grid of hyperparameters provided by the authors.

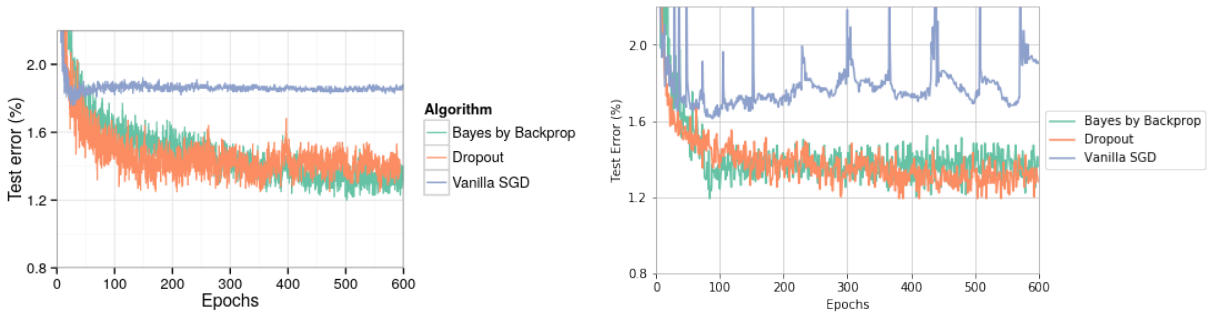
The distributions of the trained weights for the best performing BNN, MLP, and MLP with dropout models in Table 1 are plotted in Figure 7b, with the results of Blundell et al. for comparison to its left (7a). Our results are in line with the original author's observation: an MLP with dropout has a greater range of weights than a "vanilla" MLP, with BBB using the greatest range of weights.



(a) Blundell et al. (b) Our result

Figure 7: Trained weights distributions, comparison.

Our model's learning curves (test error as a function of training epoch) can be observed and compared to Blundell et al.'s results in Figure 8. Our results are generally in line with the original paper; the BNN and MLP with dropout seem to perform similarly, both much better than a vanilla MLP - though our BNN converges much faster than in the paper - a likely explanation is the authors used a lower learning rate for the Adam optimizer. Another noticeable difference is the lower test error for the vanilla MLP, and its occasional spikes in the test error.



(a) Blundell et al. (b) Our result

Figure 8: Test errors during 600 epochs of training, comparison.

To recreate the weight pruning experiment from the paper we reused our best-performing classification BNN from above. We replaced the mean and variance of the posterior distribution with zeros for weights with a Signal-to-Noise ratio ($|\mu_i|/\sigma_i$) below a moving threshold. This threshold iteratively increased from covering 0% of all weights up to 100% of all weights, removing weights with the smallest Signal-to-Noise ratio first. We drew 25 samples at each iteration. The results can be observed in Figure 9.

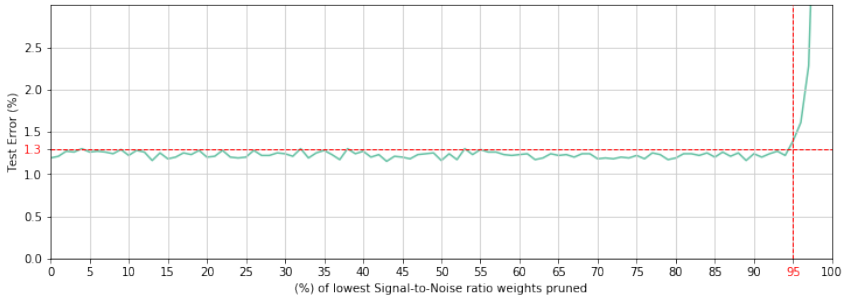


Figure 9: Consequence of weight pruning on test error.

Once again our results are in accordance with the results by Blundell et al.. Our test error did not go above 1.3% until 95% of the weights have been pruned, after which the accuracy of the model deteriorated significantly (test errors of 1.39%, 1.61%, 2.28%, 5.12%, 28.92%, 90.25% for respectively 95%, 96%, 97%, 98%, 99%, 100% of weights removed).

Interestingly, the test error hit even lower values than in the the BNN training experiment above (cf. Table 1) multiple times during the pruning experiment (e.g. test error of 1.15% when removing the 43% lowest Signal-to-Noise ratio weights).

4 Conclusion

After having implemented the BBB algorithm and the baseline models proposed by Blundell et al. in Milestone 1, we have trained the BNNs and their MLP counterparts until convergence with some hyperparameter optimization, and replicated the regression and classification experiments by the authors.

Our results generally fall in line with the original paper. We were able to observe how BNNs assert uncertainty in "out of distribution" data for regression tasks, how BNNs can outperform MLPs accuracy-wise in the same classification setting, and how classification BNNs can be pruned extensively without increasing (potentially even decreasing) the test error.

The training time required to train a BNN, even on the newest cutting-edge GPUs, very substantially lengthened the feedback circuit - most importantly in the classification task - rendering it inconceivable to run as many experiments as we would have preferred. Just by nature of the randomness of neural networks, and especially BNNs - where additionally random weight samples are drawn at each layer - it would be reasonable to train the same BNN with the same hyperparameters with multiple different random seeds for an accurate evaluation of the model performance. But in a setting where training a BNN takes 8-12 hours and the limited time available for Milestone 2, that endeavor becomes impossible.

5 Short Outlook

Milestone 3 entails going "beyond" the paper. While we need to discuss with our supervisor what this means exactly for our project, here are some topics which we came up with that could be tackled next:

- Reading about, implementing and evaluating the Local Reparameterisation Trick in the same environments as above
- Evaluating more complex datasets (imbalanced classes, more input dimensions...)
- Evaluating different priors and posteriors
- Further evaluating the results from Milestone 2, e.g. with receiver operating characteristic (ROC) curves, calibration curves, looking further into pruning, etc.
- Evaluating the effect of adversarial attacks on BNNs