A Note on Prior Selection in Bayesian Neural Networks

Khachatur Mirijanyan, Leif-Martin Sæther Sunde, Håkon Olav Torvik

May 20, 2022

Abstract

In modern deep machine learning, Bayesian neural networks have provided promising results compared to frequentist neural networks in certain applications. However these have historically almost unanimously utilized a Gaussian prior over their parameters. In this paper we investigate these BNN assumptions as well as the potential benefit of applying a Hyperspherical prior over the parameters. We compare the performance of a novel prior on a bayesian neural network with that of a gaussian and frequentist neural network. We evaluate the differing performances of the architectures on simple datasets. The performance between the gaussian and frequentist networks were similar. However, our novel prior network performed much worse, with all its predictions collapsing towards the mean of the targets, suggesting pathological posterior collapse for this architecture.

1 Introduction

The frequentist neural networks (FNN) neural network is an expressive architecture that can capture a wide variety of data relationships. These often require regularization, like a $\mathcal{L}^2$ penalty in their loss, and dropout. One of the alternative approaches is to instead include a prior and a posterior over the parameters of the network and update through bayesian posterior inference (*Vincent et al. 2022*) [3]. Such Neural Networks are called Bayesian Neural Network (BNN).

Historically, this prior has been Gaussian, which replaces the need for both dropout and $\mathcal{L}^2$ regularization. However new research suggests the use of a different prior and resulting posterior over the parameters, depending on the structure of the data.

Because BNNs require significantly more compute, due to the required sampling from a distribution in every forward pass, they are not as widely used as FNNs. Prior work on BNNs have shown that they can be trained to perform at least as well as FNNs (*Shridhar et al. 2019*) [6]. Newer work indicates that a good choice of prior for the Bayesian layers can make the BNN outperform the FNN, when similar compute resources are used during training, especially when less data is available.

This note will focus on assessing our team's efforts in using a Hypershperical Uniform distribution as prior with a von Mieses-Fisher posterior, estimated though variational inference. We will compare the predictive power to that of using Gaussian priors, and that of a similar FNN.

2 Theory

2.1 Bayesian Neural Networks

In Bayesian inference, we introduce a prior $p(\vec{\theta})$ over the parameters we wish to estimate. This prior should in theory represent the investigators prior beliefs about the parameter, hence the name. This prior belief is then updated by the observed data using Bayes' Theorem to find the posterior $p(\vec{\theta}|\mathcal{D})$.

This is done in BNNs, where we introduce our prior belief over the entire parameter space, and update this through bayesian posterior inference. In practice the posterior predictive that results from this is intractable, so approximation methods will have to be utilized. The most precise method for evaluating this is through Markov Chain Monte Carlo. However, due to the massive computational cost, we are approximating the posterior in a less precise, namely through variational inference.

2.2 Variational Inference

Variational inference is a technique for approximating the posterior, which is less precise than MCMC, but saves a lot of computation. Mathematically we are minimizing the KL divergence, a statistical distance for probability distributions, between our posterior and the true posterior. Minimizing this distance can be shown to be equivalent to minimizing the KL divergence between our approximated posterior and the prior and maximizing the likelihood (*Kingma et al. 2015*) [4].

The KL-loss here can be viewed as a regularization parameter that replaces the $\mathcal{L}^2$ loss term in FNNs. We hope that this regularization will also be a more appropriate regularization than the $\mathcal{L}^2$, since we can make a more specific prior distribution that the parameters are regularized towards, rather than just inducing a bias to reduce parameter distance from 0. Our hope is that we can find a prior such that the parameter space is regularized towards something more helpful, while also managing to replace a computational step, batch normalization. This is done in order to mitigate some of the additional computational cost of BNN compared to FNN.

2.3 Hyperspherical Uniform Prior

In choosing such a prior we are making an investigation into the following class of priors suggested recently, the radial directional distribution. (*Vincent 2021*)[2]

$$\vec{\theta} = \theta_r \vec{\theta}_d, \quad \theta_r \sim p_{\text{rad}}(\theta_r) \quad \vec{\theta}_d \sim p_{\text{dir}}\left(\vec{\theta}_d\right). \tag{1}$$

This distribution separates the direction from the magnitude, and hence allows us to train them separately. In a frequentist neural network, batch-normalization is used to draw the length of the vector weights closer to 1, and we incorporate this by setting the radial component $\theta_r = 1$, placing our prior on the hypersphere.

We use an uninformative prior for the directional component, i.e. the uniform, yielding the prior we wish to evaluate, the Hyperspherical Uniform prior. This should make each forward pass pass its vector on, with the norm of the vector remaining the same and in theory, remove the need for batch-normalization. The posterior distribution that follows from this is the Von Mieses-Fisher distribution, which can be seen as

the hyperspherical extension of the normal distribution. Hence we hope that our prior will inherit the benefits of the Gaussian prior, while adding the benefit of replacing batch normalization.

In addition to this, the stochastic nature of the net is obviously preserved. In each forward pass the size of each weight varies, meaning some of the connections become very unimportant, similarly to dropout. This means that the effect of forcing learning across several of the connections is preserved, thus also replacing dropout.

Lastly, recent results indicate that a Variational Autoencoder with a Hyperspherical Uniform prior for density estimation over a dataspace with latent hyperspherical structure can perform better than its Gaussian counterpart (*Davidson et al. 2018*)[1]. We have not investigated the effect of this possible advantage in our datasets.

2.4 The Von Mises-Fisher posterior

The probability density function of the von Mises Fisher distribution for the random p-dimensional unit vector $\mathbf{x}$ is given by: $f_p(\mathbf{x}; \boldsymbol{\mu}, \kappa) = C_p(\kappa) \exp\left(\kappa \boldsymbol{\mu}^\mathsf{T} \mathbf{x}\right)$ where $C_p(\kappa)$ is defined by

$$C_p(\kappa) = \frac{\kappa^{p/2-1}}{(2\pi)^{p/2} I_{p/2-1}(\kappa)}$$

where I_ν denotes the modified Bessel function of the first kind at order ν.

the normalization constant reduces to

$$C_3(\kappa) = \frac{\kappa}{4\pi \sinh \kappa} = \frac{\kappa}{2\pi(e^\kappa - e^{-\kappa})}.$$

with parameters $\boldsymbol{\mu}$ and κ as the mean direction and concentration parameter, respectively. The greater the value of κ, the higher the concentration of the distribution around the mean direction $\boldsymbol{\mu}$.

3 Methods

3.1 Implementation

We implemented a BNN framework in Python3 based on the PyTorch framework for neural networks. We use fully connected layers, where we sample weights from a running posterior that are reparameterized at every forward pass. The learnable parameters, W_μ and W_ρ, are used to generate the running posterior distribution F as

$$F = D\Big(\mu : W_\mu, \quad \sigma : \text{Softplus}(W_\rho)\Big),$$

where D is either the Normal distribution, or vMF, depending on what prior we use.

We also make use of the softplus, defined by

$$\text{Softplus}(x) = \log(1 + \exp(x))$$

as a smoother version of ReLU, that ensures that the variances of the distribution is positive, even though the learned parameters W_ρ are negative. The biases in the layers are non-Bayesian, to not unnecessarily increase required computation. Early tests also indicated that the model could be performing worse with Bayesian biases, so this was not further attempted. Between the layers, we use ReLU as nonlinear activation functions.

When using the Gaussian prior, we use the PyTorch implementation of the distribution. For the hyperspherical uniform and von Mises-Fischer distributions, we use the implementations from the paper (*Davidson et al. 2018*)[1]. Our code can be found at our GitHub page:
github.com/Haakooto/CS282_final_project.

3.2 Data

Two data sets were used to evaluate the efficacy of the model; data drawn for a simply generated sinusoidal as shown in Figure 1, as well as the carbon nanotubes dataset from the UCI Machine Learning Repository [5].

The sinusoidal data was was created as a bivariate sinus, $y = \sin(x_1 \cdot x_2)$, where x_1 was N equidistant points in $[-0.5, 0.5]$, and $x_2 \sim \mathcal{U}(-2, 2)$, also taken with N samples. The target y was generated from two independent variables since the implementation of the hyperspherical distributions does not allow for representations of one-dimensional spheres. The model was trained on both 32 and 200 generated observations to test the assumption that BNNs are superior to FNNs in low observation situations. For testing, 500 observations where similarly generated and tested against true values.

The carbon nanotubes data set contains 10721 initial and calculated atomic coordinates of carbon nanotubes and label as that is conducive to regression problems by the UCI Machine Learning Repository. These atomic coordinates for carbon atoms are generated using CASTEP, requiring an input of initial atomic coordinates (u, v, w) and chiral vector (n, m) and calculated coordinates ($\hat{u}$, $\hat{v}$, $\hat{w}$) where all values are real numbers. We attempted a univariate case where we predicted $\hat{w}$ using (u, v, w, n, m, $\hat{u}$, $\hat{v}$).

3.3 Experimental Procedure

Model Type	Learning Rate	Mean	Scale
FNN	.001	N/A	N/A
vMF-BNN	.00001	1	0.1
Gauss-BNN	0.1	0	0.1

Table 1: Values of hyperparameters that remained fixed for sine wave model testing

We ran experiments on both the sine wave data and the carbon nanotubes data. During testing, we found some hyperparameters that worked sufficiently well for each model on each of the datasets, and kept these fixed when performing the experiments. This included the learning rate for all FNN and BNN models, and also the mean and scale of the prior for the Gauss-BNN and vMF-BNN. These are summarized in Table 1. The number and sizes of the hidden layers for the FNNs varied.

We attempted to compare our BNNs to FNNs in a few different configurations for the sine data experiments. We tested all our models under both large and small training sizes as BNNs are supposed to perform better in low observation setting. We kept the model architecture fairly simple for all of our BNNs, with two hidden states and 20 nodes each running for 1000 epochs, and did so to save on computation time. We compared the BNNs to FNNs with the same architecture as well as FNNs with 6 hidden layers with 50 nodes each. The more complex FNNs were to test against models that used about as much computation time as the Gauss-BNNs.

The loss function for BNNs is expressed as a summation of the negative log likelihood and the KL-Divergence between the hypershperical uniform prior and the vMF learned posterior. We tested the efficacy of the KL-Divergence as a regularization parameter by disabling it some of our tests.

The metric used to examine the accuracy of the models was the R^2-score. The should give a number between 0 and 1, where 1 indicates perfect fit to data. In some cases the predictions can be so bad the R^2-score is negative. Since BNNs sample weights from the running posterior at each forward time, we passed every batch through m times, and took the average of the m predictions as the final prediction for each of the data points. We usually used $m = 500$. This also allowed us to find a measure for the uncertainty of our predictions by finding the standard deviation of the m predictions.

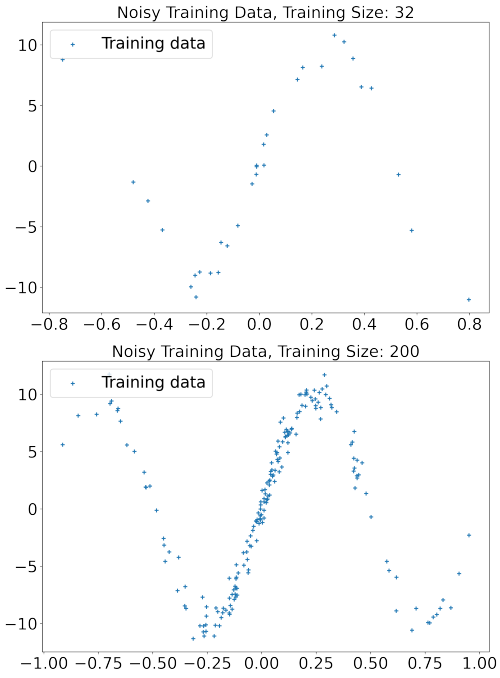

Figure 1: Visualization of sine wave training data. **Top**: 32 Observations, **Bottom**: 200 Observations

4 Results

4.1 Sinusoidal Experiments

The parameters and results of the experiments can be found in Table 2. For now, we only look at the FNN and Gauss-BNN results, leaving the vMF-BNN to the end. With 32 training observations, the Gauss-BNNs acheives a higher R^2-score than the FNNs, with a high of 0.42, against 0.16. When the observations are increased to 200, we see that the deeper FNN and Gauss-BNNs achieve very similar results, while the shallower FNN is significantly worse. However, the shallow FNN model requires significantly less compute. The deep FNN takes roughly as long to train as the Gaussian BNNs, so the results are more comparable as they account for amount of compute.

We plot the predictions on 500 test-points of the deep FNN and Gauss-BNN trained on 32 data-points in the upper and middle part of Figure 2. Notice that both models have incredible variance in the predictions, with a reduction in variability around the centre of the testing interval. This is because the the testing and training data generates more data points around the mean of $x = 0$. Thus the models, when trained on low observations, are seeing mostly observations around the mean.

Sinusoidal Testing Results					
Model Type	Train Size	Layers	Epochs	KL-On	R^2 Score
FNN	32	2	1000	N/A	0.1157
FNN	32	6	5000	N/A	0.1594
FNN	200	2	1000	N/A	0.551
FNN	200	6	5000	N/A	0.9401
Gauss-BNN	32	2	1000	False	0.4228
Gauss-BNN	32	2	1000	True	0.2858
Gauss-BNN	200	2	1000	False	0.9551
Gauss-BNN	200	2	1000	True	0.9406
vMF-BNN	32	2	1000	False	-0.0006
vMF-BNN	32	2	1000	True	-0.006
vMF-BNN	200	2	1000	True	-0.003
vMF-BNN	200	2	1000	True	-0.003

Table 2: The resulting R2 after testing a variety of models under a combination of possible hyper parameters on the generated sine wave data. Note that the BNN with a Gaussian Prior with low observations performs better than an FNN with low observations. Also shown are the difficulties of constructing a vMF-BNN with all R2 values being identical

4.2 Carbon nanotube data

Table 3 contains similar data as Table 2, but for the Carbon Nanotube data. The KL-divergence was always included in the loss of BNNs. In the case of more training data, all models, except for the Gauss-BNN achieved a R2-score above 0.98, while for less data, only the FNNs were able to get any good result, with R2 of 0.81 and 0.90 for the shallow and deep, respectively. while the Gauss-BNN scores 0.12.

4.2.1 vMF-BNN Result

The results from the vMF have been left to the end because they indicate something novel and interesting. Notice in Table 2 and Table 3 that the vMF-BNN performed poorly in all cases with R^2-score very close to 0 for all the different configurations. When we plot the prediction of the model in Figure 3, we see that the model finds the mean very quickly and only predicts this mean without much uncertainly. This is the case for both the data with mean 0, shown in the upper part of the figure, and data where the mean is not 0, shown in the middle part of the figure. The loss curve for the vMF-BNN in the carbon data is shown at the bottom of the figure. The loss is decreasing until a certain point, after which it is not changing. This is the point at which the model has found the mean of the data, and effectively stops learning.

Throughout our testing we noticed that the models were very sensitive to the parameter initialization of the prior. To confirm this theory, we replicated our best Gauss-BNN from Figure 2 with the priors initialized poorly. The predictions of the poorly initialized model were very inaccurate and showed that even a fairly good BNN will perform very poorly with bad prior initialization. Our KL-Loss function should theoretically be accounting for poor initialization, but it doesn't seem able to.

5 Discussion

Overall, the Gauss-BNN models were close to the FNN in most situations, but generally performed better in low observations settings. The vMF-BNN models on the other hand performed poorly in all scenarios by finding the mean of the data and only predicting within close bound of the mean. This behaviour could be described as a pathological posterior collapse.

Pathological posterior collapse means that the posterior distribution is not learning properly from the data, and is instead "collapsing" back to the shape of the prior in some form throughout the training. Our results for the vMF-BNN could suggest this is the case, as it predicts close to the mean of the target variable regardless of input. This behavior could come from a multitude of sources.

One such source could be erroneous implementation from our part of the KL-Divergence. However as we have tried two different implementations of the KL-loss and found the same results with both, we think this is unlikely. As our Gaussian BNN uses the same code except for the specific implementation of the distributions themselves and the KL-loss calculation while making well working predictions under appro-

Carbon Nanotube data				
Model Type	Test size	Layers	Epochs	R^2 Score
FNN	0.2	2	15	0.9936
FNN	0.2	6	15	0.9974
FNN	0.99	2	15	0.8135
FNN	0.99	6	15	0.9022
Gauss-BNN	0.2	2	15	0.9835
Gauss-BNN	0.99	2	15	0.1201
vMF-BNN	0.2	2	15	-0.0147
vMF-BNN	0.99	2	15	-0.0147

Table 3: Exeriments and R2-score performed with carbon nanotube data. The test size indicates the percentage of the all the points in the dataset were used in the test data. When the FNN has 2 hidden layers, each layer has 20 nodes, while in the case of 6 hidden layers, there ate 50 nodes in each. The BNNs have 20 nodes in each layer.

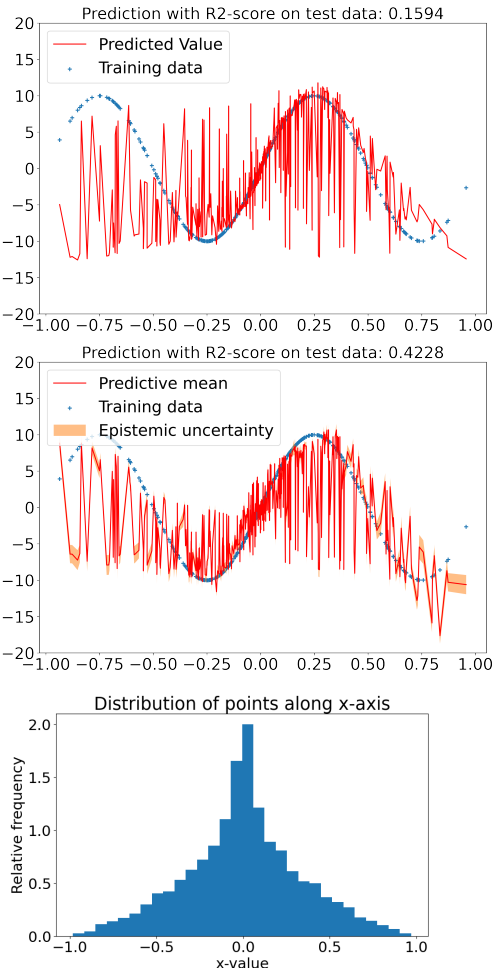

Figure 2: Prediction on test-data by **Top**: Deep FNN, and **Middle**: Gauss-BNN, both trained on 32 datapoints. **Bottom**: Distribution of datapoints along x-axis. This includes many more points than are being used, in order to show the shape of the distribution.

priate initialization. Another source could be that our initialization is sub-optimal. We consider this a more likely scenario, as our Gaussian BNN also exerted similar behaviour to our vMF-BNNs when the initialization was poor. A third scenario we consider equally likely is that the priors were properly initialization, but issues from the BNN architecture arise with a Hyperspherical Uniform prior. A BNN with this prior where mean and variance are trained simultaneously is especially prone to this pathological posterior collapse phenomenon.

For further investigations into this area we suggest training the mean network separately to reach some MAP solution. Afterwards we suggest training the variance component, or a hyper prior on the variance as another regularization term to avoid this behaviour. Whether this would actually yield improved results is beyond the scope of this note, but we look forward to seeing further investigations into this area.

6 Conclusion

Our results indicate propensity for pathological posterior collapse towards the mean, with R^2 close to 0, for BNN's with Hyperspherical Uniform prior. A specific initialization technique or adaptations to the fundamental architecture as suggested earlier may remediate this propensity.

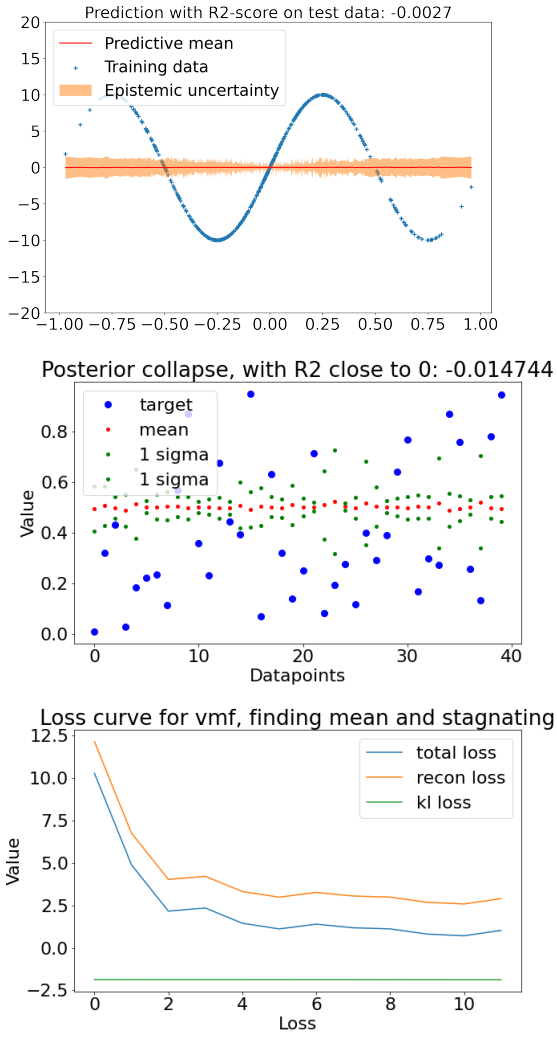

Figure 3: vMF-BNN predictions, **Top**: Sigmoid and **Middle**: Carbon-Nanotube data. Both these plots indicate pathological posterior collapse. **Bottom**: Loss-curve during training of the vMF on carbon nanotube data.

7 Contributions

Most of the work was done together, however we did specialize somewhat differently on our own in certain areas. Leif: Research and mathematical/conceptual work. Khachatur: Sinusoidal evaluation, implementation of KL-loss for vmf. Håkon Olav: Results for carbon nanotubes.

References

[1] Tim R. Davidson, Luca Falorsi, Nicola De Cao, Thomas Kipf, and Jakub M. Tomczak. Hyperspherical variational auto-encoders. 2018.

[2] Vincent Fortuin. Priors in bayesian deep learning: A review, 2021.

[3] Vincent Fortuin, Adrià Garriga-Alonso, Sebastian W. Ober, Florian Wenzel, Gunnar Rätsch, Richard E. Turner, Mark van der Wilk, and Laurence Aitchison. Bayesian neural network priors revisited, 2022.

[4] Diederik P. Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick, 2015.

[5] UCI Machine Learning Repository. Carbon nanotubes dataset, 2018.

[6] Kumar Shridhar, Felix Laumann, and Marcus Liwicki. A comprehensive guide to bayesian convolutional neural network with variational inference, 2019.