UNIVERSITY
OF OSLO

Master’s thesis

Spherical Priors for Bayesian Deep Learning

Bringing Bayesian Uncertainty Estimation and Robustness into Modern High Parameter Deep Learning by Parameters Scaling and Gradient Normalization in Bayesian Neural Networks through Prior Specification

Leif-Martin Sæther Sunde

Stokastisk Modellering, Statistikk og Risikoanalyse
60 ECTS study points

Department of Mathematics
Faculty of Mathematics and Natural Sciences

15th May 2023

Abstract

This thesis summarizes some of the important work and theory for understanding Bayesian Neural Networks. We include computational and theoretical aspects together, to show how they interplay. Afterward, we introduce the Hyperspherical Uniform prior to Bayesian Neural Networks. This prior replaces the need for Batch Normalization. Our experiments for smaller networks indicated the performance to be similar to the conventional Gaussian Priors. We argue that this prior could increase the performance significantly for deeper networks, and potentially assist interpretability.

A big thanks to my supervisors Geir Storvik and Aliaksandr Hubin.

Leif-Martin Sæther Sunde

15th May 2023

UNIVERSITY OF OSLO

UNIVERSITAS OSLOENSIS • MDCCCXI •

Thanks to Vincent Fortuin, who had no formal obligation to me, for guidance.

I must mention the course (Zhang, 2022) as particularly helpful for my understanding of deep learning. Special thanks to Xinyang (Young) Geng, Yuqing Du, and Olivia Watkins for aiding my understanding during many office hours.

An earlier less successful attempt at spherical learning by my fellow students and me in a graduate project at UC Berkeley Leif-Martin Sunde (2022). Lessons learned there has helped the work in this thesis.

I thank Terje Kvernes at IT operation at the Institute of Mathematics at UiO for helping me better understand Linux and its deployment on remote machines.

GitHub Link: `https://github.com/LeifMartin/BNN`

Contents

0.1 Terminology

- **Acronyms**:
  - **rv**: random variable.
  - **NN**: Neural Network.
  - **FNN**: Frequentist Neural Network.
  - **BNN**: Bayesian Neural Network.
  - **VI-BNN**: Variational Inference Bayesian Neural Network.
  - **pdf**: probability density function.
  - **VD**: Variational Distribution.
  - **VI**: Variational Inference.
- In classical statistical learning, the conventions for naming predictor and target variables vary. Predictor variables may be referred to by any of the following terms: "Covariate" (Continuous predictor variables), "Explanatory Variable" (Any predictor), "Confounding Variable" (Predictor variables that are dependent on other predictor variables) (O'Brien et al., 2020; Hastie et al., 2001).
- Whenever "parameter" is used without further explanation, the reader should assume this refers to all the parameters updated during model training. In Bayesian neural networks case, the μ (mean) and σ (sigma) vectors of means and variances for the variational distributions from which the weights and biases are sampled during forward passes.
- In Bayesian inference statisticians often refer to $p_{\mathcal{D}|\theta}(\mathcal{D},\theta)$ as "the model". However, "likelihood" is also often used, which is the term we will utilize since it cannot be confused with the whole Bayesian or frequentist neural network itself.
- For this thesis, ψ refers to both the weights $\mathbf{w}$ and biases $\mathbf{b}$ of a frequentist neural network (FNN) together. Alternatively, for the realizations of the pdfs of the weights and biases for a Bayesian neural network. $\boldsymbol{\Psi}$ is both the weights and biases of a Bayesian neural network (BNN) together. Being Bayesian, these are then random variables. Since we work with variational inference BNNs, $\boldsymbol{\Psi}$ is specifically the variational distribution set to approximate the true posterior of the weights and biases. θ refers to the parameters that compose the pdf's of $\boldsymbol{\Psi}$, or, equivalently, the parameters determining its distribution. $\mathbf{W}$ and $\mathbf{B}$ refer to the random variables for the weights and biases separately. $\mathbf{W}_j$ and $\mathbf{B}_j$ respectively refer to the weights and biases for layer j. While $\mathbf{W}_{j,i}$ and

$\mathbf{B}_{j,i}$ respectively refer to the weights and biases for inputting to node i in layer j.

- When we write $\nabla f(\mathbf{x})$ for a function $f : \mathbb{R}^m \to \mathbb{R}$, we mean the column vector. Such notation is conducive to deep learning by making the mathematical theory more analogous to the programming implementation. Explicitly we mean:

$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f(x)}{\partial x_1} \\ \frac{\partial f(x)}{\partial x_2} \\ \vdots \\ \frac{\partial f(x)}{\partial x_m} \end{bmatrix}$$

- When writing a subscript $f_n(\mathbf{x})$ this refers to function number n, meaning each $f_n(\mathbf{x})$ is a different function, although they probably have some relation since we denote them that way. $f(\mathbf{x})_n$ on the other hand, refers to the output of $f(\mathbf{x})$ along dimension n. When we talk about a Jacobian $J_f(\mathbf{x})$, where f has a vector codomain in addition to the vector domain, we find it more appropriate to use the denominator convention as we are working purely theoretically. Meaning:

$$J_f(\mathbf{x}) = \begin{bmatrix} \nabla f(\mathbf{x})_1^\top \\ \nabla f(\mathbf{x})_2^\top \\ \vdots \\ \nabla f(\mathbf{x})_3^\top \end{bmatrix} = \begin{bmatrix} \frac{\partial f(x)_1}{\partial x_1} & \frac{\partial f(x)_1}{\partial x_2} & \cdots \frac{\partial f(x)_1}{\partial x_m} \\ \frac{\partial f(x)_2}{\partial x_1} & \frac{\partial f(x)_2}{\partial x_2} & \cdots \frac{\partial f(x)_2}{\partial x_m} \\ \vdots & & \\ \frac{\partial f(x)_m}{\partial x_1} & \frac{\partial f(x)_m}{\partial x_2} & \cdots \frac{\partial f(x)_m}{\partial x_m} \end{bmatrix}$$

- Double Descent, or modern machine learning framework, refers to the same idea. The test error first worsens as the number of parameters increases and then gets better after an order of magnitude more parameters are added. Double Descent is primarily argued for in cases where datasets are massive. This phenomenon is explained more in Chapter 1.

- When we use the term "normalize," we refer to normalizing a vector. I.e., projecting it unto the unit-hypersphere.

Part I

Background and Theory

Chapter 1

Background

In this chapter, we summarize some of the most important background work for the work in this thesis.

1.1 Introduction

In prediction modeling, we have a set of predictor and target variables. We hope the predictor variables can be used to predict the target variables. In the classical statistical learning framework, we were historically working with datasets considered very small by modern machine learning standards. This might explain why the classical statistical learning framework developed as it did. Here the systematic relationship between the predictor and target variables was assumed denotable with a simple model. Simple in the sense of a limited amount of parameters by modern standards. I.e., less than 250 real numbers, for example.

The systematic relationship was viewed as a largely tractable and human-interpretable relationship; the rest would, in reality, be random noise. Therefore we wished to be careful about having a model with too many parameters and too high a capacity. Although many parameters may yield low training error, it would yield a higher error when we test our model on new data (Hastie et al., 2001). Because the performance gain we saw on the training data was not caused by superior modeling of the true systematic relationship. Instead, our model was modeling the noise, or even worse idiosyncratic noise, of our dataset. This effect is referred to as "overfitting." Hence, the model should have a large enough number of parameters to model the systematic relationship adequately, but without having so many, it captured the noise of our dataset. Another way of limiting overfitting is regularization. That is often used to avoid too high model variance or overfitting. Regularization is part of a model's architecture that aims to reduce variance by incentivizing a smaller parameter norm. The norm referred to here is not necessarily $\mathcal{L}^2$, but often is.

In statistical learning, it is common to employ more parameters than optimal on their own and then add regularization. This approach often results in lower test errors than the alternative. The interpretation is that we need to add extra parameters since we do not know which are capable of modeling the true relationship. Regularization is added to avoid modeling too complicated relationships apparent in the data. Since we fear this will lead to overfitting because we have assumed from the beginning that the relationships would not be too complicated. Perhaps that is an implicit Occam's razor argument (Duignan, 2023). Therefore, we allow introducing this regularization bias to have a greater reduction in the variance of our model. In the classical sense, we wish to find the best trade-off between bias and variance by tuning the number of parameters and intensity of regularization. The best compromise between these two has classically been called the bias-variance trade-off best fit.

However, when computational recourses and datasets are massive, Belkin et al. (2019); Sejnowski (2020); Sevilla et al. (2022); Belkin et al. (2020); Nakkiran et al. (2019); Mei and Montanari (2021) argue that the test error may not strictly follow this classical framework. This behavior is referred to as "Double Descent" by most of those authors. We refer to this framework as Double Descent or modern machine learning. Most of the arguments are empirical. However, Mei and Montanari (2021) mathematically argues this is the case under certain assumptions. In Double Descent, we first observe the classical bias-variance-trade-off curve, and then, slowly, the test error decreases as the number of parameters increases. Even below the original bias-variance best fit.

Double Descent does not entail going arbitrarily far in the variance direction in the classical sense either. It is only the addition of parameters that lowers the test error. The results in Nakkiran et al. (2019) further empirically exemplify the Double Descent phenomenon. However, they also show, in particular, that regularization terms remain effective for arbitrarily large models. The significance of regularization is relevant for the work in this thesis. One of our goals is to employ a more sophisticated form of regularization for these high-parameter models than usual for the field. The models most used in this new Machine Learning framework of *More Compute* (Sevilla et al., 2022) are based on the artificial neural network (Cun and Fogelman-Soulié, 1987).

The classical statistical learning and deep learning frameworks are seemingly at odds. Two distinct research cultures are attempting the same thing in contradictory frameworks. The two cultures were explored already in Breiman (2001), where the author suggested to *"... move away from exclusive dependence on data models and* *adopt a more diverse set of tools."*. Part of the goal of this thesis is to attempt to follow the advice from there. However, moving away from exclusive dependence does not entail discarding altogether. We try to bring some of the expertise in regularization and the Bayesian framework into modern deep learning. The culture gap has been more recently discussed, taking into account the new Double Descent framework, in Nalisnick et al. (2023). There the authors attempt something related to the goal of this thesis, namely understanding Double Descent deep models, and their surprisingly impressive performance from the statistical view.

The drop in test error seems to be caused by the increase in parameters and capacity without omitting regularization. An attempt to bridge some of the gaps between modern deep learning and the classical statistical learning framework might be to incorporate more competent regularization into arbitrarily large models. This could reduce the test error in deep learning further. Perhaps one could outperform the current combination of Dropout, see 2.1.2.2, layer by layer (Srivastava et al., 2014) and $\mathcal{L}^2$ regularization penalty on the parameter vector in the calculation of the loss (Wu et al., 2014). Although $\mathcal{L}^2$ regularization has had good support in the classical statistical learning literature for a long time (Marquardt and Snee, 1975), referred to there as "Ridge Regression," we aim to find room for improvement in both the Dropout and $\mathcal{L}^2$ part simultaneously. Gal and Ghahramani (2015) indicate BNNs with Gaussian priors induce Dropout. Furthermore, Figueiredo (2003) indicates that the Gaussian prior also induces L2 regularization. Therefore, we find the BNN architecture interesting and aim to outperform the current Gaussian prior BNNs in this thesis.

However, these deep models tend to fail spectacularly on rare occasions. Models that exert less of such behavior are in the literature referred to as robust models (Huber, 2011, 2002). These are generally desirable, with a particular use in e.g., medical applications. In addition, in applications where robustness is especially desired, there is often a desire for an uncertainty estimate of the predictions the models make. Statistical learning where the classical bias-variance curve is optimized is still done today. Regardless of Double Descent, these models are much more interpretable. Hence, making an informed decision on how to treat their predictions and their uncertainty is much easier, in addition to their usual tendency to be more robust.

An attempt to try to incorporate the best of both worlds, i.e., having the performance of high-parameter deep neural networks, with the uncertainty estimates and robustness of classical models, is the Bayesian Neural Network[1]. Bayesian Neural Networks (BNNs) have been shown to both match conventional neural networks and

[1] See (Shridhar et al., 2019) for reading on the convolutional case, also helpful for fully connected BNNs.

outperform them in terms of robustness and uncertainty estimation, although at a greater cost of computation (Goan and Fookes, 2020). They are usually only selected when datasets are small enough that computational recourses are sufficient regardless, or where uncertainty estimates are desired. Although the interpretability of classical models is still far out of reach for these models, the Bayesian Neural Network is aimed to compete with the conventional, or as we will address them henceforth, frequentist neural networks (FNNs). Both BNNs and FNNs suffer from the problem of uninterpretablility.

BNNs usually employ an isotropic Gaussian prior over their parameters (Dusenberry et al., 2020; Zhang et al., 2019; Immer et al., 2020; Louizos and Welling, 2017; Hernandez-Lobato and Adams, 2015; Neal, 1996). However, in this thesis, we will attempt to find alternative priors that hopefully replace the need for Batch Normalization. This is an attempt to offset some of the increase in computational expense and, at the same time, hopefully, be more computationally stable and informative.

A gradient descent algorithm is used in neural networks, forcing us to compute the gradient. This computation is a primary source of problems, where the perhaps most severe is vanishing and exploding gradient. We wish to specify a prior such that the norm of the backward pass is one, alleviating the need for Batch Normalization. As explained in Subection 2.1.3, the vanishing and exploding gradient problem becomes increasingly severe the deeper, i.e., the more parameters the network has. Therefore specifying the prior such that the weights are $\mathcal{L}^2$ norm 1, the backward pass is greatly restricted, and the gradient size problem is alleviated. When discussing the norm later, we always mean $\mathcal{L}^2$-norm unless we specify otherwise.

Furthermore, we expect to see greater and greater benefit the larger the model is. Since massive models have better prediction performance, being restricted to non-massive models is not ideal. In the Double Descent framework, massive models seem to be the future of non-interpretable prediction. Restriction to the unit hypersphere could be a helpful form of regularization regardless of the properties of the gradient. Since a deep network with predictor and target variables on a standardized scale, i.e., normalized, does, in theory, not need the weights of any layers to have a norm different from one.

We see no obvious reason to believe that restriction to the unitsphere should meaningfully reduce model capacity. That suggests the restricted model has the same capacity as Gaussian prior BNNs but with fewer optimization solutions to the same problem. However, it is hard to evaluate whether that constitutes an advantage, given the earlier mentioned Double Descent trend. First-order optimization methods, Adam in particular (Kingma and Ba, 2014), are utilized for deep learning due to computational concerns. This optimization method class can perform worse when parameters are not on the same scale (Zhang et al., 2017; Bottou et al., 2016). That problem is compounded with the numerical problems of gradient computation mentioned earlier. If certain parameters are of inappropriate size, they are not only themselves less amenable to optimization. But they affect the other parameters earlier in the networks, whose gradient are approximated later, and dependently on the later parameters. That provides compounding suboptimal optimization steps for parameters of differing size, both from theory and computation. In addition to that, we hope that this prior specification could bring the Bayesian advantages of robustness and uncertainty estimation into the modern world of massive models.

Double Descent indicates massive models are useful. Massive models are usually built on FNN-based architectures. Gradient problems are currently addressed with heuristics and worsen as models grow. The optimization method used to optimize them is less adept at handling parameters at differing scales. These two problems magnify each other. If we specify a norm one prior we redress both issues at once, less heuristically. Hopefully, making them marginally more interpretable. Using BNNs in themselves, with the conventional Gaussian prior already has benefits in uncertainty estimation and robustness. We aim to inherit these useful traits in the novel-prior BNN we propose here. In this thesis, we propose and apply the Hyperspherical Uniform prior BNN and evaluate its performance.

Chapter 2

Theory

In this chapter, we summarize some of the most important theory for the work in this thesis.

2.1 Frequentist Neural Networks

The benchmark for the aforementioned high-parameter models has, for a while, been the neural network. Although it has gone through many improvements over the years, currently more utilized as a module in modern larger architectures, it is at its core the same as it was when proposed in '87 (Cun and Fogelman-Soulié, 1987). However, we will have to refer to these as frequentist neural networks (FNNs) for the remainder of this thesis since we will explore novel solutions in Bayesian neural networks (BNNs).

We will begin by introducing a barebones FNN. It consists of alternating linear transformations and non-linear activation functions. The linear transformations are of the form $\mathbf{x}^\top \mathbf{W} + \mathbf{b}$ where x is the output from the previous activation function. The activation function is usually a ReLU function. This applies to each of the elements x_i in the output vector $\mathbf{x}$ from the previous linear layer:

$$g(x_i) = \max(0, x_i) = \begin{cases} 0 & x_i < 0 \\ x_i & x_i \geq 0 \end{cases}$$

Although in our implementation, we use a variant called LeakyReLU, where $-c * x_i$ is returned instead of 0. c is usually a small number, around 0.05:

$$g(x_i) = \begin{cases} -c * x_i & x_i < 0 \\ x_i & x_i \geq 0 \end{cases}$$

The number of such layers is called the depth of the network. The output dimension of each $\mathbf{W}_j$ (the $\mathbf{b}$'s are applied to the outputs so its size always coincides with the output size of $\mathbf{W}$) are referred to as the width of each layer n. Often this structure is visualized in a manner where the output elements from the activation layer are nodes and the linear transformation that follows as a set of connections between every previous node and every next node, see Figure 2.1.

Sending the input data through the entire network, i.e., making a prediction, is called a forward pass. The trainable parameters here are the weights **W** and biases **b** of the linear part of each layer. The ReLU is not trained. By trainable, we mean these are the parameters updated when we fit our FNN to our training data. This procedure is done by first specifying how we wish to measure our model's "goodness of fit" to the data. Often Residual Sums of Squares, RSS, is chosen for regression. This is defined as $RSS = \sum_{i=1}^{n}(y_i - f(x_i))^2$ where f and the y_i's is the neural networks prediction for x_i and its corresponding true values respectively.

For classification, we apply the SoftMax function to the last five outputs and then get a scalar by applying, for example, binary cross-entropy on those with regards to the true class value: $-\frac{1}{N}\sum_{i=1}^{N}\mathbf{y}_i \cdot \log(f(\mathbf{x}_i)) + (1-\mathbf{y}_i) \cdot \log(1-p(\mathbf{y}_i))$. Where f is the networks, $\mathbf{y}$ is the true class, and $p(\mathbf{y}_i)$ is the probability for that class. This is called the loss. In the case of classification, the SoftMax function will be applied to each of the five nodes in the graphic to the far right of Figure 2.1. Each node represents a class, which after the SoftMax becomes the prediction probabilities. In order to yield the predictions of the network, all the steps must, of course, be applied to the input vector $\mathbf{x}$. This application of the whole nested function is called the forward pass. In order to begin training the network, a random sample of observations from our data set is first drawn. The network then conducts a forward pass on this data sample's input, or predictor, vectors. The loss between the predictions yielded from this forward pass and the targets from the data sample is then calculated.

The gradient of this loss is calculated by invoking the chain rule sequentially backward through every layer. This process is referred to as backpropagation Rumelhart et al. (1986) due to the conventional use of a forward-pass computation graph, from subsection 2.1.3, to keep an overview assisting in the step-by-step application of the chain rule. The computation graph was invented for the overview during the manual calculation. However, conducting backpropagation is intractable analytically for humans in modern applications. PyTorch (Paszke et al., 2019) can do this for us. In PyTorch, the computation graph refers to its internal tracking of the sequence

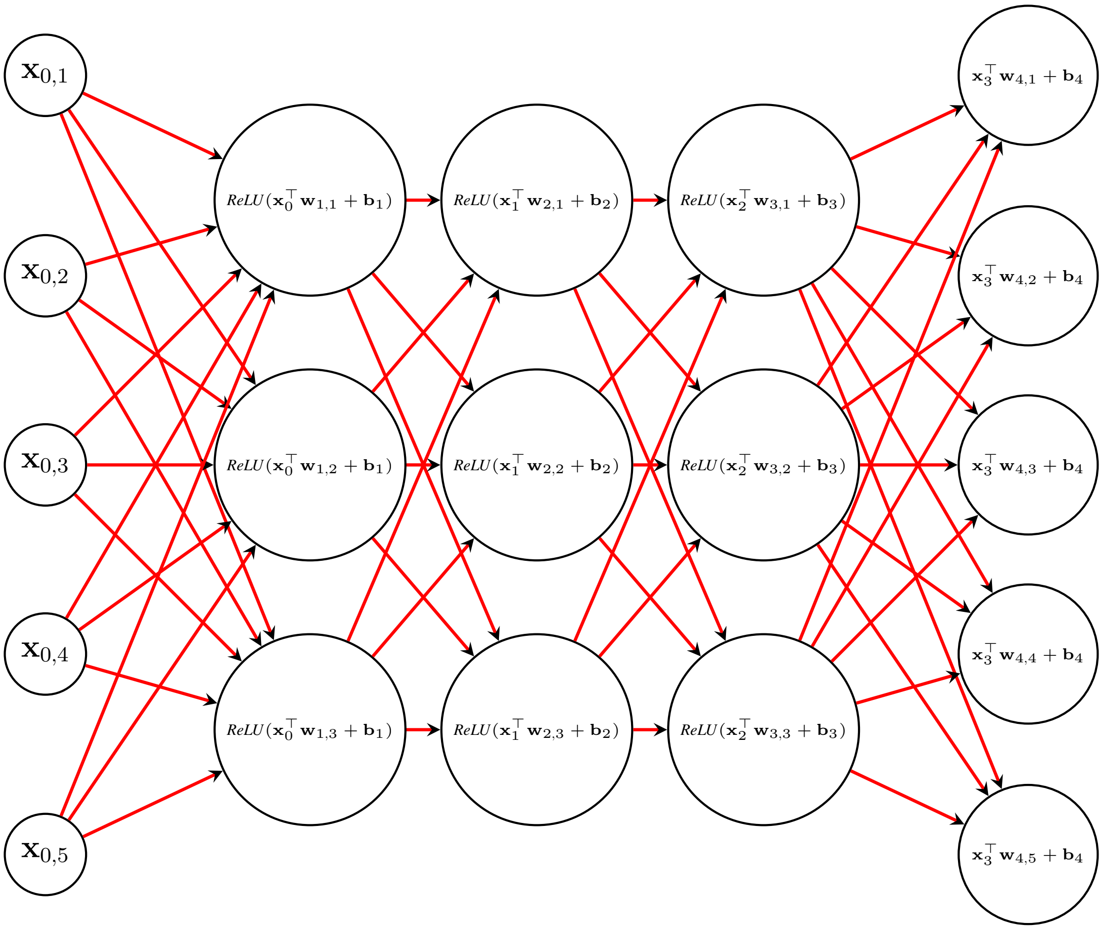

$\mathbf{x}_{0,1}$
$\mathbf{x}_{0,2}$
$\mathbf{x}_{0,3}$
$\mathbf{x}_{0,4}$
$\mathbf{x}_{0,5}$
$ReLU(\mathbf{x}_0^\top \mathbf{w}_{1,1} + \mathbf{b}_1)$
$ReLU(\mathbf{x}_0^\top \mathbf{w}_{1,2} + \mathbf{b}_1)$
$ReLU(\mathbf{x}_0^\top \mathbf{w}_{1,3} + \mathbf{b}_1)$
$ReLU(\mathbf{x}_1^\top \mathbf{w}_{2,1} + \mathbf{b}_2)$
$ReLU(\mathbf{x}_1^\top \mathbf{w}_{2,2} + \mathbf{b}_2)$
$ReLU(\mathbf{x}_1^\top \mathbf{w}_{2,3} + \mathbf{b}_2)$
$ReLU(\mathbf{x}_2^\top \mathbf{w}_{3,1} + \mathbf{b}_3)$
$ReLU(\mathbf{x}_2^\top \mathbf{w}_{3,2} + \mathbf{b}_3)$
$ReLU(\mathbf{x}_2^\top \mathbf{w}_{3,3} + \mathbf{b}_3)$
$\mathbf{x}_3^\top \mathbf{w}_{4,1} + \mathbf{b}_4$
$\mathbf{x}_3^\top \mathbf{w}_{4,2} + \mathbf{b}_4$
$\mathbf{x}_3^\top \mathbf{w}_{4,3} + \mathbf{b}_4$
$\mathbf{x}_3^\top \mathbf{w}_{4,4} + \mathbf{b}_4$
$\mathbf{x}_3^\top \mathbf{w}_{4,5} + \mathbf{b}_4$

Figure 2.1: A Frequentist Neural Network. Each of the red lines represents each of the dimensions of $\mathbf{x}_j$. In the literature, the lines often represent each of the entries in the weights $\mathbf{w}$. Note, for each i, j: $\mathbf{x}_{i,j} = \mathit{ReLU}(\mathbf{x}_{i-1}^\top \mathbf{w}_{i,j} + \mathbf{b_i})$.

Although there is usually no activation in the last layer.

of every computation. This tracking is essential for PyTorch to manage to do the backpropagation for us automatically.

2.1.1 SGD and Adam

A small change in the parameters along the direction of this calculated gradient is made. This process of sampling from the data and conducting a parameter gradient step from that subsample stochastic gradient descent, SGD (Robbins and Monro, 1951). A fixed step size was originally set, but an adaptive learning rate is used more nowadays. Adaptive gradient descent methods track the direction of earlier gradients. A larger step is made if the gradients of the last few steps are roughly in the same direction. Contrarily, smaller steps are made if the last few gradients are more different in their direction. This memory feature of adaptive methods is called momentum. The most common is called "Adam" (Kingma and Ba, 2014), which uses a variant of it. These small steps are conducted several times. The exact number of times is usually referred to as epochs and is a tuneable hyperparameter of the network. SGD only utilizes the first derivative and is hence a first-order method. First-order methods do not handle parameters on different scales well. Adam's momentum variant may be viewed as an approximation to the second derivative. Yet, no evidence suggests this approximation ameliorates the scale sensitivity. Thus, Adam retains the propensity for scale sensitivity. Scale sensitivity is relevant for the prior specification later (Zhang et al., 2017; Bottou et al., 2016).

To describe them formally, let us begin with their predecessor, static learning rate gradient descent. Let f and ψ denote the network and its parameters. Let α denote the static learning rate, and t denote the optimization step. Let L denote the loss function of the whole network with respect to the parameters. Then one optimization step would be expressed by:

$$\psi_{t+1} = \psi_t - \alpha \nabla_\psi L(\psi_t)$$

Originally Robbins and Monro (1951) suggested the Stochastic variant of this procedure due to reduced demand for compute. In SGD, ∇_ψ is replaced an unbaised estimator $\widetilde{\nabla}_\psi L(\psi_t)$. $\widetilde{\nabla}_\psi L(\psi_t)$ is, as mentioned, just the gradient calculated on a subsample of the available data. Adam, as mentioned, will introduce another layer of complexity. There, the size of α per optimization step t will depend what $\widetilde{\nabla}_\psi L(\psi_t)$ was calulated to be from the earlier t. Formally:

$$
\begin{aligned}
m_{\psi_{t+1}} &\leftarrow C_1 m_{\psi_t} + (1 - C_1)\,\widetilde{\nabla}_\psi L(\psi_t) \\
v_{\psi_{t+1}} &\leftarrow C_2 v_{\psi_t} + (1 - C_2)\left(\widetilde{\nabla}_\psi L(\psi_t)\right)^2 \\
\hat{m} &= \frac{m_{\psi_{t+1}}}{1 - C_1} \\
\hat{v} &= \frac{v_{\psi_{t+1}}}{1 - C_2} \\
\psi_{t+1} &\leftarrow \psi_t - \alpha \frac{\hat{m}}{\sqrt{\hat{v}} + \varepsilon}
\end{aligned}
$$

where ε is a small scalar (e.g. 10^{-8}) used to prevent division by 0, and C_1 (e.g. 0.9) and C_2 (e.g. 0.999) are the forgetting factors for gradients and second moments of gradients, respectively. Squaring and square-rooting is done element-wise.

2.1.2 Computational Heuristics for Performance

2.1.2.1 Batch Normalization

Two additional techniques are usually employed to supplement test-error performance further. Batch Normalization Ioffe and Szegedy (2015) is applied between every layer or every so often. It consists of normalizing the entire forward pass tensor between two layers. This helps prevent the forward-passing tensor from growing too large. The norm of the forward pass will inherently influence the gradient norm. Batch Normalization, therefore, influences the forward and backward passes to remain closer to a computationally desirable norm. Bayesian Neural Networks, introduced in Section 2.3, also have to, in principle, utilize this method. Although in this thesis, we will resolve that.

2.1.2.2 Dropout

The second technique is called Dropout (Srivastava et al., 2014; Hinton et al., 2012). This technique consists of randomly setting the output weights of a node to 0. Dropout is usually applied only during training. The original motivation was to make each node learn something general rather than specific to the dataset. Dropout generally improves performance when the probability of dropping each node is tuned appropriately during training. See Sanderson (2015) for an excellent visual introduction to frequentist neural networks.

2.1.3 Computation Graphs, Backpropagation and Autograd

Tracking the computation graph and its chain rule differentiation is key to understanding Neural Networks. It will also be key to understand the developments in this thesis, as our selection of Prior and Variational Distribution to achieve a norm one backward pass without losing the other benefits of Gaussian BNNs are the main goals of this thesis.

Solving any arbitrarily complex differentiation analytically is not tractable for computers. Numerical approximation methods do not resolve that, as they would yield compounding inaccuracies. Therefore, another method is needed. Computation graphs get their name from, intuitively, being the graph of the computation. This graph is helpful for humans when we solve the backpropagation manually by giving us an overview. The relevancy remains for computers as well. A third option is automatic differentiation, autograd (Paszke et al., 2017) for short. Here we program our forward pass into a series of primitive operations that we know how to differentiate from the chain rule. Thus we get exact derivatives for the compositions of our defined primitive operations and avoid repetitive calculations of the same values resulting in compounding inaccuracies that we would get in numerical approximation. In practice, the computer tracks all the computations, i.e., implicitly drawing a graph of them, and then chain-rule differentiates all the computations in reverse order.

Imagine a function f with input $\mathbf{x}$, component functions f_1, f_2, f_3, f_4, input $\mathbf{x}$ and $\mathbf{y}$ as the output. It is unnecessary to view f, its components and output as a neural network, its layers, and its loss just yet. Nevertheless, the reader may already do so, as this is the motivation. In that setting, Autograd is used only in training. Here the output(s), or predictions, are wrought into a loss scalar. In that case, $\mathbf{y}$ is a scalar. In Explicitly writing out the component functions:

$$\mathbf{y} = f(\mathbf{x}) = f_4(f_3(f_2(f_1(\mathbf{x}))))$$

Let $f_1 : \mathbb{R}^n \to \mathbb{R}^3$, $f_2 : \mathbb{R}^3 \to \mathbb{R}^3$, $f_3 : \mathbb{R}^3 \to \mathbb{R}^3$, $f_4 : \mathbb{R}^3 \to \mathbb{R}^m$. Then the dimensionality of the intermediate variables are $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{x}_1 \in \mathbb{R}^3$, $\mathbf{x}_2 \in \mathbb{R}^3$, $\mathbf{x}_3 \in \mathbb{R}^3$, with $\mathbf{y} \in \mathbb{R}^m$. These four functions are the primitive operations mentioned earlier. The computer tracks the computation graph by storing these four components instead of f as is. Now that we have each component function of f, we can utilize the chain rule:

$$\frac{\partial f(\mathbf{x})}{\partial x} = \frac{\partial f(\mathbf{x})}{\partial \mathbf{x_4}}\frac{\partial \mathbf{x_4}}{\partial \mathbf{x_3}}\frac{\partial \mathbf{x_3}}{\partial \mathbf{x_2}}\frac{\partial \mathbf{x_2}}{\partial \mathbf{x_1}} = \frac{\partial f_4(\mathbf{x_3})}{\partial \mathbf{x_4}}\frac{\partial f_3(\mathbf{x_2})}{\partial \mathbf{x_3}}\frac{\partial f_2(\mathbf{x_1})}{\partial \mathbf{x_2}}\frac{\partial f_1(\mathbf{x})}{\partial \mathbf{x_1}} = J_f(\mathbf{x})$$

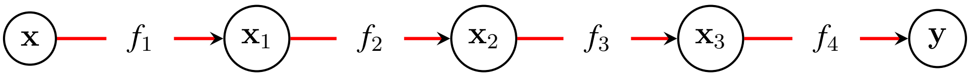
Remember from Terminology 0.1, $\frac{\partial f_i}{\partial \mathbf{x_j}}$ is the ith row and jth column of the Jacobian, $J_f(\mathbf{x})$, of f. Backward differentiation (Curtiss and Hirschfelder, 1952) is more computationally appropriate for finding our Jacobian since we are building this theory towards neural networks, meaning the computational cost of forward differentiation is $O(n * (...))$ while for backward differentiation, it is $O(m * (....))$, which is smaller for neural networks since the loss function is at the end wrought into one scalar. Hence, we will build the Jacobian up row-wise using the vector-Jacobian product, VJP for short. Let the unit vector $\mathbf{e_j} \in \mathbb{R}$, which we view as a vector for generality. The point of this generality is to illustrate why backward differentiation is more appropriate computationally than forward differentiation. Let us calculate the VJP with backward differentiation with this generality:

$$\nabla f(\mathbf{x})_1 = \mathbf{e_1}^\top J_f(\mathbf{x}),\ \nabla f(\mathbf{x})_2 = \mathbf{e_1}^\top J_f(\mathbf{x}), \ldots,\ \nabla f(\mathbf{x})_m = \mathbf{e_1}^\top J_f(\mathbf{x})$$

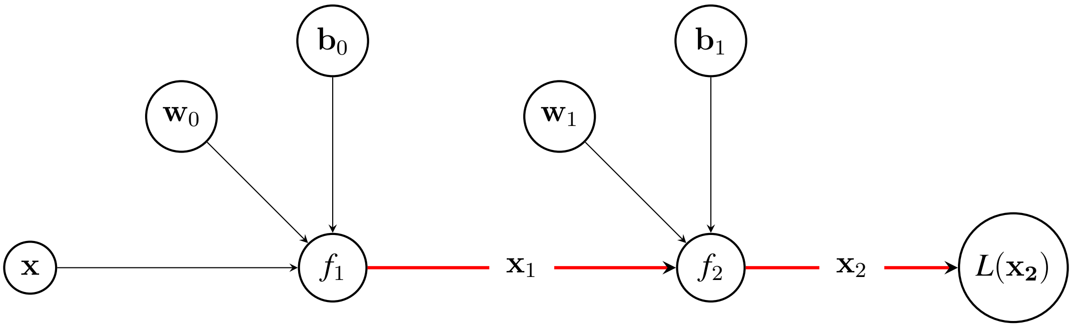
Now see what happens if we instead were to construct the Jacobian using forward differentiation:

$$\tfrac{\partial f(\mathbf{x})}{\partial x_1} = J_f(\mathbf{x})\mathbf{e_1},\ \tfrac{\partial f(\mathbf{x})}{\partial x_2} = J_f(\mathbf{x})\mathbf{e_1}, \ldots,\ \tfrac{\partial f(\mathbf{x})}{\partial x_n} = J_f(\mathbf{x})\mathbf{e_1}$$

In short, we calculate n products in forward differentiation and m in backward differentiation. Since m will, for our purposes, denote the dimension of our loss function, namely one, while n denotes the dimension, or number, of parameters. Hence, we use backward differentiation in optimizing neural networks. The analogy to optimization to neural networks may still seem strange because here. It may seem as though we are trying to optimize our predictors with respect to the loss with respect to our targets, which is not quite it. However, we need to start with the rudimentary before we can advance. The computation graph for our function f in this example can be graphed followingly:



Now we will advance to a more intricate setting reminiscent of actual neural networks. The following graph is of an entire feed-forward network and the calculation of its loss, $L(\mathbf{x_2})$. Let us now explicitly define $f_1(\mathbf{x}; \mathbf{w}, \mathbf{b}) = f_2(\mathbf{x}; \mathbf{w}, \mathbf{b}) = \mathbf{x}^\top \mathbf{w} + \mathbf{b}$. Then we also define $\mathbf{x}_1 = f_1(\mathbf{x}; \mathbf{w}_0, \mathbf{b}_0) = \mathbf{x}^\top \mathbf{w}_0 + \mathbf{b}_0$ and $\mathbf{x}_2 = f_2(\mathbf{x}; \mathbf{w}_1, \mathbf{b}_1) = \mathbf{x}^\top \mathbf{w}_1 + \mathbf{b}_1$.



Let us calculate the derivatives with respect to our parameters $\mathbf{w}_1$, $\mathbf{b}_1$, $\mathbf{w}_0$, $\mathbf{b}_0$:

$$\frac{\partial L(\mathbf{x_2})}{\partial \mathbf{w}_1} = \frac{\partial L(\mathbf{x}_2)}{f_2(\mathbf{x};\mathbf{w_1},\mathbf{b_1})} \frac{\partial f_2(\mathbf{x};\mathbf{w_1},\mathbf{b_1})}{\partial \mathbf{w}_1} = \frac{\partial L(\mathbf{x_2})}{\mathbf{x}_2} \frac{\partial \mathbf{x}_1^\top \mathbf{w}_1 + \mathbf{b}_1}{\partial \mathbf{w}_1} = \frac{\partial L(\mathbf{x_2})}{\mathbf{x}_2} \mathbf{x}_1$$

$$\frac{\partial L(\mathbf{x_2})}{\partial \mathbf{w}_0} = \frac{\partial L(\mathbf{x}_2)}{f_2(\mathbf{x};\mathbf{w_1},\mathbf{b_1})} \frac{\partial f_2(\mathbf{x};\mathbf{w_1},\mathbf{b_1})}{\partial \mathbf{w}_0} = \frac{\partial L(\mathbf{x_2})}{\mathbf{x}_2} \frac{\partial \mathbf{x}_1^\top \mathbf{w}_1 + \mathbf{b}_1}{\partial \mathbf{x}_1} \frac{\partial \mathbf{x}_0^\top \mathbf{w}_0 + \mathbf{b}_0}{\partial \mathbf{w}_0} = \frac{\partial L(\mathbf{x_2})}{\mathbf{x}_2} \mathbf{w}_1 \mathbf{x}$$

$$\frac{\partial L(\mathbf{x_2})}{\partial \mathbf{b}_1} = \frac{\partial L(\mathbf{x}_2)}{f_2(\mathbf{x};\mathbf{w_1},\mathbf{b_1})} \frac{\partial f_2(\mathbf{x};\mathbf{w_1},\mathbf{b_1})}{\partial \mathbf{b}_1} = \frac{\partial L(\mathbf{x_2})}{\mathbf{x}_2} \frac{\partial \mathbf{x}_1^\top \mathbf{w}_1 + \mathbf{b}_1}{\partial \mathbf{b}_1} = \frac{\partial L(\mathbf{x_2})}{\mathbf{x}_2}$$

$$\frac{\partial L(\mathbf{x_2})}{\partial \mathbf{b}_0} = \frac{\partial L(\mathbf{x}_2)}{f_2(\mathbf{x};\mathbf{w_1},\mathbf{b_1})} \frac{\partial f_2(\mathbf{x};\mathbf{w_1},\mathbf{b_1})}{\partial \mathbf{b}_0} = \frac{\partial L(\mathbf{x_2})}{\mathbf{x}_2} \frac{\partial \mathbf{x}_1^\top \mathbf{w}_1 + \mathbf{b}_1}{\partial \mathbf{x}_1} \frac{\partial \mathbf{x}_1}{\partial \mathbf{b}_0} = \frac{\partial L(\mathbf{x_2})}{\mathbf{x}_2} \mathbf{w}_1 \frac{\partial \mathbf{x}_0^\top \mathbf{w}_0 + \mathbf{b}_0}{\mathbf{b}_0} = \frac{\partial L(\mathbf{x_2})}{\mathbf{x}_2} \mathbf{w}_1$$

We omit calculating $\frac{\partial L(\mathbf{x_2})}{\mathbf{x}_2}$ as this varies by loss function. The most important clue here is the intermediate term $\mathbf{w}_1$ that always shows up when we backward pass. In other words, the size of the $\mathbf{w}$'s in deep neural networks will be key in determining how large or small the gradient will be. This illustrates why one of the main goals of the thesis is to restrict the size of the weights to norm 1. Since this will keep the entire gradient with respect to all parameters within bounds, avoiding exploding and vanishing gradients. The point of computation graphs was originally to help humans that did not have access to PyTorch, including students who need to learn what is going on under the hood. The forward graph is mostly there to keep track of how to conduct the backward-chain rule operations.

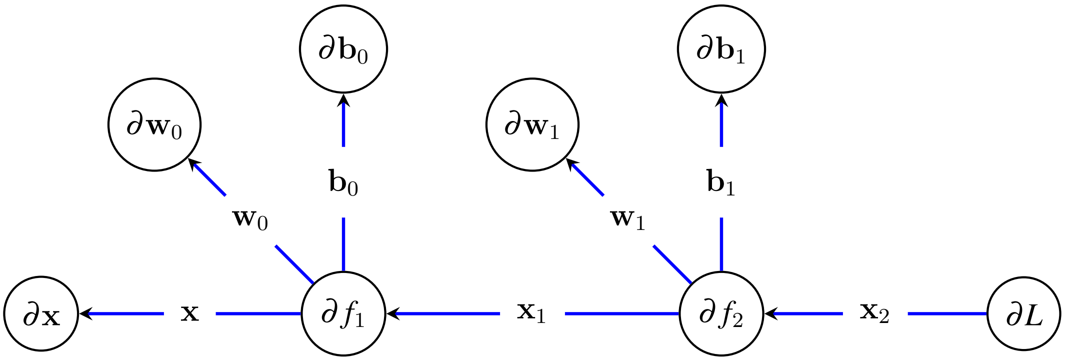


Figure 2.3: Conventional Notation Computation Graph

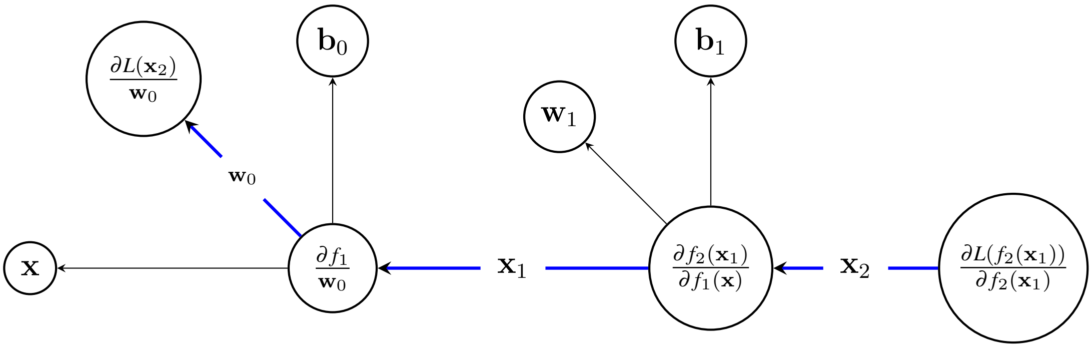

$\frac{\partial L(\mathbf{x}_2)}{\mathbf{w}_0}$
$\mathbf{b}_0$
$\mathbf{w}_0$
$\mathbf{x}$
$\frac{\partial f_1}{\mathbf{w}_0}$
$\mathbf{x}_1$
$\mathbf{b}_1$
$\mathbf{w}_1$
$\frac{\partial f_2(\mathbf{x}_1)}{\partial f_1(\mathbf{x})}$
$\mathbf{x}_2$
$\frac{\partial L(f_2(\mathbf{x}_1))}{\partial f_2(\mathbf{x}_1)}$

Figure 2.2: Mathematical Notation Computation Graph

The reader may react to the sudden change with $\frac{\partial f_1}{\mathbf{w}_0}$ in the node for $\mathbf{w}_0$. This is not the conventional way of notation in this field. The convention is, in fact, a grave abuse of notation. Conventional notation would instead fill the nodes in Figure 2.2 with ∂L, ∂f_2, ∂f_1, $\partial \mathbf{w}_0$, which does not make mathamatical sense. However, this makes the computation graph's manual calculation much easier. Because in the way we drew it above, we need to fill the nodes with new partial derivatives for every weight or bias derivative that we want to calculate. This convention also has a useful analogy to what PyTorch is doing under the hood. We will also draw a graph with this convention because we use abuses the notation in order to draw the whole backward graph at once. A warning for the mathematically rigorous reader, view Figure 2.3 at your own risk.

2.1.3.1 PyTorch

Every computational step described in the last section can be implemented where the weights biases and input vector are torch tensors. For the weights and biases or any other parameter we would like to optimize, PyTorch (Paszke et al., 2019) will automatically differentiate these tensors. Pytorch is a library within the Python programming language (Van Rossum and Drake, 2009; Hunt, 2019). This automatic differentiation has many details we cannot go into here, but it is, at its core, just an implementation of what was described in the last section. It will find the gradient of the loss with respect to all the parameters we wish to optimize, and this will be fed to Adam, and an optimization step will be conducted.

Autograd is a core property of PyTorch. It is perhaps its main advantage and why it is the leading library for deep learning (He, 2019; OpenAI, 2023). Manual implementation of the backward differentiation is orders of magnitude more time-consuming than the forward passing architecture. A property of PyTorch that must be mentioned is that there is *fundamentally* no way to instruct it of a domain restriction over its parameters. We write *fundamentally* because it seems this is something that really could not tractably be changed. One would more or less have to write the library over again. It will always optimize all its parameters registered for optimization across the entire real line. This PyTorch limitation is a problem for the goal of this thesis. We attempt Normalized Initialization inheritance in Subsection 5.1.2 to redress this. Terminologically, the backward pass is the inverse of the forward pass. Meaning it refers to calculating the gradient at each step, which intuitively goes "in reverse." Autograd is the system that PyTorch uses to conduct automatic differentiation in general. *Backpropagation* is a term that somewhat overlaps with automatic differentiation. It refers to the calculation of the expression of each factor, each of which is represented as a node in Figure 2.3, of the expression for the gradient.

2.2 Bayesian Inference

In Bayesian inference, we assume there is a true model describing the entire systematic relationship between our target and predictor random variables. Mathematically we also assume that this true model contains parameters. These true parameters are even in the theoretically ideal setting of specifying the true model, not necessarily reachable unless we have infinite data $\mathcal{D}$. However, in the theoretical case with infinite data, the posterior random variable $\mathbf{\Psi}|\mathcal{D}$ will be the true parameters describing the true model. There, the true parameters *are* mathematically not real numbers, but real-valued random variables.

2.2.1 Random Variables

This section assumes the reader is familiar with introductory measure theory[1]. To complete the formalism, the function $\mathbf{\Psi} : \Omega \rightarrow \mathbb{R}^m$ is a real-valued random variable if and only if:

$$\{\omega \in \Omega : \mathbf{\Psi}(\omega) \leq \mathbf{r}\} \in \mathcal{F} \quad \forall \mathbf{r} \in \mathbb{R}^m$$

Given a probability space $(\Omega, \mathcal{F}, \mathcal{P})$. Where Ω is the sample space, i.e., the set of all outcomes. $\mathcal{F}$ is the event space, each event $f \in \mathcal{F}$ is s.t. $f \subseteq \Omega$. Meaning $\mathcal{F}$ is a set of subsets of Ω. Lastly, $P : \Omega \rightarrow [0, 1]$ is a measure on Ω, with the property that $P(\Omega) = 1$. Measures with this last property are referred to as probability measures.

Another class of functions accompanies most of these random variables, probability density functions (pdfs). Let $(\mathcal{X}, \mathcal{A}, \mu)$ be the Lebesgue measure space. Assume also that $\mathbf{\Psi}$ has a pushforward measure that is absolutely continuous with regards to the Lebesgue measure. We assume that last property because strictly speaking, only such random variables have pdfs. A probability density function $p_{\mathbf{\Psi}}$ for $\mathbf{\Psi}$ is then any function that satisfies:

$$P[\{\omega \in \Omega : \mathbf{\Psi}(\omega) \in F\}] = \int_{\mathbf{\Psi}^{-1}(F)} dP(\omega) = \int_F p_{\mathbf{\Psi}}(\mathbf{x}) d\mu(\mathbf{x}) \quad \forall F \in \mathcal{A}$$

Remember, the conditional random variable $\mathbf{\Psi}|\mathcal{D}$ *is* the true parameters in the true model only when the model is the true model, and we have infinite data. We go through this to help clarify whether we are referring to the random variable itself, its pdf, or its realizations.

[1] A great resource here is chapters 7 and 8 in Lindstrøm (2017)

2.2.2 Bayesian Inference

The theoretical true posterior $\mathbf{\Psi}|\mathcal{D}$ is a conditional random variable. To find it, we first need the true prior $\mathbf{\Psi}$, likelihood $\mathcal{D}|\mathbf{\Psi}$, and marginal of the data $\mathcal{D}$. Moreover, know their respective pdfs. Knowing those expressions, we can invoke Bayes Theorem:

$$p_{\mathbf{\Psi}|\mathcal{D}}(\mathbf{\Psi}, \mathcal{D}) = \frac{p_{\mathbf{\Psi}}(\mathbf{\Psi}) p_{\mathcal{D}|\mathbf{\Psi}}(\mathcal{D}, \mathbf{\Psi})}{p_{\mathcal{D}}(\mathcal{D})} \tag{2.1}$$

In practice, the whole framework is not necessarily entirely representative of reality. On the other hand, Bayesian inference is often an excellent approximation to the actual situation. Moreover, even if it were, we do not know the pdf of either the true prior, the true likelihood $p_{\mathcal{D}|\mathbf{\Psi}}(\mathcal{D}, \mathbf{\Psi})$, or the true marginal distribution $p_{\mathcal{D}}(\mathcal{D})$ of the data. Selecting the likelihood is the only part an investigator should prioritize effort, as prior selection is also paramount. In practice, this selection is often made by plotting the training data and seeing which of the conventional toolbox methods works the best.

Although the choice of likelihood could be a significant source of error, the Bayesian statistician is often more concerned with the selection of prior. There is no generally accepted recipe for selecting a reasonable prior the same way as for the likelihood. Although one always has the option of uniform priors, this neglects some of the advantages of the Bayesian framework. A vital strength of the Bayesian framework is that it allows the investigator to let the prior represent his or her prior beliefs about the parameters $\mathbf{\Psi}$. Hence, the name "prior."

To connect intuition to mathematics, the prior represents the investigator's prior beliefs about the parameters, and the likelihood represents the parameters' role in the data. The parameters' role in the data will determine how to deal with observations from the data. When the data is introduced, the prior and likelihood for this data are multiplied, as seen in Bayes Theorem. Intuitively this can be viewed as a weighted integral of probabilities between prior belief and introduced evidence interpreted through the choice of likelihood. Hence, the posterior distribution over our parameters is obtained.

There are more reasons for conducting Bayesian inference instead of frequentist inference. The first is if the parameters are actually random variables. Bayesian inference should then yield superior results since it is closer to the mathematical reality. However, even in cases where the parameters are not real numbers, we can never know them exactly. Thus, we can utilize the mathematical framework of random variables to better capture our uncertainty of the parameters than in the frequentist setting. Therefore, our notion of random variable modeled uncertainty and true randomness may be similar enough to yield great results. Hence, the Bayesian framework might better account for the uncertainty of parameters even if the parameters actually are real numbers.[2]

2.2.3 Finding the Posterior

From (2.1), we see that we need three expressions to find our posterior. The epressions in question are the prior $p_{\mathbf{\Psi}}(\mathbf{\Psi})$, the likelihood $p_{\mathcal{D}|\mathbf{\Psi}}(\mathcal{D}, \mathbf{\Psi})$, and the marginal distribution of our data $p_{\mathcal{D}}(\mathcal{D})$. In the Bayesian framework, we allow ourselves to select a prior and the likelihood of our data, which we hope are reasonably appropriate. The marginal distribution $p_{\mathcal{D}}(\mathcal{D})$ is determined by those two earlier choices. Hence, if we want to find the posterior analytically, we would first need to find the marginal:

$$p_{\mathcal{D}}(\mathcal{D}) = \int p_{\mathcal{D}|\mathbf{\Psi}}(\mathcal{D}, \mathbf{\Psi}) p_{\mathbf{\Psi}}(\mathbf{\Psi}) d\mathbf{\Psi}$$

In Bayesian inference, where we have selected a likelihood and prior that is reasonably simple, this integral can be possible to calculate. However, if when dealing with complicated priors and likelihoods, we quickly find a problem. The integral is not tractable, perhaps impossible, to calculate analytically. This is the case for BNNs. Therefore we resort to algorithms for approximation of the true posterior instead.

2.2.3.1 Markov chain Monte Carlo

One of the most famous algorithms for posterior approximation is Markov chain Monte Carlo (MCMC), in modern practice through the Metropolis-Hastings Algorithm (Hastings, 1970). Here a Markov chain that has the desired distribution as its equilibrium distribution is constructed, and we can obtain a sample of the desired distribution by recording states from the chain. As with most optimization algorithms, we conduct several steps along this chain in order to have enough to sample from. In addition, this algorithm also requires a substantial "burn-in" period. "Burn-in" is necessary because it takes several steps before the Markov chain converges towards its equilibrium distribution, our posterior. Depending on the distributions in question, several steps on an order of magnitude, often similar to the number of steps used as the final sample, must be discarded for "burn-in."

We will not explain these methods in detail. The interested reader can see (Robert, 2015). As the reader might have already grasped, this method of computing the pos-

[2] See (Gelman et al., 2013) for more comprehensive reading on this topic.

terior will have a tremendous computational expense. MCMC makes few assumptions on the form of the posterior and hence tends to yield accurate predictions. It is generally more accurate than Variational Inference of the next section (Dupuy and Bach, 2016). However, it is computationally intractable for complicated architectures and large datasets. Therefore we will have to look at a better compromise. We allow ourselves to make certain assumptions about what form our posterior has, in order to let our computation be tractable.

2.2.3.2 Variational Inference

As mentioned earlier, Sevilla et al. (2022), together with the Double Descent authors, show that more compute is the general trend in deep learning. Thus, we wish to be careful with utilizing methods that entail great computational cost since any compute utilized there could be utilized elsewhere in our architecture. Therefore BNNs typically utilize a less precise but much cheaper method to approximate the posterior of its parameters. Namely, Variational Inference (Kingma et al., 2015; Blei et al., 2017), VI for short. The idea behind VI is first to posit a family of distributions and then to find the family member closest to the true posterior. Therefore, selecting what family to use for this purpose is key and entails the main weakness of VI compared to MCMC. Since MCMC makes no assumptions on the form of the posterior, VI will only ever be able to approximate the best fit to the true posterior within the family of distributions selected beforehand.

We refer to the family we select as the variational distribution, VD for short. The goodness of fit of our VD to the true posterior is measured with Kullback-Leibler divergence, KL divergence, or KL loss for short. However, this divergence is mathematically not a measure nor a metric. Because it is not symmetric. The definition of this KL divergence between the VD, $q_{\Psi}(\psi)$, and true posterior, $p_{\Psi|\mathcal{D}}(\psi, \mathcal{D})$, is defined as:

$$KL(q_{\Psi}(\psi) \,||\, p_{\Psi|\mathcal{D}}(\psi, \mathcal{D})) \overset{def}{=} \int q_{\Psi}(\psi) \log\left(\frac{q_{\Psi}(\psi)}{p_{\Psi|\mathcal{D}}(\psi, \mathcal{D})}\right) d\psi$$

The VD is parametrized by θ. We minimize the aforementioned KL divergence by finding the θ such that the KL divergence between the VD and true posterior is minimized. In order to conduct this optimization problem, it is helpful to rewrite the expression for this divergence. Note that the posterior, $p_{\Psi|\mathcal{D}}(\psi, \mathcal{D})$, is included in the denominator to the right here, of which we do not know the form. Therefore we invoke Bayes Theorem on the true posterior in the denominator to find a more helpful expression:

$$\int q_{\Psi}(\psi) \log\left(\frac{q_{\Psi}(\psi)}{p_{\Psi|\mathcal{D}}(\psi, \mathcal{D})}\right) d\psi$$

$$= \int q_{\boldsymbol{\Psi}}(\psi) \log \left(\frac{q_{\boldsymbol{\Psi}}(\psi)}{\frac{p_{\mathcal{D}|\boldsymbol{\Psi}}(\mathcal{D},\psi) p_{\boldsymbol{\Psi}}(\psi)}{p_{\mathcal{D}}(\mathcal{D})}} \right) d\psi$$

$$= \int q_{\boldsymbol{\Psi}}(\psi) \left(\log \left(\frac{q_{\boldsymbol{\Psi}}(\psi)}{p_{\boldsymbol{\Psi}}(\psi)} \right) - \log \left(p_{\boldsymbol{\Psi}|\mathcal{D}}(\psi, \mathcal{D}) \right) + \log(p_{\mathcal{D}}(\mathcal{D}) \right) d\psi$$

$$= \int q_{\boldsymbol{\Psi}}(\psi) \left(\log \left(\frac{q_{\boldsymbol{\Psi}}(\psi)}{p_{\boldsymbol{\Psi}}(\psi)} \right) - \log \left(p_{\mathcal{D}|\boldsymbol{\Psi}}(\psi, \mathcal{D}) \right) \right) d\psi + \int q_{\boldsymbol{\Psi}}(\psi) \log(p_{\mathcal{D}}(\mathcal{D}))\, d\psi$$

$$= \int q_{\boldsymbol{\Psi}}(\psi) \left(\log \left(\frac{q_{\boldsymbol{\Psi}}(\psi)}{p_{\boldsymbol{\Psi}}(\psi)} \right) - \log \left(p_{\mathcal{D}|\boldsymbol{\Psi}}(\psi, \mathcal{D}) \right) \right) d\psi + \log(p_{\mathcal{D}}(\mathcal{D})) \int q_{\boldsymbol{\Psi}}(\psi)\, d\psi$$

The last integral will integrate to 1 since it integrates a pdf over its entire domain.

$$= \int q_{\boldsymbol{\Psi}}(\psi) \left(\log \left(\frac{q_{\boldsymbol{\Psi}}(\psi)}{p_{\boldsymbol{\Psi}}(\psi)} \right) - \log \left(p_{\mathcal{D}|\boldsymbol{\Psi}}(\psi, \mathcal{D}) \right) \right) d\psi + \log(p_{\mathcal{D}}(\mathcal{D}))$$

Note that the logmarginal density of $\mathcal{D}$, $\log p_{\mathcal{D}}(\mathcal{D})$, will not change as we update the θ parameters of the pdf of $\boldsymbol{\Psi}$. Thus, we can discard the last addend, yielding us:

$$= \int q_{\boldsymbol{\Psi}}(\psi) \log \left(\frac{q_{\boldsymbol{\Psi}}(\psi)}{p_{\boldsymbol{\Psi}}(\psi)} \right) d\psi - \int q_{\boldsymbol{\Psi}}(\psi) \log \left(p_{\mathcal{D}|\boldsymbol{\Psi}}(\psi, \mathcal{D}) \right) d\psi \qquad (2.2)$$

Now we have the KL divergence between VD and true posterior on a form much more conducive to optimization since these integrals are tractable to approximate. consequently,our optimization task becomes to find the θ such that this term is minimized. The expression above is often referred to as the evidence lower bound, ELBO for short. Since the last $\log(p_{\mathcal{D}}(\mathcal{D}))$ term was discarded, this term cannot be smaller than the complete expression that includes the discarded term.

2.3 Bayesian Neural Networks

In the previous sections, we went through Neural Networks, Bayesian Inference, Variational Inference, and backpropagation in PyTorch. A Bayesian Neural Network (BNN) is a neural network that has random variable weights and biases. The BNNs we introduce are trained through variational inference. Thus, we posit a variational distribution over our random variable weights and biases $\mathbf{\Psi}$[3]. Consequently, the loss is specified as the KL divergence between the true posterior and the variational distribution. Then this loss is fed to Adam, and an optimization step is conducted. The computational structure of the forward pass is critical. Whether PyTorch can conduct automatic differentiation is determined by whether or not it can draw a differentiable computation graph. The keen reader may already suspect that random variables are not inherently amenable to automatic differentiation. There is a solution for this, which we will explain in this section.

2.3.1 Forward-pass and Reparametrization Trick

For PyTorch to draw a well-defined computation graph, we must separate the systematic and stochastic components of $\mathbf{\Psi}$. This separation is called the Reparametrization Trick (Kingma et al., 2015). Here we construct our variational distribution with a sampler function that separates the stochasticity from the systematic components. For example, the Gaussian could be mathematically constructed as $\mu + \sigma * \varepsilon$. Here μ and σ are the real number parameters stored in PyTorch, while ε is a standard normal Gaussian that we sample from. It is μ and σ that PyTorch calculates the gradient of the loss with respect to. Here, the minimization of loss with a step in σ is conducted, avoiding the backward pass flowing "into" ε, therefore resolving PyTorchs inability to handle random variables as parameters directly. From Subsection 2.1.3, we know drawing the computation can be a good idea to help to understand. Figure 2.4 represents how the reparametrization of $\mathbf{\Psi}$ is conducted in a forward pass structure. We draw for layer n, only considering the weights since the case for biases is analogous.

Note in the backwardpass graph, Figure 2.5, that the reparametrization of $\mathbf{W} = \mu_{\mathbf{W}_n} + \sigma_{\mathbf{W}_n}$ allows its differentiation with respect to $\sigma_{\mathbf{W}_n} * \varepsilon$ and $\mu_{\mathbf{W}_n}$, which become 1 and ε respectively. Allowing PyTorch to optimize both parameters with autograd as usual. PyTorch can then optimize all parameters describing the variational distribution $\mathbf{\Psi}$ with autograd. For BNNs with a Gaussian variational distribution, these parameters consist of the mean μ and variance σ.

[3] It may be somewhat confusing whether we are using $\mathbf{\Psi}$ to refer the "true" parameters themselves, or a variational distribution. In this section, $\mathbf{\Psi}$ refers to the variational distribution set to approximate the weights and biases.

The loss function, L, is specified as the KL divergence between the variational distribution and the true posterior. To approximate that, we sample from all the ε_ns and conduct a forward pass with those realizations, and repeat that several times. This means each of the forward passes will be different since they have different realizations of the ε_n. Then an approximation of the KL-divergence with all those forward passes and a single backward pass is conducted with that estimate. This process is essentially a complicated Monte Carlo approximation to the KL divergence. In the last paragraph, we used the example of a Gaussian to reparametrize. That is because BNNs conventionally utilize Gaussian variational distributions. In those cases the ε_n's are iid $\mathcal{N}(0,1)$. The reparametrization must still be done for novel variational distributions but is often far less trivial. That was the case for our upcoming novel variational distribution in Section 4.4.

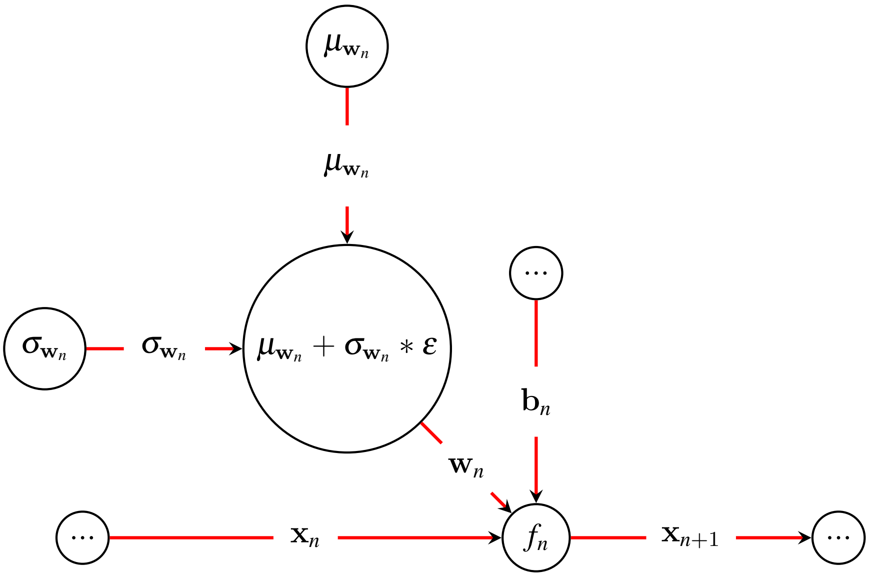


Figure 2.4: Reparametrized Computation graph for Bayesian Forward Pass

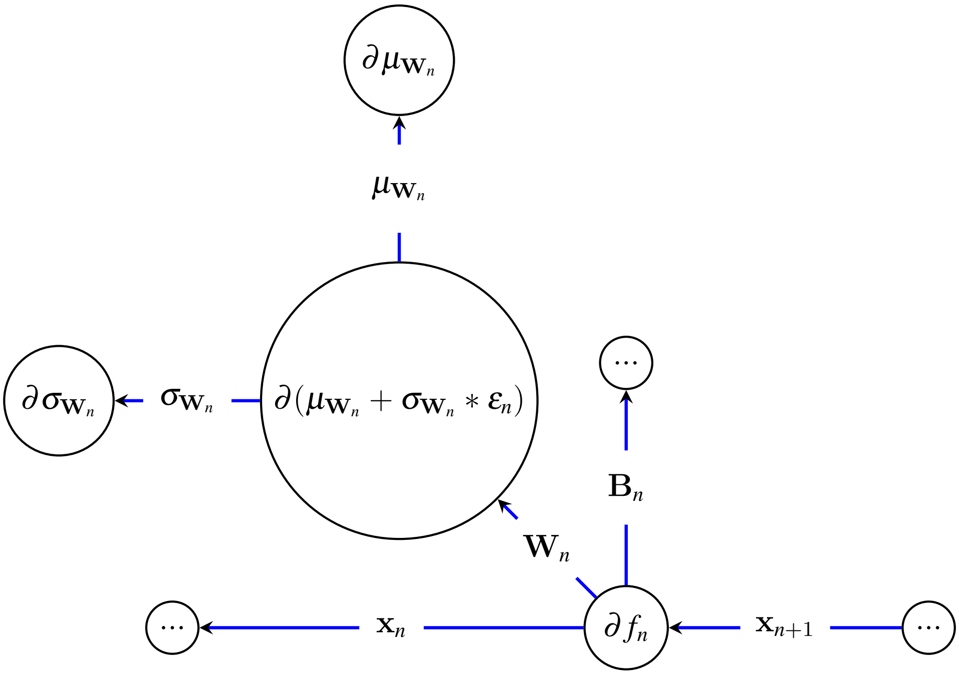


Figure 2.5: Reparametrized Bayesian Computation graph for Backwards Propagation

2.3.2 Variational Inference Bayesian Neural Networks and their Intuition

Equation (2.2) is called the evidence lower bound (ELBO) because it is the lower bound on the log-likelihood (evidence) of our data. For BNNs, the minuend and subtrahend there are often called the KL-loss and the Reconstruction-loss, respectively. That is because the first term coincides with the definition for the forward KL divergence between our prior and our VD, while the second term coincides with the expectation of the likelihood of our data. Consequently, a helpful, intuitive interpretation is to view the KL-loss as replacing the $\mathcal{L}^2$ regularization term in FNNs and the reconstruction loss as maximizing the likelihood, analogous to the traditional sum of squares loss. This is relevant to the goal of this thesis, which will be explored further in the next section. In short, the KL-loss is the ELBO part that induces this $\mathcal{L}^2$ regularization; see Subsection 4.2.1. Therefore it is even common in Bayesian Deep Learning to adjust this term directly to "temper" it, for example, since it is in practice known to represent the regularizing effects of the prior. Even though that direct interference often has little statistical motivation as such. The idea behind

BNNs is explored more at length in the Introduction. Restated, perhaps there are "actual" Bayesian relationships in the data that we are working with, but this framework allows for accounting for the uncertainty of the parameters in a better manner than in the FNN case. That is the main reason why we bother making something already complicated even more complicated. BNNs accommodate evaluation of both parameter and model uncertainty (Hubin and Storvik, 2019).

Chapter 3

Reproducibility and Replicability

In this chapter, we argue for the importance of reproducible and replicable science in general. We show why reproducibility, especially computationally, is crucial for modern machine learning. Lastly, we demonstrate how we achieved that in practice.

3.1 Introduction

Computational reproducibility is easy to achieve if efforts are made early in a project. If delayed, it becomes less and less feasible later on. Computational reproducibility is especially relevant for deep learning. Constraints in computational recourses are widespread in deep learning, which exasperates the difficulties. Furthermore, the body of libraries used to conduct deep learning is vast. Oddities may thus arise from seemingly innocuous changes to the computational environment. Downloading new libraries or updating existing ones are examples of this. This chapter is, therefore, deliberately placed before the experimental chapter. Because effort into reproducibility *before* starting the implementation will be greatly rewarded. The importance of reproducible science can not be understated. Deep learning models and their performance have far outpaced our understanding of them. Reproducible investigations of them assist the scientific community in understanding. Therefore, we prioritize the reproducibility of this thesis. Because we want to make our tiny contribution to the larger effort of science. This section thus goes through how we achieve computational reproducibility in practice. Let us open this chapter with a bold statement, and see whether it holds merit: **"Nonreproducible science" is an oxymoron.**

3.2 Terminology

Confounding use of terminology is a problem in scientific publications across scientific disciplines. "Reproducible" is not exempt, often confounded with "replicable". There are three main usage groups of the terms. The first group uses reproducible to denote "new data and new methods in an independent study=same findings." They use "replicable" to denote "same data and same methods=same results." The second group interchanges the two terms. Of course, there is a third group that makes no distinction between them. They interchange the two terms as if they refer to the same thing without making it too clear what that is. This problem is well addressed by (Barba, 2018). We will use what is called the Claerbout-Donoho-Peng convention, meaning we let the terms follow the first group.

3.3 The Philosophical Foundation of Science

An attempt at defining science is made by (Richards, 1928):

"The systematic description of phenomena."

This definition is quite broad. However, the word phenomenon means observable event. Therefore, the definition refers to the gain of knowledge by systematic observation.

Another attempt at a definition of science from Oxford Reference (2019) reads as follows:

"The intellectual and practical activity encompassing the systematic study of the structure and behavior of the physical and natural world through observation and experiment."

Neither does this definition specify what it means by "systematic," so we will use the definition from the same people: "done or acting according to a fixed plan or system." These two definitions are not enough to conclude with statistical significance that there is philosophical consensus that science necessarily must be systematic. However, this was meant to evoke the memory of a reader that already, either consciously or subconsciously, agreed. If a scientist were to look at a potato, i.e., observe it. Occasionally pick it up and then drop it again. While doing this, it starts to rain. Say the scientist dropped the potato many times in the exact same manner while writing down all that was observed. The rain continued the whole time. Then the scientist concludes that the potato causes rain. This procedure would satisfy *a* systematic observation of the structure and behavior of the physical and natural world through observation and experiment. Since the potato was observed and intervention on the potato was made, according to a fixed plan, thus following both our definitions of science. The scientific conclusion is that dropping that potato causes rain. However, most would agree that this is not science, at least not good science. We would object to the scientist's conclusion from their observation as "untrue" or a variant of that. The concept of truth or its variations never entered our definition of science, and we want to avoid getting into that part. Progress there would be intractable for this thesis. However, both definitions have "systematic" and "study/observation" in common. This example might have been somewhat pessimistic since we argue that completely wrong conclusions can be made from systematic investigations. However, the point of the example was not to criticize science as a whole. It was rather to implore us to, as researchers, to *at least* manage to be systematic in our investigations.

Fully fleshed out, the systematic part of the definition refers to the fact that we observe the world with a well-formulated hypothesis in our currently proven or assumed body of scientific truths. We then observe nature either as is or in an experiment we fashion. Using the framework of previous science to interpret these results, we evaluate if the observations support or contradict our hypothesis. In 1600 we would now be done. Certain parts of the scientific community could seem to think the same. In order to decide whether or not this hypothesis or its rejection should be incorporated into our body of scientific knowledge, there must be a converging body of evidence from several independent studies. If this happens, we are done, and the knowledge is admitted. This is why we, and the philosophical or definitional consensus, emphasized the *systematic* part in the definitions of science. In order to improve how systematic we are, we need multiple stages of reproduction and replication of scientific results. That is why we began this section with that deliberately bold statement. Since both replication and reproduction of results are essential to the inherent systematic property of science.

3.4 Reproducibility's Importance for Modern Science

In the earlier potato example, everyone would agree that that is insufficient evidence to conclude that dropping the potato causes rain. The reader probably also agrees that this should not warrant the scientific community to prioritize the investigation of my claim. Instead, I should do it several times in different settings, aggregate my observations into a dataset, and statistically evaluate my findings. Say I write a report stating the correspondence is genuine and include a summary of my investigation of the data and the methods used in my code. At this point, many might argue that an investigation from the scientific community is now warranted. However, we argue that we are still not done. A common source of error happens during the stages where data is handled. For our potato example, say I published the code from my data preprocessing. Here another researcher discovered that I had overwritten my binary "rain" column with the observations from the binary "potato drop" column. Therefore I found 1-1 correspondence over ten thousand potato drops when the correspondence was not there at all. I wrote this last part in first person deliberately since surprisingly "trivial" problems find their way into peer-reviewed journals. To address issues like this, humility in accepting one's own propensity for embarrassing errors is paramount. Therefore, I wrote both this paragraph in the first person and have made efforts to be reproducible since the start of my thesis. Since my part or all of my results could be invalid due to a computational error. A particularly egregious example is in Kaiser (2015), where there was active fraud. Open access to data and code would have helped discover the fraud much faster, perhaps saving lives. However, reproducibility has a long list of advantages in addition to correcting embarrassing mistakes or fraud (Macleod et al., 2014; Chalmers and Glasziou, 2009). It may be that another researcher's surprisingly small changes to my work could result in much more significant findings than what I managed on my own. So let us dive into how to achieve it.

3.5 Computational Reproducibility in Practice

Reproducibility is not relevant to all the stages of the scientific process. However, in most modern science, data and code will be involved. From that point onwards, efforts into reproducibility provide far more reward than it requires investment. Implementing reproducibility in practice means making sure other researchers can redo or reproduce our computation. We will explain the tools we believe are the most relevant, most of which we have used ourselves.

3.5.1 Computational Reproducibility in this thesis

We include permanent links to all the datasets. We have not been completely consistent with including permanent links to all papers referenced as well, but almost. In the cases where one was not included, we have listed a URL we have no reason to believe will disappear anytime soon. In our opinion, more important is including the data. Since even if there was an error in the preprocessing in the included code, debugging without seeing the same outputs as us is much more difficult. The ideal way of doing so, which we have managed to do, is constructing the code s.t. when others run it, the code downloads the data from a permanent link on its own. The exception would be the vision data, but that is trivially built into Torchvision and loaded from there.

We have utilized Git (Git, 2023; Chacon and Straub, 2014). Git is a program that tracks all changes a coder conducts to their code when the coder "commits" their changes to a repository. The structure is, in short, a directed acyclical graph (DAG) of all the versions of the code that the user has committed to the graph. This is great for a researcher tracking their entire scientific progress. Commits should be frequent and contain comments that explain what has been done to the best of the researcher's ability. This history of commits is also great for the researcher to spot where they might have made a mistake or for other researchers to stop a mistake in their code or data processing. This also helps science to be more open and allows researchers to learn how other researchers conduct their research. GitHub is an almost universal website for backup-storage of commits, the collaboration between multiple Git users, and publishing the git repositories.

Jupyter notebooks Kluyver et al. (2016); Granger and Perez (2021) are software facilitating interactive coding. Their main point is that several files can be imported into the cells in the notebook, and the output can be saved. In this way, Notebooks provide a summary of how the code was used and the results, allowing other researchers not necessarily to look through the entirety of the codework. This works well with Git, where the outputs and commands used to yield them can be committed at each step, accommodating version control.

Conda environments (Anaconda, 2020) helps set up the appropriate computational environment on their machine. It lets us specify all libraries and dependencies to be of consistent version. We have created such a Conda environment and attached a .yml file that allows other researchers to recreate the Conda environment we have used. The .yml file can be viewed as a recipe for Conda to recreate the computational environment on another machine. This eliminates several sources of errors by allowing other researchers to have a computational environment very close to our own. Not only does it help other researchers, but also the original publisher. Sometimes code that used to work suddenly no longer does. A reason for that could be that when downloading new libraries and amending others to make them compatible, the computational environment may have changed such that the code malfunctions. For the ideal use of Conda, the researcher should export their environment at every installation of a new package at that step and commit it to the Git DAG. Optimally, the code should be run, and the output on the Jupyter Notebook also be committed. If the researcher had then been diligent in exporting the environment every time it was amended and committed it to the repository, the researcher could look back to the last well-functioning commit and recreate the environment from the .yml file there. Whether or not that is reproducibility is hard to answer, since we usually implicitly use that term to refer to other researchers' ability to reproduce. However, we think it is a good idea to start with being able to reproduce your own results, which Conda helps with. We would like to, in particular, mention the analogous Mamba Mam (2020). Mamba is a lot faster computationally than Conda.

3.5.2 Additional Computational Reproducibility

Well-documented readable code is an essential part of coding. This boils down to coding code with a reasonably intuitive structure from the start while concurrently adding comments, i.e., documentation, to explain the code further. McConnell (2004) is a resource for this. Although efforts were made, we do not consider the repo available at the time of publication to be of sufficient readability or documentation for a scientific publication.

Binder Jupyter et al. (2018), is a project website that constructs from a Jupyter notebook from the git repo of the research. What we should have done was to store a pre-trained model such that other researchers could investigate our models on the interactive Binder site. Although training the BNNs is too computationally expensive for Binder, the benefit is somewhat limited. Other researchers without our GPUs would be limited to rerunning different tests with the same pre-trained models.

Utilizing this framework and the tools listed could greatly assist reproducibility. That accommodates less erroneous and cheaper science on its own. Furthermore, it accommodates collaboration across scientists. Another term we have not delved into yet is that of open and accessible science. The framework we have summarized, and the tools we demonstrated to achieve it, also allow a much larger variety of people to partake in the scientific process. Other researchers do not need direct access to the University of Oslo's mainframe to run our code. Although to train the models, they would either need significant GPU resources or significant patience. In AI research, this is unavoidable since substantial computation is necessary. This is also why this whole framework allows for more inclusive science. This is great for the newly included scientists who want to contribute and for science as a whole. As more scientists working together in a well-functioning collaborative environment will scientifically outperform fewer scientists working less closely together.

3.5.3 Replicability

A general issue in deep learning is that one seed may yield a particular result while another seed yields another. Often the two results are significantly different.[1] The

[1] Whether this is strictly reproducibility or replicability is hard to determine.

obvious solution is to run many different seeds and evaluate the aggregate results together. This is easier said than done for deep learning. As mentioned, the problem is that the computational demands in deep learning are inherently large, sometimes massive. Hence, the community often only tests for one or a few seeds. We have run into this issue ourselves as well. Only one seed per experiment was affordable within our computational budget.

Accommodating reproducibility will allow other later researchers to aggregate the code of earlier publications and test them for different seeds if they can afford to do so. Perhaps no other researchers could prioritize doing so initially. However, should the results become relevant later, the reproducible framework allows later researchers with later and more powerful hardware to recreate the Conda environment of earlier researchers. Thus, they can test different seeds more easily with their more powerful hardware.

Part II

The Hyperspherical Uniform Prior

Chapter 4

Priors and Variational Distributions in Bayesian Neural Networks.

This chapter shows the earlier work in novel priors and variational distributions for Bayesian neural networks. We describe the issues we believe are the most important and propose priors and variational distributions we believe best redress these.

4.1 Introduction

Optimizing a neural network BNN or FNN involves small optimization steps and adjusting them based on the norm of the gradient. In theory, there is already reason to suspect that the gradient norm is important for optimization (Bottou et al., 2016). Adam can mathematically use arbitrarily small or large gradients for optimization. However, being a first-order method, it is not invariant to the gradient norm, and that can cause problems. Furthermore, setting one learning rate for all parameters can be problematic since it is not obvious it will account for their differing scales. In practice, the norm can cause even more problems. Computers do not handle arbitrarily large or small numbers trivially. The chain rule derived an expression for the gradient can result in components too large or too small, causing overflow or a gradient that disappears to zero. Over and underflow may lead to computational inaccuracies or even a crash. This problem is referred to as vanishing and exploding gradients in the literature, first specified Hochreiter (1998). Keeping the gradient norm within bounds is, therefore, crucial.

As mentioned in Section 2.1, Batch Normalization (Ioffe and Szegedy, 2015) addresses this issue. It often works well in practice. However, the mathematical or modeling theoretical motivation for doing so is not convincing. Theoretically, it is a curiosity; although it should not harm, it probably will not do any good. The Bayesian interpretation is analogous in that Batch Normalization is merely a numerical heuristic. Implementation-wise, there are also several problems with it. Wu and He (2018); Arpit et al. (2016) explore some of these problems and propose solutions. Although Batch Normalization and Dropout work well independently, they sometimes have problems being used together in the FNN case (Li et al., 2018). As mentioned, the Gaussian prior has also been shown to work well. One reason for this might be that it induces Dropout (Gal and Ghahramani, 2015). By induces, we mean that Dropout could be viewed as an approximation to an underlying Bayesian structure. Novel prior work has already been done in this field (Polson and Rockova, 2018). However, we would like to specify a prior similar to the Gaussian in hopes that it will inherit its beneficial properties. Specifying a prior and variational distribution that can induce both benefits without the seeming conflict could be fruitful. Furthermore, normalization of weights as a replacement for Batch Normalization has already indicated usefulness in the FNN case (Salimans and Kingma, 2016). In addition, this restriction should make the network have fewer combinations of weights and biases that result in the same loss. **Specifying a prior that restricts the norm of the gradient close to 1 while retaining the desirable qualities of the Isotropic Gaussian** is a primary goal of this thesis.

4.2 Isotropic Gaussian Prior

As mentioned earlier, the Isotropic Gaussian, isotropic meaning independent and identical variance along all dimensions, is the standard prior used for BNNs (Dusenberry et al., 2020; Zhang et al., 2019; Immer et al., 2020; Louizos and Welling, 2017; Hernandez-Lobato and Adams, 2015; Neal, 1996). In order to find more appropriate novel priors, we first need to understand why the Isotropic Gaussian has worked so well. Therefore, we wish to specify novel priors that inherit as much desirable traits from the Gaussian as possible while hopefully adding more.

4.2.1 Relationship to $\mathcal{L}^2$ regularization

Both $\mathcal{L}^2$ regularization and the Gaussian prior have worked well for FNNs and BNNs. In order to specify a more appropriate prior than the Gaussian, it might be fruitful to understand its relationship to $\mathcal{L}^2$ regularization. Showing analytical results for their respective effects in FNNs and BNNs is complicated, so we examine a more straightforward case. Let us look at a network with only one layer with no activation function, analogous to linear regression. Let $\mathbf{y}$ be our target variable, $\mathbf{x}$ be our predictor variable, and $\mathbf{w}$ be our weights. Let their true relationship be as follows:

$$\mathbf{y}^* = \mathbf{x}^\top \mathbf{w} + b + \varepsilon$$

ε is a Gaussian random variable with mean 0 and variance σ^2. We assume that b is known for simplicity and concern ourselves with $\mathbf{y} = \mathbf{y}^* - b$. For each n, the data $\mathcal{D}$ consisting of both $\mathbf{y}_n$, $\mathbf{x}_n$ will have a normal distribution. Maximum likelihood with $\mathcal{L}^2$ regularization for regression is called ridge regression. Unlike the posterior from Bayesian inference, ridge regression yields an estimate without distributions. We, therefore, compare that to the maximum aposteriori estimate, MAP for short.

We, therefore, compare the ridge regression estimate to the maximum aposteriori estimate of the parameters. The likelihood for N observations of $\mathcal{D}$ then has the following pdf:

$$p_{\mathcal{D}|\mathbf{w}}(\mathcal{D}; \mathbf{w}) = \prod_{n=1}^{N} p_{\mathcal{N}(\mathbf{x}_n^\top \mathbf{w}, \sigma^2)}(\mathcal{D}; \mathbf{w})$$

Where $p_{\mathcal{N}(\mathbf{x}_n^\top \mathbf{w}, \sigma^2)}$ denotes the pdf for a Gaussian rv with mean $\mathbf{x}_n^\top \mathbf{w}$ and variance σ^2. As we are conducting Bayesian inference for this problem, we wish to find the posterior, $p_{\mathbf{w}|\mathcal{D}}(\mathcal{D}; \mathbf{w})$. We therefore need a prior, $p_{\mathbf{w}}(\mathbf{w})$, on $\mathbf{w}$. We set a Gaussian prior with mean 0 and variance λ^{-1} along all its dimensions. Why we set it inversely will become clear later. Thus, we can invoke Bayes Theorem:

$$p_{\mathbf{w}|\mathcal{D}}(\mathcal{D}; \mathbf{w}) \propto p_{\mathbf{w}}(\mathbf{w}) p_{\mathcal{D}|\mathbf{w}}(\mathcal{D}; \mathbf{w})$$

Inserting the pdf of the likelihood into the above expression yields us:

$$p_{\mathbf{w}}(\mathbf{w}) p_{\mathcal{D}|\mathbf{w}}(\mathcal{D}; \mathbf{w}) = p_{\mathcal{N}(0, \lambda^{-1})}(\mathbf{w}) \prod_{n=1}^{N} p_{\mathcal{N}(\mathbf{w}\mathbf{x}_n, \sigma^2)}(\mathcal{D}; \mathbf{w})$$

We wish to find the MAP estimate of the term above wrt $\mathbf{w}$. Applying a strictly mononotonically increasing function will not affect the maximum estimate. The logarithm is the standard choice for such a function. As it makes the calculation much easier. Hence, we fill in the pdf's, apply a logarithm and let C denote our constants since they are invariant to the maximization:

$$\sum_{n=1}^{N} -\frac{1}{\sigma^2}(\mathbf{y}_n - \mathbf{x}_n^\top \mathbf{w})^2 - \lambda \mathbf{w}^2 + C$$

Maximizing this expression with respect to $\mathbf{w}$ yields the maximum a posteriori estimate for $\mathbf{w}$, MAP estimate for short. The $(\mathbf{y}_n - \mathbf{x}_n^\top \mathbf{w})^2$ term might be familiar. The solution for $\mathbf{w}$ for this term in isolation is the least squares solution. Meaning if we were to set lambda to a very small number, or, equivalently setting the variance very high, the optimization problem collapses to the least squares solution.[1] Now it may become clear why we specified our prior variance in such odd terms. It was to make clear that this is the $\mathcal{L}^2$ penalty term from ridge regression. Ergo, the Bayesian MAP and frequentist ridge regression solutions for a Gaussian likelihood problem are equivalent. Where the variance of the prior is inversely related to the regularization intensity in ridge regression. Hopefully, this illustrates why the terms in equation (2.2), as mentioned in Subsection 2.3.2, are interpreted as regularization and maximum likelihood, respectively.

We went through this, because of the opening statements of this thesis. Models in deep learning are performing better and better, with higher and higher parameters and capacities. As mentioned earlier, even with the enormous year-on-year increases in model complexity, regularization is still employed. So we are not merely going further and further towards overfitting in the classical sense, as we are only increasing parameters. But we retain the regularization component. The $\mathcal{L}^2$ regularization remains the gold standard for such regularization for FNNs. Although the case we proved here would only hold true for a BNN with one layer that only has weights and no biases, the Gaussian prior does in fact induces the same regularization (Figueiredo, 2003). Therefore, for the novel priors we wish to investigate here, we want to pick priors that have similar $\mathcal{L}^2$ regularization behavior on the space of weights and biases.

4.3 Hyperspherical Uniform Prior

A recently suggested class of distributions for prior and VD use is the radial directional distribution(Oh et al., 2019; Fortuin, 2021):

$$\mathbf{w} = w_r \mathbf{w}_d, \quad w_r \sim p_{\text{rad}}(w_r) \quad \mathbf{w}_d \sim p_{\text{dir}}(\mathbf{w}_d) \tag{4.1}$$

This class of distributions lets the direction and magnitude be separated. Meaning $||\mathbf{w}_d|| = 1$, while w_r remains a scalar. As mentioned, we want to select a prior that restricts the gradient close to 1. We achieve this by selecting a radial-directional

[1] This is relevant since in the PyTorch implementation we occasionally set the variance of our priors to infinity. In the code referred to as "Tempering" the priors. This is done by setting the "KL-loss" term in (2.2), that is, in the loss function of the whole network, to 0.

prior with all its radial mass on one. That means a completely informative prior radially.

Furthermore, we see no obvious reason to introduce prior information along the directional component. Therefore we let the directional component remain uniform. The Hyperspherical distribution describes these two properties. Let Γ denote the gamma function. The area of the hypersphere with dimension m is then expressed by:

$$\frac{2(\pi^{m/2})}{\Gamma(m/2)}$$

Meaning the probability density function of a Hyperspherical Uniform rv, $p_{HU}(\mathbf{w})$, becomes:

$$p_{HU}(\mathbf{w}) = \begin{cases} \frac{2(\pi^{m/2})}{\Gamma(m/2)} & ||\mathbf{w}|| = 1 \\ 0 & ||\mathbf{w}|| \neq 1 \end{cases}$$

The Hyperspherical Uniform prior should not be placed on $\mathbf{W}$ in its entirety directly. That does not give us the backward pass we want. Instead, we should use one HU prior for each $\mathbf{W}_j$. However, we can use even more HU priors by setting one separately per $\mathbf{W}_{j,i}$. We always place one HU prior per $\mathbf{B}_j$ for the biases. We do not need an HU for the biases to get the desirable gradient properties. However, keeping what Adam optimizes on the same scale is beneficial. Therefore we place the HU on the biases as well.

This distribution is uniform along the hypersphere. Therefore, it is called the Hyperspherical Uniform distribution, HU for short. The "KL-loss" term from (2.2) just becomes the same $\frac{2(\pi^{m/2})}{\Gamma(m/2)}$ scalar everywhere and does not contribute to the gradient. Since it is always the same scalar everywhere. Implementation-wise, we do the restriction to norm one is done in selecting VD in the .[2]

Furthermore, if some connections into a neuron are large, the others are forced by the HU prior to being very small. If they are small enough to make no meaningful difference, then this is analogous to what is referred to as parameter sparsity in the statistical literature. Here some are precisely equal to 0. There is reason to suspect sparsity is desirable (Vadera et al., 2022; Hoefler et al., 2021). Therefore, our HU prior, in forcing some parameters to be close to zero, may have analogous benefits to that parameter sparsity.

[2] Therefore, we omit calculating this in the code. Because it is just the same scalar everywhere.

4.3.1 Adam's aptitude optimizing differing scale parameters

First-order methods, like SGD and its adaptive Adam variant, are not suited for optimizing parameters at different scales (Zhang et al., 2017; Bottou et al., 2016). Bottou et al. (2016) illustrates some issues, which we reiterate a short summary of here. Let θ denote the parameters determining the variational distribution, i.e. the parameters we optimize in BNNs. Observe the simplified case of the full gradient descent method, with a constant learning rate:

$$\theta_{t+1} \leftarrow \theta_t - \alpha_t \nabla L\left(\theta_t\right) \tag{4.2}$$

Where t denotes the optimization step, α the learning rate, and L is the loss. Observe how a linear transformation, or rescaling, of the parameters affects the optimization. Let C be a positive definite matrix, and consider $\min_{\tilde{\theta}} L(C\theta^*)$. The full gradient iteration for this problem becomes:

$$\theta^*{}_{t+1} \leftarrow \theta^*{}_t - \alpha_t C \nabla L\left(C\theta^*{}_t\right)$$

which, after scaling by C and defining $\{\theta_t\} := \{C\theta^*{}_t\}$, corresponds to:

$$\theta_{t+1} \leftarrow \theta_t - \alpha_t C^2 \nabla L\left(\theta_t\right) \tag{4.3}$$

Comparing equation 4.2 with equation 4.3, we see that the behavior of the algorithm changes under a change of variables. For instance, when L is strongly convex quadratic with unique minimizer $\tilde{\theta}$, the full gradient method equation 4.2 generally requires many iterations to approach the minimizer.

The momentum variant property of Adam may be viewed as an approximation to the second derivative. See subsection 2.1.1. Thus Adam would be a pseudo-second-order optimization method. Adam's variation of momentum consists of remembering the last few gradients and comparing whether they directionally agree. This approximation is not a localized estimate of the second derivative. Hence, it is probably not a good enough approximator to the second derivative to resolve this weakness. The literature provides no empirical evidence indicating that Adam is exempt from the differing scales problem. Therefore, Adam should work better with parameters on the same scale. Normalization should thus make the loss landscape more amenable to optimization. That is why going through autograd and trying to get down to the computational details of the gradient's size is calculated to be in practice. A gradient of inappropriate size will hurt the optimization in first-order methods, like Adam. Therefore, we hypothesize that hyperspherical restriction of both the weights and biases should increase test performance.

4.3.2 Replacing Batch Normalization

This manner of gradient restriction is much more conducive to Bayesian interpretation. Theoretically, excluding the heuristic Batch Normalization could facilitate increased performance and reduce computational expense. This restriction should, theoretically, incorporate all the benefits listed in this section thus far. We will do this by specifying the prior and VD s.t. the weights are restricted to the unit hypersphere. From the equations in subsection 2.1.3, we know that the layerwise weights', $\mathbf{W}_j$, the norm can make the gradient's norm arbitrarily large or small. The partial derivative of a given $\mathbf{W}_j$ or $\mathbf{B}_j$ is multiplied by the partial derivative of all the $\mathbf{W}_j$s after that layer. Ergo, restricting the norm of the weights per layer close to one will, ergo, restrict the gradient's norm close to 1 as well. By close to one, we do not necessarily mean that the norm is between 0.5 and 1.5. Instead, on a reasonable scale, the computer should handle it well. Between 0.05 and 50, for example.

4.3.2.1 Gradient Computational Issue of Batch Normalization

Furthermore, while it remains a heuristic theoretically, it is more problematic in practice. The backpropagation for Batch Normalization is significantly more complicated than the backpropagation for a linear or activation function (Agrawal, 2023). Remember from subsection 2.1.3, that when we backpropagate, utilizing the chain rule, the partial derivative of every applied function will be applied in reverse. So although one backward pass through one derivative of a batch normalized layer should not be too problematic, the issue will compound over deeper networks. Therefore, we can suspect the numerical stability of a HU prior BNN to be greater than a Gaussian BNN in this regard, and that this advantage will grow for deeper networks.

4.3.3 Summary: Two Compounding Problems Solved

Adam is not scale-invariant and can work better with parameters on the same scale. Furthermore, during the backward pass, the gradient may, at some point, either explode or implode. That can either crash the program or cause numerical inaccuracies that exacerbate Adams's scale inaccuracy of the optimization step size of each parameter. Specifying the prior such that $||\mathbf{W}_j|| = ||\mathbf{B}_j|| = 1$ directly helps to let Adam more easily calculate an appropriate gradient step by setting all the parameters on a similar scale, but also computationally by greatly reducing the propensity for gradient explosion or implosion. Lastly, that gradient solution allows us to discard the heuristic Batch Normalization. Perhaps that also helps with performance, uncertainty estimation, and making the models more interpretable by operating in a more mathematically supported, or at least theoretically understood Bayesian framework.

4.3.4 Prior Use of Radial-Directional Distribution

Recent results indicate that a Variational Autoencoder (Kingma and Welling, 2022) 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). In Variational Autoencoders, the goal is to make inferences on the distribution of the data presented. The prior and VD specified there is for inference on the general distribution of the dataset. Their results indicated that the HU prior and vMF VD are more appropriate than the Gaussian for datasets with latent hyperspherical properties. The prior and VD we specify are analogous, but instead of prediction. Thus advantages for inference on the distribution of datasets are not obviously transferrable to prediction. A BNN that describes the systematic relationship between predictor and target variables differs from a VAE that describes the data distribution. However, making inferences on a dataset as a whole and making inferences between the target and predictor random variables still holds an analogy. Therefore, some of the advantages may transfer to our prediction purposes.

For Mean Field VI BNNs, the radial-directional distribution (4.1) has been applied to the VD (Farquhar et al., 2019). Thus explicitly assuming the form of the posterior and implicitly changing the prior. There, a Gaussian VD was set over the radial dimension, and they set a uniform VD over the angles.

4.4 The von Mises-Fisher Variational Distribution

With the prior choice made, it remains to find an appropriate VD. Let us assume that our prior is "true ." True in the sense that a BNN with the exact number of layers and nodes we are trying at a given moment *is* the true relationship between predictor and target variables. From that, the true posterior over the parameter space has a radial component equal to 1. Therefore a VD like the one used in Farquhar et al. (2019) is not too helpful. Although the uniform distribution over the angles is desirable, setting a Gaussian on a component that is always one is strange. Setting the HU as our VD is not helpful since it is always uniform. Another radial-directional distribution is the von Mises-Fisher distribution (vMF). This distribution can set its radial component equal to 1, as we want. The pdf of this distribution is expressed in the following manner:

$$f_p(\mathbf{w};\mu,\kappa) = C_p(\kappa)\exp(\kappa\mu^{\mathsf{T}}\mathbf{w})$$

where $||x|| = ||\mu|| = 1$. p denotes their dimensionality. $C_p(\kappa)$ is defined by:

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

Where I_v denotes the modified Bessel function of the first kind at order v.

μ and κ are the mean direction and concentration parameter. The greater the value of κ, the higher the concentration of the distribution around the mean direction μ. This parameter is how the variance of the vMF is specified. Greater κ means lower variance, hence the "concentration parameter" name. Analogously to (4.1), the vMF is only defined along the unit sphere.

Both μ and κ denote the mean and variance along the directional component. The radial component, however, desirably remains equal to 1 with our specified $C_p(\kappa)$. Thus, the mean and variance along the radial component come out to 1 and 0 respectively, as we wanted from the specification of our HU prior. κ is a scalar. Meaning the vMF has an isotropic variance. That is in contrast to the Gaussian prior and VD.

Since we would like to keep computational costs as low as possible, avoiding both Batch Normalization and Dropout is helpful. Neither of them are too intensive for the forward pass, especially Dropout. However, Batch Normalization entails increasing the computational demand during the backward pass (Agrawal, 2023). Accordingly, this novel VD-prior combination will still have an architecture significantly more expensive per depth than its frequentist counterpart. However, the goal is to compete with Gaussian BNNs for applications where they are already preferred over FNNs, i.e. where robustness and especially uncertainty estimates are desired. In that case, we have hopes to be computationally competitive. Our PyTorch implementation of the vMF sampling function was based on Kim (2021).

4.4.1 Two ways of utilizing von Mises-Fisher

There are two ways of specifying a vMF VD over our weights $\mathbf{W}$ and biases $\mathbf{B}$. In both solutions, $\mathbf{B}$ is one vMF for each layer. We place one vMF on each $\mathbf{B}_j$. But there are two solutions for $\mathbf{W}$:

The layerwise vMF is where we specify one vMF over the entire $\mathbf{W}_j$ tensor between two layers. Restricting the $\mathbf{W}_j$ in this manner yields the advantage of keeping the norm of the backward pass quite tightly bounded. See Subsection 2.1.3. However, the downside of this solution is that the variance along the entire $\mathbf{W}_j$ tensor between two layers is isotropic.

The nodewise vMF is where we specify multiple vMF's for each node of $\mathbf{W}_j$. Here we select each component $\mathbf{W}_{j,i}$ that is multiplied with $\mathbf{x}$, and let this subset of $\mathbf{W}_j$ be a vMF. I.e. we are letting all the connections into each next node be specified as one vMF. The advantage here is that the restrictions on the variance are less severe. However, the desirable restriction of the tightly restricted backward pass is partially lost. We are instead left with only restricting the input to each node to be of norm 1. That restricts the norm of the backward pass much more than we would see in the Gaussian case. Since the norm will then formally be restricted to the width of that layer, being the sum of the norm of all the nodes. Specifying the vMF per node means each vMF distribution will have a lower dimensionality. That is advantageous as the vMF is much cheaper computationally for lower dimensions.

Since the HU prior is uniform, we never calculate it in the code. Since it will always just remain a constant. Therefore, when we select to either use the Nodewise or layerwise vMF, we have also implicitly selected whether we place the HU per $\mathbf{W}_{j,i}$ or $\mathbf{W}_j$ respectively.

Chapter 5

Limitations and Solutions

In this chapter, we describe the main limitations and propose solutions to the Hyperspherical Uniform Prior and von Mises-Fisher variational distribution.

5.1 Restricted Optimization in PyTorch: "Ghost Mu"

PyTorch itself has a fundamental limitation regarding spherical learning. Parameters, usually μ, are only defined for the unit hypersphere here. However, PyTorch does not support the optimization of parameters under any restriction. Therefore, we must register our μ as a parameter that PyTorch optimizes across the entire real line. Afterward, we project it unto the unit hypersphere for use in the spherical random variable. This is why we are introducing the term "Ghost Mu" since PyTorch optimizes parameters outside their domain. In practice, we found that the norm of the "Ghost Mu"s has an unfortunate tendency to grow for every epoch. The growing "Ghost Mu"s introduce increasing numerical issues as the epochs continue. This is only the case for the optimization of restricted parameters. The Gaussian VD has parameters defined for the entire real line. Hence, the mus never get projected there. In that case, the mathematical μs coincide directly with the parameters registered in PyTorch. The "Ghost Mu" problem is present for all spherical VDs since they all have mus only defined for the unit hypersphere. Hence, the Power-Spherical or Projected Normal distributions would not ameliorate this problem.

Statistically or mathematically, there is no trivial interpretation of what PyTorch is doing. The mathematical concept of μ does not exist, while PyTorch optimizes the parameters for spherical learning. There are only the registered parameters that exist on the entire real line and their projection unto the unit hypersphere before use. After the projection, the mathematical μ "appears." Therefore, spherical learning in PyTorch must be viewed as partly computationally heuristic from the statistical view. However, heuristic approaches are common and work well in practice in deep learning. So we cannot expect to get rid of them entirely anytime soon.

5.1.1 The Favourable Gradient Properties of the Hyperspherical Unifrom Prior are not compromised by "Ghost Mu"

However, although it may be obvious, the desirable restrictions we have placed on our gradient are not compromised by the "Ghost Mus." During backpropagation, they are first projected. Therefore, all of the desirable gradient norm properties discussed are retained no matter how far the "Ghost Mu"s diverge. It is merely the projection step from the registered parameters into the domain of the actual parameters where we are concerned with numerical instability, especially for deeper networks.

5.1.2 Normalized Initialization Inheritance.

Directly addressing the "Ghost Mu" problem would entail reworking PyTorch internally, down to the autograd itself, in order to support restricted optimization. Internally restricted optimization is the most optimal solution. However, that would be far beyond the scope of this thesis. Perhaps that is even intractable for leading researchers since it may require rewriting the entire library. While the optimal solution may remain elusive, effective heuristic solutions can still make significant improvements in practice. Reinserting the normalized mus right after every epoch might be the most straightforward approach available. Then, the numerical inaccuracies of the Ghosts can not be compounded. However, this also proved much harder to implement than to propose. Therefore, we instead tried to re-instantiate the networks every 10-20 epochs. While re-instantiating, we let the initialization inherit the variance and means from the earlier 10 or 20 epochs. However, the means initialized to the new object are projected unto the unit hypersphere. That is why we call the procedure Normalized Initialization Inheritance (NII).

NII is not any more heuristic than not having it. Since we are merely changing *when* we have to resolve the underlying problematic heuristic. I.e., changing when the projection step is done. We are conducting a projection of a ghost earlier than PyTorch forces us to at a later stage regardless. Thus, keeping the ghosts closer to the mathematical mu we want it to represent than if we were to leave it until later. What is heuristic, is trying to optimize restricted parameters with PyTorch, since PyTorch will force all parameters to exist on the entire real line.

5.2 Computational intensity of the von Mises-Fisher

Sampling from the vMF can be computationally intense and somewhat unstable for high concentrations (Davidson et al., 2018). An idea to address this could be to utilize the Power-Spherical distribution. This distribution was already proposed for Variational Autoencoders (De Cao and Aziz, 2020). However, the Power-Spherical distribution also has isotropic covariance. This computational property is what we suspect makes the network occasionally halt its training. It gets stuck in the rejection sampling loop in our implementation of the vMF.

5.3 Isotropic Covariance of the von Mises-Fisher

The vMF has isotropic covariance along all its dimensions. That is not analogous to the Isotropic Gaussian prior, since there a non-isotropic VD is utilized. It is forcing all the variances along all the VD's dimensions to be one. This is in contrast to the conventional Gaussian Variational Distribution and could limit performance. The Projected Normal Gaussian would redress this issue, as it supports separate variance along each dimension.

Chapter 6

Experiments

This chapter tests the Hyperspherical Uniform (HU) prior and von Mises-Fisher variational (vMF) distribution in practice. We compare it to the conventional Gaussian solution and investigate the limitations and solutions from Chapter 5.

6.1 Introduction

The experimental design compares the efficacy of our vMF and NIIvMF BNNs to the conventional Gaussian BNN. "Gaussian" is short for the conventional BNN setup, with both Gaussian prior and variational distribution (VD). "vMF" is short for von Mises-Fisher and refers to the usage of the von Mises-Fisher VD with a hyperspherical uniform prior. The "NIIvMF" designation is the same as the vMF, but with normalized initialization inheritance added. See Subsection 5.1.2.

We are designing experiments with four settings. In three of them, we will test classification performance, and one will also support the evaluation of uncertainty estimation. Lastly, we include one regression dataset. We have opted only to test the Nodewise vMF. The nodewise vMF is already much more computationally intensive than the Gaussian. We quickly found the layerwise vMF to be computationally intractable during the experimental setup. In each test case, we train one network. Then we sample ten realizations from that BNN and test their ensemble performance on a separate test set. We quickly found that the vMF and NIIvMF architectures work best for high learning rates and high initialization of the concentration parameter of the weights. However, both became increasingly computationally intensive as the learning rate and concentration parameter was tuned up. We, therefore, had to, throughout this chapter, balance their performance and computational demand. We report the differing learning rates for differing epochs in the experiments where we were forced to change them. The number inside the parentheses for the Hidden Layers denotes the number of nodes for that layer. Then how many such entries there are denotes how many hidden layers there are.

6.2 Phoneme data

Phoneme Testset Accuracy			
Test case	Gaussian	vMF	NII vMF
10 epochs	921/1000	928/1000	NA
30 epochs	923/1000	931/1000	931/1000
50 epochs	920/1000	926/1000	929/1000
Hyperparameters			
Prior	$\mathcal{N}(0, e^{-0.1})$	HU	HU
Learning Rate	0.02, 0.01, 0.007	0.14, 0.05, 0.03	0.14
Hidden Layers	(10, 10)	(10, 10)	(10, 10)

Table 6.1: Hyperparameters and results for the Phoneme experiment.

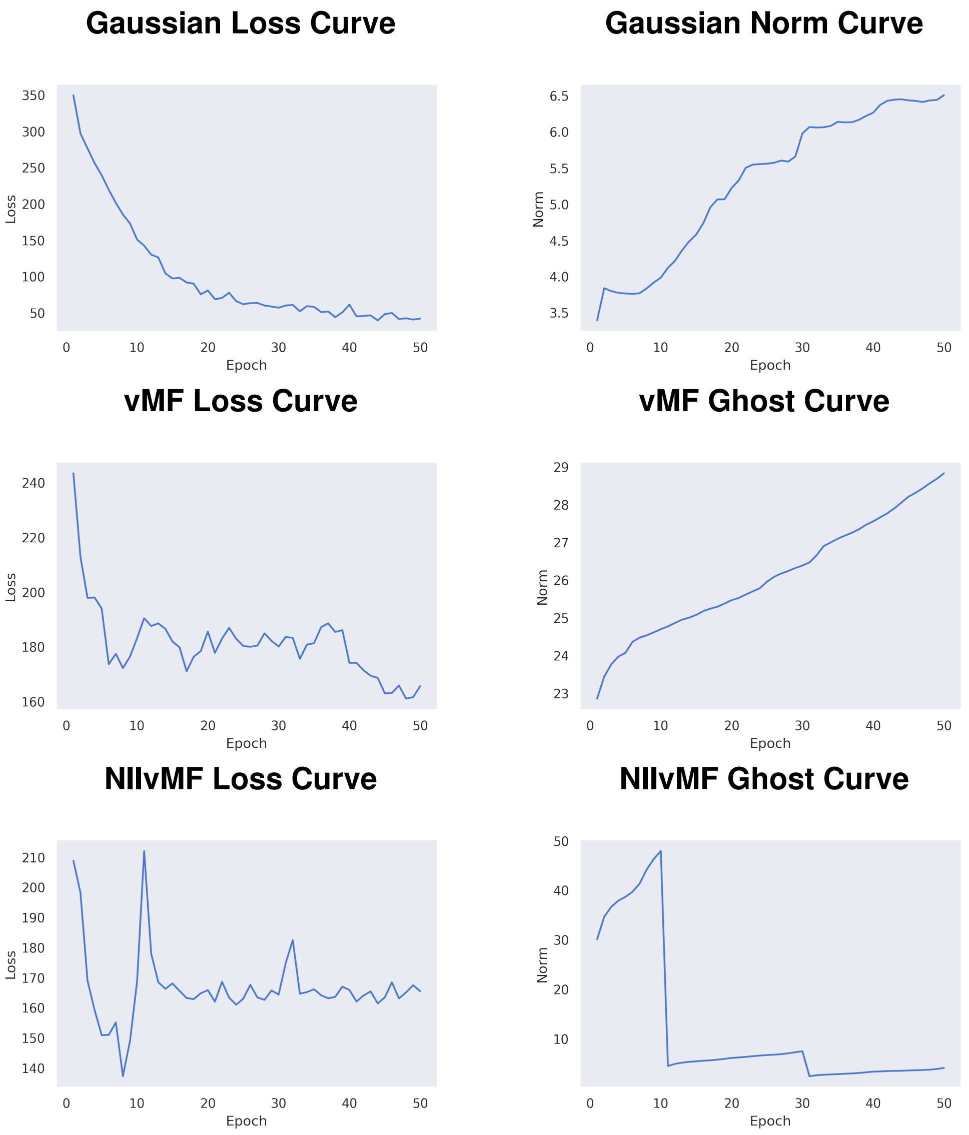
This experiment uses the phoneme data from Hastie et al. (1995). The data was extracted from the TIMIT database (TIMIT Acoustic-Phonetic Continuous Speech Corpus, NTIS, US Dept of Commerce). This data is widely used in speech recognition. The networks are tasked with predicting which of five phonemes for classification based on digitized speech. The phonemes are transcribed as follows: "sh" as in "she," "dcl" as in "dark," "it" as the vowel in "she," "aa" as the vowel in "dark," and "ao" as the first vowel in "water." From the continuous speech of 50 male speakers, 4509 speech frames of 32 msec duration were selected, and approximately two examples of each phoneme from each speaker are included as the predictor variables. Each speech frame is represented by 512 samples at a 16kHz sampling rate. Furthermore, each frame represents one of the above five phonemes. A log-periodogram was computed from each speech frame. Log-periodograms are one of several widely used methods for casting speech data in a form suitable for speech recognition. Thus, the data consist of 4509 log-periodograms of length 256, with known class (phoneme) memberships. The networks are trained on a random sample of about three-quarters of the data. Then they are tested on the remaining quarter, conveniently 1000 observations. The NII vMF is not calculated for ten epochs since NII applies normalization in intervals of 10 epochs.

Table 6.1 shows that the performance of the vMF appears marginally superior to the Gaussian's. The NIIvMF slightly outperforms the vMF again. However, we are reluctant to consider the results here significant since we should see either these results across several different seeds before concluding significance or see much larger differences.

The setup we attempted was to select the best hyperparameters for the ten-epoch case. Then we planned to run with those hyperparameters through thirty and fifty epochs. Here we ran into the computational instability of the vMF, described in Section 5.2. The training of the vMF would completely halt at around epoch 14, with a high learning rate. So we were forced to tune it down for higher epochs successively. We must also note that in the NIIvMF, Adam loses its earlier learning rate from the last epoch. We, therefore, had to guess where we thought it was.

Consequently, we applied successively lower learning rates there. For consistency, we opted to tune the hyperparameters for each case separately to give each network its optimal parameters. This should yield an equal comparison between the models. Figure 6.1 for the loss and average norm of registered $\mathbf{W}_j$ per epoch are informative. Remember, for the vMF and NIIvMF, the average norm curve refers to registered mus, not the actual mathematical mus. Therefore we refer to them as "Ghost Mu"s. See Section 5.1. The behavior for the Gaussian loss convergence is what we would expect. However, the vMF loss curve seems more uneven. The spikes at 10 and 30 in the NII vMF are not surprising. However, there appears to be no loss descent after the first ten epochs. This behavior is even more unusual than the vMFs, considering the impressive performance of the NIIvMF on the test set. We would expect a model with such an unusual loss curve not to have converged properly.

Regarding the norm curves, the Gaussian increases slower for higher epochs but does not decrease. Therefore, we could suspect that gradient explosion would be a problem for large models. Regarding the vMF's norm curve, the numerical inaccuracies would keep compounding. Although it is hard to evaluate how much of an impact the numerical inaccuracy of the projection of larger and larger mu vectors would have. The norm seemingly increases linearly with no slowing down. For the NIIvMF, we see the registered normalization effects for epochs 10 and 30 reducing the norm, as expected. However, less expected is the behavior after each normalization. The norm of the "Ghost Mu"s grows slower and slower for each application of the NII. Likely due to the decreasing learning rate.


Gaussian Loss Curve

Gaussian Norm Curve

vMF Loss Curve

vMF Ghost Curve

NIIvMF Loss Curve

NIIvMF Ghost Curve

Figure 6.1: Loss and Norm Curves for Phoneme Classification.
The left side plots plot the loss, i.e. the ELBO for the networks, per epoch. The plots in the right plot the norm of the registered parameters per epoch. For the vMF and NIIvMF, the registered parameters are not the actual parameters. Instead, the plots provide a look into what PyTorch does under the hood with them.

6.3 Simulated Data

Bivariate Gaussian Testset Accuracy			
N	Gaussian	vMF	NII vMF
50	137/200	129/200	138/200
150	484/600	497/600	499/600
250	862/1000	822/1000	853/1000
Hyperparameters			
Prior	$\mathcal{N}(0, e^{-0.1})$	HU	HU
Learning rate	0.04	0.07	0.14
Hidden Layers	(5, 5)	(5, 5)	(5, 5)

Table 6.2: Hyperparameters and results for the Simulated data experiment.

We sampled our data from five bivariate Gaussian random variables for this experiment. The networks' task is to use the two-dimensional coordinate of each observation to predict from which of the five bivariate Gaussians that point was sampled. The random variables have means $(-2, -2)$, $(-2, 2)$, $(0, 0)$, $(2, -2)$ and $(2, 2)$. They have the same variance $\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$.

We have generated three different sizes of datasets. N refers to the total number of generated observations of each class. We split the data into four-fifths for the training set and the remaining fifth for the test set. We test the networks on the three different datasets. Table 6.2 describes the test results. There we see that the NIIvMF systematically outperforms its non-NII vMF counterpart. However, the relative performance of the Gaussian and NII vMF do not differ significantly. We measured the networks' aptitude for uncertainty estimation. This aptitude is measured in cross-entropy, expressed by:

$$-\sum_{k=1}^{5} f(\mathbf{x})_k \log(f(\mathbf{x})_k)$$

Where f is the network and $\mathbf{x}$ is the covariate vector, which in our case is the cartesian product [-5,5] x [-5,5]. k denotes the class. We report the results for each case compared to the true uncertainties visually by plots in Figures 6.2 6.3 6.4. That is why bivariate Gaussians were the most practical for visualization.

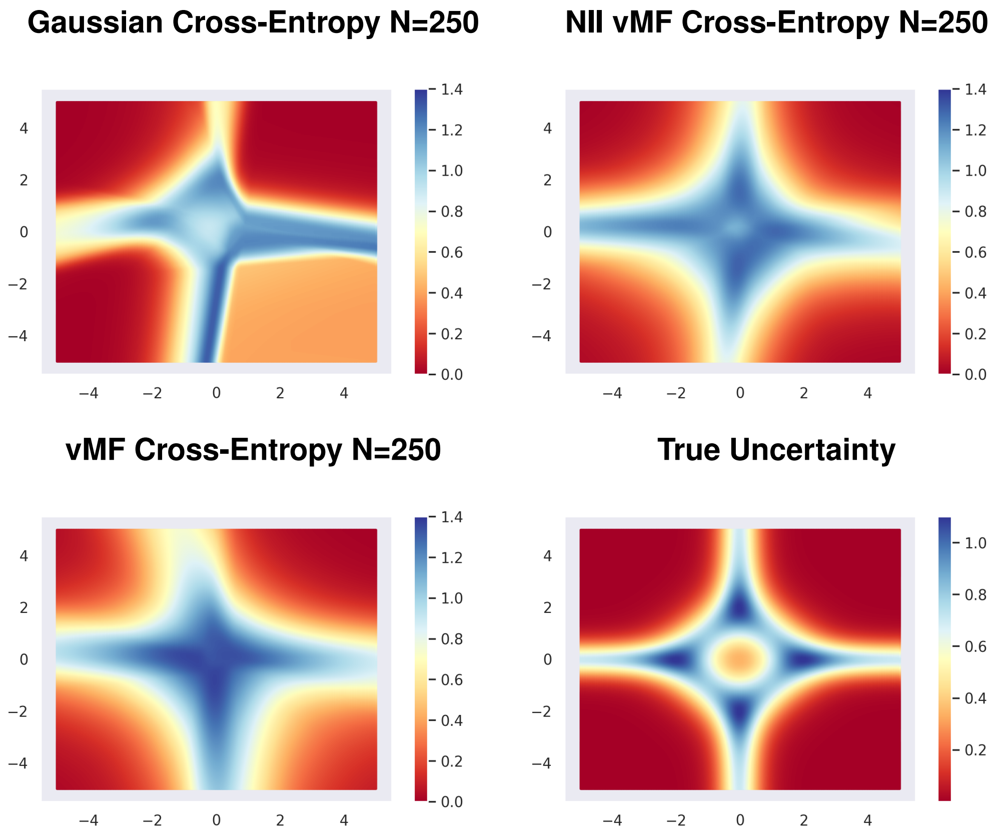


Figure 6.2: N=250: Cross-Entropy Uncertainty. z-axis: $-\sum_{k=1}^{5} f(\mathbf{x})_k \log(f(\mathbf{x})_k)$

Figure 6.2 shows that all three networks find the four quadrants where they should be more certain. The Gaussian manages to detect that there is more certainty around origo, which the NIIvMF also manages to do. The vMF does not appear to detect this. In general, the vMF and especially NII vMF appear much smoother and symmetric, which visually is more resemblant of the true uncertainty.

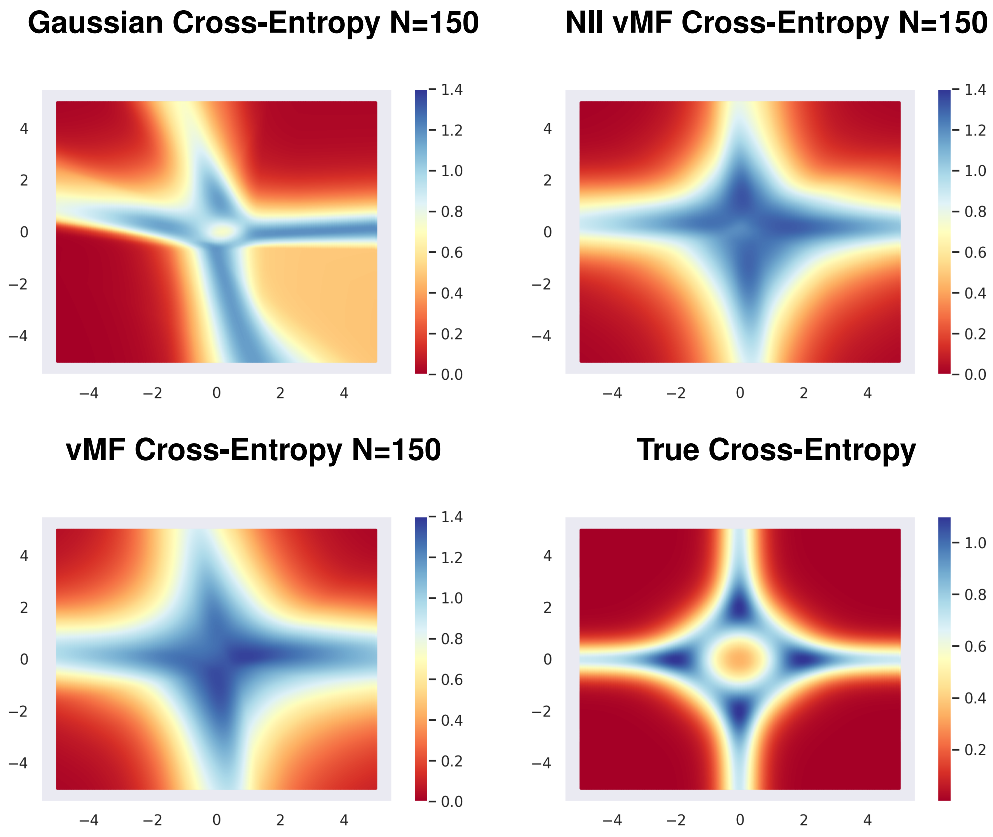


Figure 6.3: N=150: Cross-Entropy Uncertainty. z-axis: $-\sum_{k=1}^{5} f(\mathbf{x})_k \log(f(\mathbf{x})_k)$

Figure 6.3 shows, like before, that all three networks find the four quadrants where they should be more certain. This time the Gaussian appears less certain than it should in the lower right quadrant. There was already a hint of this for $N = 250$ in Figure 6.2, but this is now more pronounced. The Gaussian manages to detect that there is more certainty around origo. The NIIvMF barely manages to do so. The vMF does not appear to detect this. Again, the vMF and especially NII vMF appear much smoother and symmetric, which visually is more resemblant of the true uncertainty.

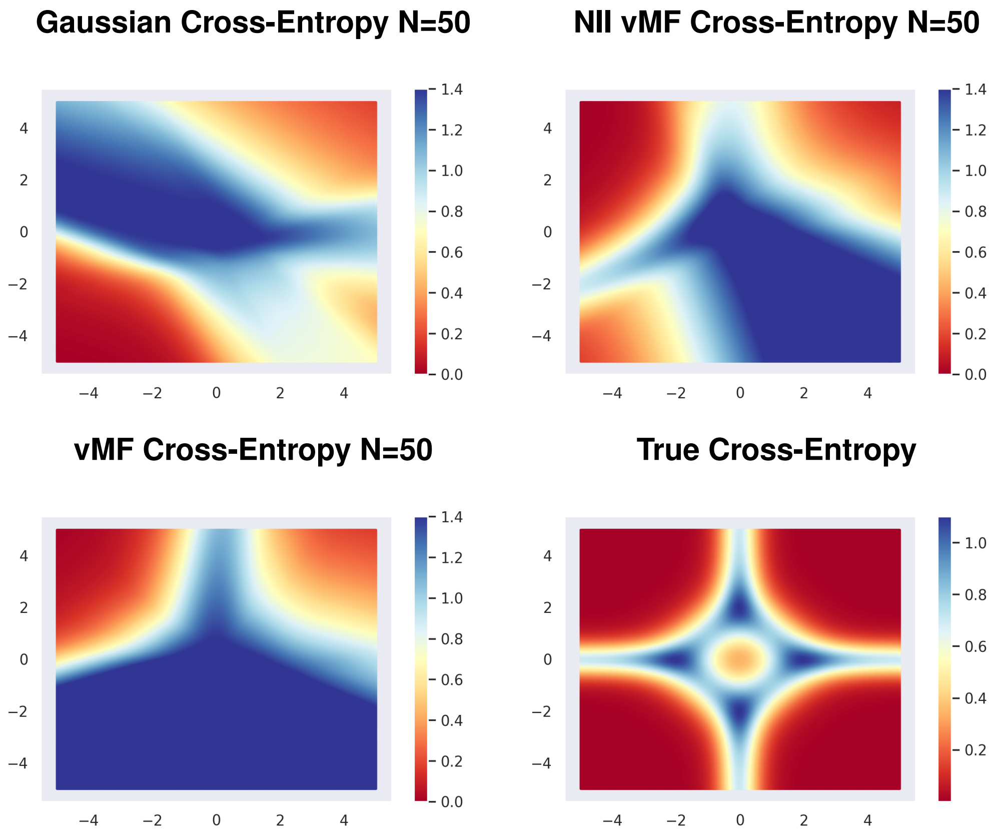


Figure 6.4: N=50: Cross-Entropy Uncertainty. z-axis: $-\sum_{k=1}^{5} f(\mathbf{x})_k \log(f(\mathbf{x})_k)$

Here in Figure 6.4, we see that all the networks are struggling. The Gaussian and NII vMF exhibit similar behavior, while the vMF appears essentially like the NII vMF but with the bottom left quadrant lost.

We considered measuring $\mathcal{L}^2$ distance between the cross-entropy of the networks and the true uncertainty. However, this could disproportionally benefit an overconfident network, as the true uncertainty does not take into account the actual realized data. Generally, we would consider the NII vMF to have yielded the most appropriate uncertainty estimates in this experiment due to its symmetric properties over the Gaussian. Moreover, for its ability to detect an extra quadrant over the vMF when $N = 50$ and slightly more clearly detecting the certainty in origo for $N = 250$.

6.4 Regression with Carbon Nanotubes Data

Carbon Nanotubes Data test set accuracy in R^2			
Epochs	Gaussian	vMF	NII vMF
10	0.7483	0.8791	NA
30	0.9724	0.9592	0.9693
50	0.9906	0.9735	0.9696
Hyperparameters			
Prior	$\mathcal{N}(0, e^{-0.1})$	HU	HU
Learning rate	0.0007	0.08	0.11
Hidden Layers	(5, 5)	(5, 5)	(5, 5)

Table 6.3: Hyperparameters and results for the Carbon Regression experiment.

For this task, the networks are instructed to predict the calculated atomic coordinate of carbon atoms. This is done with UCI (2018) from ACI et al. (2017) and Acı and Avcı (2016). The predictor variables are real numbers, chiral indices, and initial atomic coordinates. The network's task is to predict the real number w-axis of the atomic coordinate calculated by CASTEP from ACI et al. (2017) and Acı and Avcı (2016). The predictor variables are standardized in the data pre-processing for this test. Standardized here means transformed to the standard normal distribution. The target variable is a real number from zero to one. The architecture we used for this experiment was four layers. We used five nodes on both of the two hidden layers.

There seems to be a somewhat inverse relationship to the results in Section 6.2. The vMF trains faster initially before the Gaussian somewhat outperforms the vMF and NIIvMF. Figure 6.5 might explain why this is the case. Here, we see that the loss descent for the vMF and NIIvMF are much lower, to begin with, but seem to struggle to converge. The NIIvMF appears to struggle in particular. On the other hand, the Gaussian shows optimal loss convergence. It quickly reaches a low loss and largely stays there. The Gaussian exhibits favorable behavior regarding the norms of the $\mathbf{W}_j$. It appears to slowly and monotonically converge to 1. Therefore, any gradient issues should not be expected even for larger models or models trained for longer. The "Ghost Mu"s in the vMF are severe. Although somewhat less so than in the phoneme classification. The Gaussian may disproportionately benefit from the standardized data on this task. Perhaps that is part of why both its loss and norm are favorable.

We found in the earlier experiments that setting a prior on the Gaussian that had slightly lower variance than one improved performance. Perhaps this slightly stronger regularization helps the Gaussian stay closer to norm one in its weights $\mathbf{W}_j$.

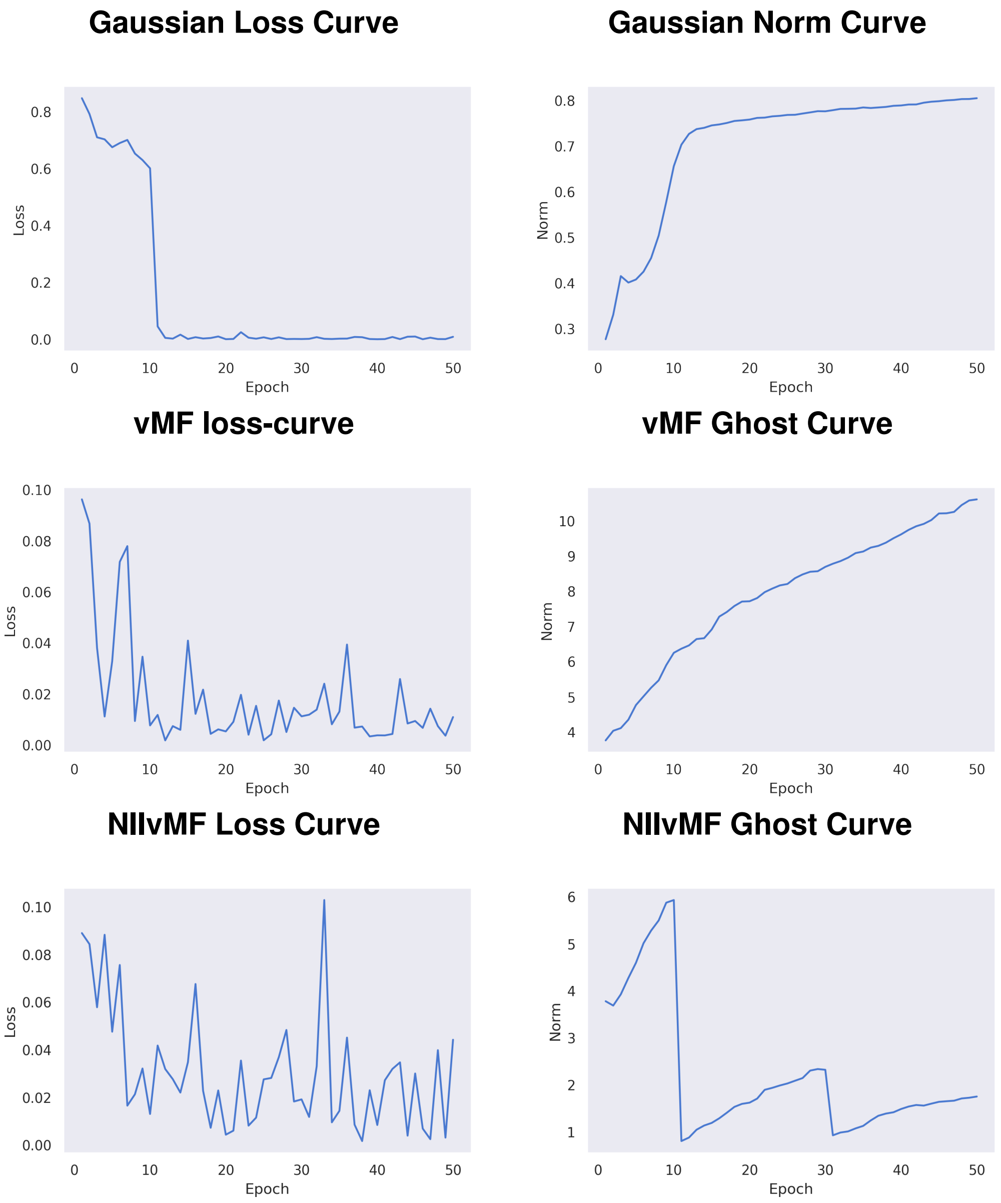


Figure 6.5: Loss and Norm Curves for Carbon Regression.
The left side plots plot the loss, i.e. the ELBO for the networks, per epoch. The plots in the right plot the norm of the registered parameters per epoch. For the vMF and NIIvMF, the registered parameters are not the actual parameters. Instead, the plots provide a look into what PyTorch does under the hood with them.

6.5 Vision

Modified National Institute of Standards and Technology (MNIST) dataset (Lecun et al., 2000) is a database of images of handwritten digits. Each image is a 28x28 pixel bounding box and anti-aliased compression of black and white images. The task for the networks will be to predict what digit the human attempted to write. It became apparent early on that test prediction was easy while computationally demanding. Therefore we sampled ten percent of the original training data for them to train on. We used the entire test set for testing. The same ten-network realization ensemble testing regimen is used as before. The architecture we used for this experiment was four layers. We used five nodes on both of the two hidden layers.

Furthermore, we encountered issues for the vMF as described in Section 5.2, which become worse as the dimensionality of each vMF instance increases. The dimension of the first layer vMFs, even with the nodewise architecture, was too large. It had to be 28x28=784. Therefore, we let the first layer be frequentist to evaluate both networks. Frequentist meaning that the parameters in this layer are optimized with maximum likelihood, using the same likelihood function as the later Bayesian layers. The Bayesian interpretation of this is that the "true" parameters, $\mathbf{W}_j$ and $\mathbf{B}_j$, of the first layers, are not random variables but vectors of real numbers. Parameters of the layer coming afterward are then random variables again. Although this does not break the theory, it is not supported either. We are doing this because our limited computational recourses force us to.

6.5.1 Vision Classification with MNIST

MNIST test set accuracy			
Trained on a subsample of 10% of the training data			
Epochs	Gaussian	vMF	NII vMF
10	8625/10000	7395/10000	NA
30	9023/10000	8473/10000	8882/10000
50	9083/10000	8630/10000	9129/10000
Hyperparameters			
Prior	$\mathcal{N}(0, e^{-0.1})$	HU	HU
Learning rate	0.0007	0.08	0.10
Hidden Layers	(10, 10)	(10, 10)	(10, 10)

Table 6.4: MNIST Hyperparameters and Results.

Table 6.4 shows that the Gaussian exhibits superior performance on 10 and 30 epochs. It appears to train initially faster than the vMF and NIIvMF. That is strange since the vMF and NIIvMF have vastly larger learning rates than the Gaussian, 0.08 vs. 0.0007. The vMF is a consistent underperformer in this test. However, it does improve with each epoch. The NIIvMF exerts essentially the same performance as the Gaussian. The difference we consider significant is that the NIIvMF and Gaussian outperform the vMF.

The plots for the loss and average norm of the layerwise weights $\mathbf{W}_j$ are included in Figure 6.6. The results indicate that the "Ghost Mu" problem is severe for this setting. The vMF norm curve grows steadily from 42 to 62. That is an already high start. The Gaussian norm curve is also relatively large. That could indicate that this is a setting where the NIIvMF could outperform its competition for deeper nets trained for more epochs. The NIIvMFs gradient and "Ghost Mu"s will significantly differ from the others in that setting. However, that would only be the case for deeper networks regarding the Gaussian since a four-layer network with norm 5.5 is not problematic computationally. The large average $\mathbf{W}_j$ norm of the vMF may explain why it underperforms the NIIvMF and Gaussian.

The plots for the loss show that the Gaussian has a much smoother descent than the vMF and NIIvMF. In this case, the NIIvMF becomes smoother than the vMF further in training. However, earlier results indicate that a smooth loss convergence does not necessarily indicate better test performance.

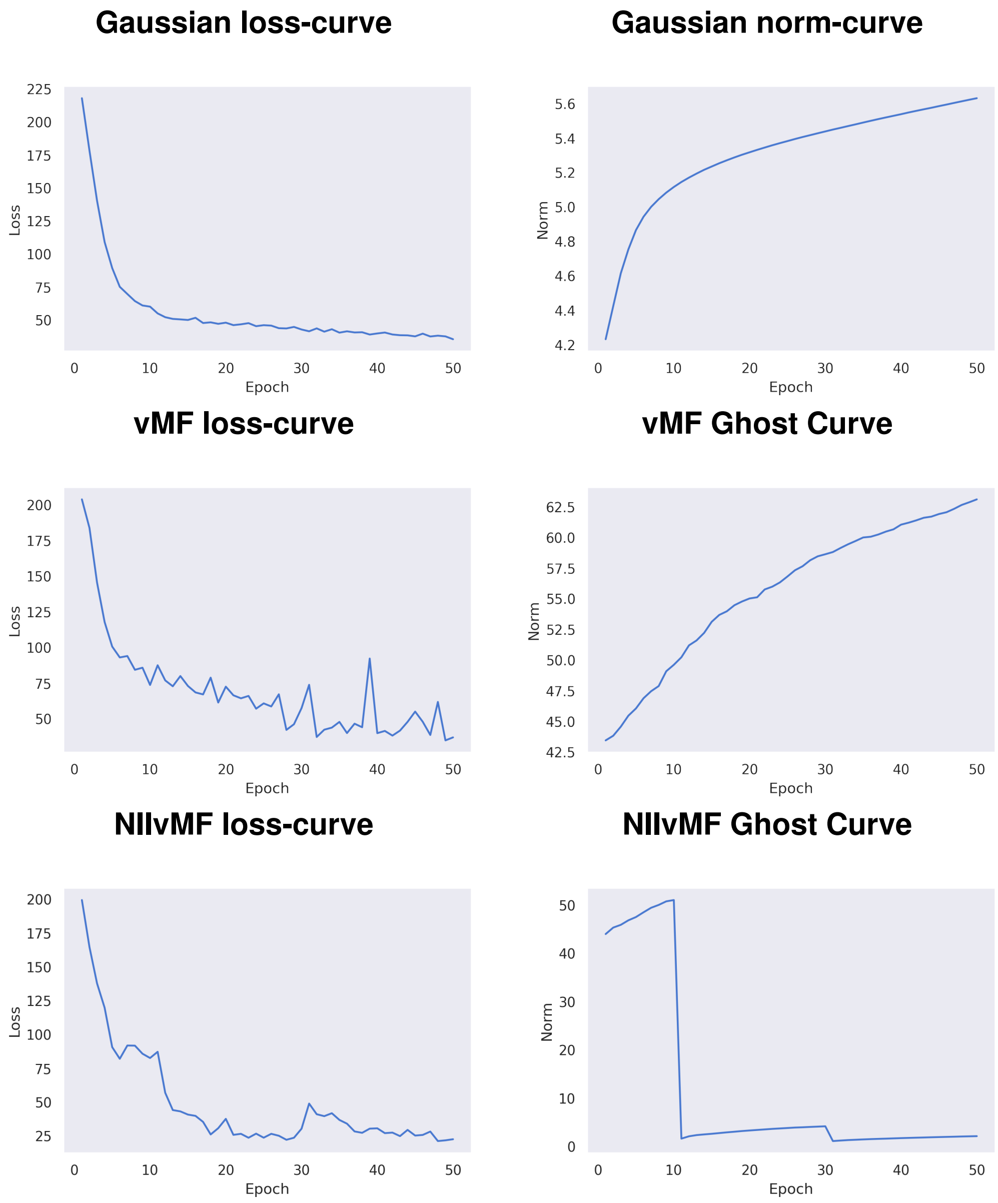


Figure 6.6: Loss and Norm Curves for MNIST Vision Classification.
The left side plots plot the loss, i.e. the ELBO for the networks, per epoch. The plots in the right plot the norm of the registered parameters per epoch. For the vMF and NIIvMF, the registered parameters are not the actual parameters. Instead, the plots provide a look into what PyTorch does under the hood with them.

6.5.2 Vision Classification with EMNIST

EMNIST test set accuracy			
Trained on a subsample of 10% of the training data			
Epochs	Gaussian	vMF	NII vMF
10	8748/10000	6994/10000	NA
30	9097/10000	8576/10000	9063/10000
50	9172/10000	8742/10000	9197/10000
Hyperparameters			
Prior	$\mathcal{N}(0, e^{-0.1})$	HU	HU
Learning rate	0.0007	0.08	0.1
Hidden Layers	(10, 10)	(10, 10)	(10, 10)

Table 6.5: EMNIST Hyperparameters and Results.

We test the efficacy of the networks, with the exact same setup as in Subsection 6.5.1. The only difference is that we have replaced the MNIST dataset with the EMNIST dataset (Cohen et al., 2017). EMNIST stands for Extended Modified National Institute of Standards and Technology. Here, data has been pulled from the same large source in NIST. The code used to process the images in the older MNSIT into 28 by 28 grayscale has been lost. EMNIST is a newer dataset that does this and documents how. That means the EMNIST is a dataset very close to MNIST, but not quite. We can view this data as new data gathered in the exact same manner, or a new random sample from the same source. Therefore, this data should essentially be an attempted replication of the MNIST experiment in Subsection 6.5.1, since we have kept all other things equal.

The results here, shown in Table 6.5 and Figure 6.7 generally replicate the results from Subsection 6.5.1. We can therefore be slightly more confident in our results. Summarized, the NIIvMF and Gaussian largely exhibit similar performance, while the vMF lags slightly behind.

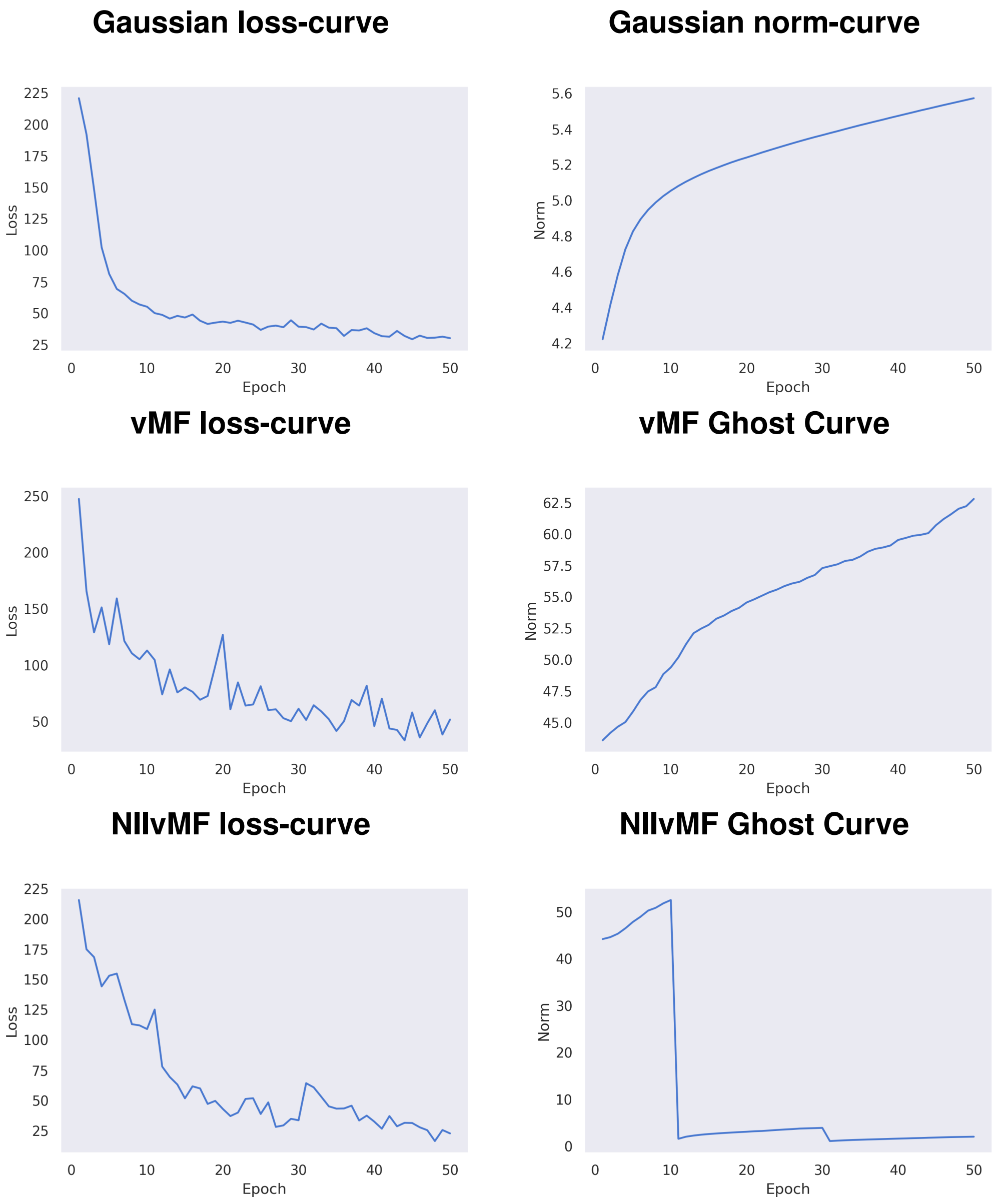


Figure 6.7: Loss and Norm Curves for MNIST Vision Classification.
The left side plots plot the loss, i.e. the ELBO for the networks, per epoch. The plots in the right plot the norm of the registered parameters per epoch. For the vMF and NIIvMF, the registered parameters are not the actual parameters used when optimizing, they are projected first.

Chapter 7

Discussion

In this chapter, we further discuss the results from Chapter 6. We set them in light of the theory from earlier, and what they could mean for future work.

7.1 Summary and Significance of Results

7.1.1 Validity of results

The experiments we conducted were on networks that are rather small by modern standards. Two hidden layers with five or ten nodes each are small models. Furthermore, we could only test for one seed per experiment. Both of these choices were made since the von Mises-Fisher distribution is computationally intensive for higher concentration parameters. Together with the fact that it performs well when initialized for higher concentrations, made deep networks and multiple seeds testing out of scope for the recourses we were allocated.

However, the EMNIST experiment, which is deliberately a closely related dataset to MNIST, both provided similar results. The test set accuracy in Table 6.4 and Table 6.5 were similar, and the plots for the loss and norms in Figure 6.6 and Figure 6.7 were also similar. Therefore, we could speculate that repeated testing may yield results that do not vary as much as feared. Thus, we can draw somewhat stronger conclusions than we otherwise could. However, neural networks are in general sensitive to initialization and hyperparameters. Furthermore, their performance is known to be variable, so we should still be careful with making too strong conclusions.

7.1.2 Summary of Results

The results were that the NII vMF performed on par with the Gaussian. It marginally outperformed the Gaussian in the uncertainty estimation in Section 6.3, while itself being outperformed in the regression task from Section 6.4. In the phoneme and vision classification tasks from Section 6.5 and Section 6.2 the Gaussian and NII vMF performed similarly. The vMF was in all tasks marginally outperformed by the NII vMF, except in regression.

7.2 Norm Results

The NIIvMF consistently outperformed the vMF. The NII vMF, also in general, was on par with the performance of the Gaussian. Informatively, the case where the Gaussian outperformed the vMF was in the case of carbon regression 6.4. That is informative because this was the singular case where the average norm of the layerwise weights $\mathbf{W}_j$ of the Gaussian was close to 1. Therefore, our theory would indicate the Hyperspherical Uniform prior to not perform better than the Gaussian in that specific instance if the norm plots were provided before the results. Our theory would predict the gradient to be within bounds, and hence the gradient's restrictive properties would be superfluous.

The parameters were largely on the same scale naturally for the Gaussian there, which is beneficial for Adam. Although, it could, of course, be the case that the norm of $\mathbf{W}_j$ varies greatly between each layer while only averaging close to 1. However, during our experiments, we measured the norm of each layer in the regression task. The measurement of the norm of the first layer is even included as one of the commits. There we found, that even for each j, $||\mathbf{W}_j|| \approx 1$. This supports our hypothesis that maintaining $||\mathbf{W}_j|| \approx 1$ is beneficial. Furthermore, that indicates that our hypothesis in Subsections 4.3.1 and 4.3.2 may have been correct. They would not predict the superior performance of the HU in cases where the Gaussian naturally trained parameters $||\mathbf{W}_j|| \approx 1 \ \forall j$.

Therefore, we could speculate that NII would eventually become superfluous for much higher epochs. We speculate that this means the optimization inside PyTorch might understand that the "Ghost Mu"s were supposed to live on the unit hypersphere. That observation congrues with the NIIvMF's superior test performance, although we refrain from concluding. These results would have to be replicated across several seeds before one should conclude.

7.2.1 Replacing Batch Normalization

As mentioned in subsubsection 4.3.2.1, the replacement of Batch Normalization by the vMF will not yield large advantages for smaller networks. As we have only utilized two hidden layers, we did not employ Batch Normalization, since that is only needed for deeper networks. Therefore, this advantage of the vMF and NII vMF has not been manifested. We can, therefore, speculate that the HU prior would exert greater benefits for deeper networks.

7.2.2 Efficacy of Normalized Initialization Inheritance

The Normalized Initialization Inheritance solution appears from the results to improve the performance of the networks. The NII vMF almost consistently outperforms its vMF counterpart. Therefore, we could speculate that large parts of the suspected issues arising from the "Ghost Mu" problem have been resolved by this simple approach. The issues were not expected to be extensive, to begin with, so we find that these results hint that it was mostly resolved.

7.3 Impact of non-resolved limitations

Although the negative effects of the "Ghost Mu" may have been resolved, other issues are not. The isotropic variance of the von Mises-Fisher distribution and its computational intensity for higher concentration parameters has yet to be resolved.

7.3.1 Von Mises-Fisher Computational Demand

As mentioned, the NII vMF architecture generally performs on par with the Gaussian. It outperforms it on uncertainty estimation but is itself outperformed on the regression task. It generally performs optimally for high concentration parameters. This is suboptimal regarding the computational intensity of its variational distribution for these high-concentration parameter. This behavior has dictated why we did not test networks with wider or more hidden layers due to the computational expense of the NII vMF and vMF architectures for those scenarios. That is also why we did not test for higher epoch situations either.

7.3.2 Isotropic Covariance of the von Mises Fisher

It is hard to evaluate how much of an effect this has on performance since we have yet to compare the von Mises-Fisher to another hyperspherical distribution with non-isotropic covariance. However, considering the general on-par performance with the Gaussian, we could speculate that the other positive attributes of the NII vMF outweigh the impact of this limitation.

7.4 Spherical Learning for Deep Learning Interpretability

Bayesian Neural Networks with Gaussian prior have already replaced the perhaps heuristic Dropout method (Gal and Ghahramani, 2015). With the Hyperspherical prior Batch Normalization is replaced, see Section 4.3.2. As Batch Normalization has no motivation from the Bayesian perspective, we have taken one more step into making our models better fit the Bayesian framework. Therefore, By doing so, we may assist in better understanding how parameters are optimized since one layer of heuristic is eliminated. Our implementation of spherical learning, with the Hyperspherical Uniform prior, works generally on par with the conventional Gaussian prior solution, even with remaining limitations. A better understanding of how parameters are optimized and what they do can make one tiny step toward a better understanding of how deep models work. We have made a slightly more interpretable model that could be used without sacrificing performance. That dramatically increases the chance that it will be helpful for further research or application.

7.5 Suggestions for future work

In general, we believe the Hyperspherical Uniform prior has shown desirable performance, with its advantages discussed earlier. Therefore we suggest future work also utilize this prior. For our von Mises-Fisher variational distribution, we could suggest alternatives here.

7.5.1 Projected Normal Distribution

Another candidate for variational distribution that has the unit sphere as its codomain is the projected normal distribution. However, this distribution has been used successfully for directional inference (Hernandez-Stumpfhauser et al., 2017), not for deep learning. This distribution is a multivariate Gaussian distribution that is projected. For our purposes, it would be along the unit sphere. That leaves us with both the native norm one property that was desirable in the vMF.

Additionally, it allows separate variances along each of its dimensions. Hypothetically, this distribution should also more closely inherit the Gaussian's $\mathcal{L}^2$ regularization properties. However, the important part is that this distribution supports a separate variance along each dimension.

In theory, then, this would be our ideal VD. However, its pdf is difficult for higher dimensions. A trigonometric reparametrization could be made. Wang and Gelfand (2013) and Wang et al. (2014) demonstrated how to do so, $\mathcal{PN}_2(\mu, \Sigma)$ with $\Sigma \neq I$. Although for the special case of circular data $(k = 2)$. Let us do a trigonometric transformation of a Projected Gaussian Normal, $\boldsymbol{\Psi}$, by $\boldsymbol{\Psi} = (\cos\Theta, \sin\Theta)^T$. The expression for the density of Θ is:

$$p(\theta \mid \mu, \Sigma) = \left(\frac{1}{2\pi A(\theta)}\right) |\Sigma|^{-\frac{1}{2}} \exp(C) \left\{ 1 + \frac{B(\theta)}{\sqrt{A(\theta)}} \frac{\Phi\left(\frac{B(\theta)}{\sqrt{A(\theta)}}\right)}{\varphi\left(\frac{B(\theta)}{\sqrt{A(\theta)}}\right)} \right\} I_{[0,2\pi)}(\theta)$$

Here $u^T = (\cos\theta, \sin\theta)$, $A(\theta) = u^T \Sigma^{-1} u$, $B(\theta) = u^T \Sigma^{-1} \mu$ and $C = -\frac{1}{2}\mu^T \Sigma^{-1} \mu$. $I_{(0,2\pi]}(\cdot)$ is an indicator function, and $\Phi(\cdot)$, $\varphi(\cdot)$ are the standard normal distribution and density functions, respectively. The expression above is reminiscent of one in Pukkila and Rao (1988). This expression of the pdf much more readily generalizes to higher dimensions.

The trigonometric reparametrization resolves the problem of the pdf itself being unusable. However, as mentioned in Section 5.1, any gradient registered parameter will be optimized along $\mathbb{R}$ in its entirety. In Hernandez-Stumpfhauser et al. (2017), the sampling was conducted by first sampling from a Gaussian and then projecting it afterward. After conversations with the authors, we found no way to conduct this sampling without the projection step. That leaves us, in principle, with the same "Ghost Mu" problem as before. However, the straightforward Normalized Initialization Inheritance proved to improve results systematically, so the effect of the "Ghost Mu" is hard to determine. We know this problem can get worse the larger the original vector that has to be projected is. As we expect the norm of μ to increase the same way we saw them do for the vMF, this problem would keep getting worse on its own. However, the NII method proposed in Subsection 5.1.2 could help ameliorate that issue. That method consistently improved the performance of the vMF in our testing. Hence, we suggest implementing this VD with NII for further research. To reparametrize this distribution one could construct the sampler to be from a Gaussian and then project. While for the pdf, use the trigonometric transformation above.

Chapter 8

Conclusion

Our results indicate that the Hyperspherical Uniform (HU) prior Bayesian neural network (BNN), with von Mises-Fisher (vMF) variational distribution (VD), generally performs on par with the conventional Gaussian one. The novel prior provided three main weaknesses, computational intensity for lower variance, "Ghost Mu," and isotropic covariance. We proposed Normalized Initialization Inheritance (NII) to address the "Ghost Mu" issue. Testing this solution indicates that it ameliorates the issue. As the Hyperspherical Unform prior replaces Batch Normalization, it is more coherent with the Bayesian framework. Therefore, our prior can help deeper networks that need Batch Normalization be marginally more interpretable while reducing computational demand during the backward pass.

The Projected Normal (PN) VD redresses the current VD's isotropic covariance and computational intensity issues. While the "Ghost Mu" issue would remain, NII could also be applied there. We suggest investigating the HU prior with PN VD. Our prior and VD combination already performs well despite the remaining limitations. We expect the gradient advantages to increase for deeper nets. For those two compounding reasons, we speculate that the HU prior with PN VD may yield significantly better performance than the Gaussian BNN for deep networks.

Bibliography

Marvin Zhang. Course cs 282a, designing, visualizing and understanding deep neural network, 2022. URL https://cs182sp22.github.io/.

Håkon O. Trovik Leif-Martin Sunde, Khachatur Mirijanyan. Cs282a final projects, 2022. URL https://github.com/Haakooto/CS282_final_project.

Aidan R O'Brien, Gaetan Burgio, and Denis C Bauer. Domain-specific introduction to machine learning terminology, pitfalls and opportunities in CRISPR-based gene editing. *Briefings in Bioinformatics*, 22(1):308–314, February 2020. doi: 10.1093/bib/bbz145. URL https://doi.org/10.1093/bib/bbz145.

Trevor Hastie, Jerome Friedman, and Robert Tibshirani. *The Elements of Statistical Learning*. Springer New York, 2001. doi: 10.1007/978-0-387-21606-5. URL https://doi.org/10.1007/978-0-387-21606-5.

Brian Duignan. Occam's razor, 2023. URL https://www.britannica.com/topic/Occams-razor.

Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off, jul 2019. URL https://doi.org/10.1073/pnas.1903070116.

Terrence J. Sejnowski. The unreasonable effectiveness of deep learning in artificial intelligence. *Proceedings of the National Academy of Sciences*, 117(48):30033–30038, 2020. doi: 10.1073/pnas.1907373117. URL https://doi.org/10.1073/pnas.1907373117.

Jaime Sevilla, Lennart Heim, Anson Ho, Tamay Besiroglu, Marius Hobbhahn, and Pablo Villalobos. Compute trends across three eras of machine learning. In *2022 International Joint Conference on Neural Networks (IJCNN)*. IEEE, July 2022. doi: 10.1109/ijcnn55064.2022.9891914. URL https://doi.org/10.1109/ijcnn55064.2022.9891914.

Mikhail Belkin, Daniel Hsu, and Ji Xu. Two models of double descent for weak features. *SIAM Journal on Mathematics of Data Science*, 2(4):1167–1180, 2020. doi: 10.1137/20M1336072. URL https://doi.org/10.1137/20M1336072.

Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt, 2019.

Song Mei and Andrea Montanari. The generalization error of random features regression: Precise asymptotics and the double descent curve. *Communications on Pure and Applied Mathematics*, 75(4):667–766, June 2021. doi: 10.1002/cpa.22008. URL https://doi.org/10.1002/cpa.22008.

Y. Le Cun and Françoise Fogelman-Soulié. Modèles connexionnistes de l'apprentissage. *Intellectica. Revue de l'Association pour la Recherche Cognitive*, 2(1):114–143, 1987. doi: 10.3406/intel.1987.1804. URL https://doi.org/10.3406/intel.1987.1804.

Leo Breiman. Statistical Modeling: The Two Cultures (with comments and a rejoinder by the author). *Statistical Science*, 16(3):199 – 231, 2001. doi: 10.1214/ss/1009213726. URL https://doi.org/10.1214/ss/1009213726.

Eric Nalisnick, Padhraic Smyth, and Dustin Tran. A brief tour of deep learning from a statistical perspective. *Annual Review of Statistics and Its Application*, 10(1):219–246, March 2023. doi: 10.1146/annurev-statistics-032921-013738. URL https://doi.org/10.1146/annurev-statistics-032921-013738.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

Wei Wu, Qinwei Fan, Jacek M. Zurada, Jian Wang, Dakun Yang, and Yan Liu. Batch gradient method with smoothing regularization for training of feedforward neural networks. *Neural Networks*, 50:72–78, February 2014. doi: 10.1016/j.neunet.2013.11.006. URL https://doi.org/10.1016/j.neunet.2013.11.006.

Donald W. Marquardt and Ronald D. Snee. Ridge regression in practice. *The American Statistician*, 29(1):3–20, feb 1975. doi: 10.1080/00031305.1975.10479105. URL https://www.researchgate.net/publication/243779628_Ridge_Regression_in_Practice.

Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning, 2015. URL https://arxiv.org/abs/1506.02142.

M.A.T. Figueiredo. Adaptive sparseness for supervised learning. *IEEE Transac-*

tions on Pattern Analysis and Machine Intelligence, 25(9):1150–1159, September 2003. doi: 10.1109/tpami.2003.1227989. URL https://doi.org/10.1109/tpami.2003.1227989.

Peter J. Huber. Robust statistics. In *International Encyclopedia of Statistical Science*, pages 1248–1251. Springer Berlin Heidelberg, 2011. doi: 10.1007/978-3-642-04898-2_594. URL https://doi.org/10.1007/978-3-642-04898-2_594.

Peter J. Huber. John w. tukey's contributions to robust statistics. *The Annals of Statistics*, 30(6):1640–1648, 2002. ISSN 00905364. URL http://www.jstor.org/stable/1558734.

Kumar Shridhar, Felix Laumann, and Marcus Liwicki. A comprehensive guide to bayesian convolutional neural network with variational inference, 2019. URL https://arxiv.org/abs/1901.02731.

Ethan Goan and Clinton Fookes. Bayesian neural networks: An introduction and survey. In *Case Studies in Applied Bayesian Data Science*, pages 45–87. Springer International Publishing, 2020. doi: 10.1007/978-3-030-42553-1_3. URL https://doi.org/10.1007/978-3-030-42553-1_3.

Michael W. Dusenberry, Ghassen Jerfel, Yeming Wen, Yi-An Ma, Jasper Snoek, Katherine Heller, Balaji Lakshminarayanan, and Dustin Tran. Efficient and scalable bayesian neural nets with rank-1 factors, 2020. URL https://arxiv.org/abs/2005.07186.

Ruqi Zhang, Chunyuan Li, Jianyi Zhang, Changyou Chen, and Andrew Gordon Wilson. Cyclical stochastic gradient mcmc for bayesian deep learning, 2019. URL https://arxiv.org/abs/1902.03932.

Alexander Immer, Maciej Korzepa, and Matthias Bauer. Improving predictions of bayesian neural nets via local linearization, 2020. URL https://arxiv.org/abs/2008.08400.

Christos Louizos and Max Welling. Multiplicative normalizing flows for variational bayesian neural networks, 2017. URL https://arxiv.org/abs/1703.01961.

Jose Miguel Hernandez-Lobato and Ryan Adams. Probabilistic backpropagation for scalable learning of bayesian neural networks. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1861–1869, Lille, France, 07–09 Jul 2015. PMLR. URL https://proceedings.mlr.press/v37/hernandez-lobatoc15.html.

Radford M. Neal. *Bayesian Learning for Neural Networks*. Springer New York, 1996. doi: 10.1007/978-1-4612-0745-0. URL https://doi.org/10.1007/978-1-4612-0745-0.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. URL https://arxiv.org/abs/1412.6980.

Zijun Zhang, Lin Ma, Zongpeng Li, and Chuan Wu. Normalized direction-preserving adam, 2017. URL https://arxiv.org/abs/1709.04546.

Léon Bottou, Frank E. Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning, 2016. URL https://doi.org/10.48550/arXiv.1606.04838.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, October 1986. doi: 10.1038/323533a0. URL https://doi.org/10.1038/323533a0.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019. URL https://arxiv.org/abs/1912.01703.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3):400–407, September 1951. doi: 10.1214/aoms/1177729586. URL https://doi.org/10.1214/aoms/1177729586.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015. URL https://arxiv.org/abs/1502.03167.

Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors, 2012. URL https://doi.org/10.48550/arXiv.1207.0580.

Grant Sanderson. The basics of neural networks, and the math behind how they learn, 2015. URL https://www.3blue1brown.com/topics/neural-networks#neural-networks.

Adam Paszke et al. Automatic differentiation in pytorch. *NIPS 2017 Workshop Autodiff Submission*, March 2017. URL https://openreview.net/forum?id=BJJsrmfCZ.

C. F. Curtiss and J. O. Hirschfelder. Integration of stiff equations. *Proceedings of the*

National Academy of Sciences, 38(3):235–243, March 1952. doi: 10.1073/pnas.38.3.235. URL https://doi.org/10.1073/pnas.38.3.235.

Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009. ISBN 1441412697.

John Hunt. *Advanced Guide to Python 3 Programming*. Springer International Publishing, 2019. doi: 10.1007/978-3-030-25943-3. URL https://doi.org/10.1007/978-3-030-25943-3.

Horace He. The state of machine learning frameworks in 2019. *The Gradient*, 2019.

OpenAI. Gpt-4 technical report, 2023. URL https://arxiv.org/abs/2303.08774.

T.L. Lindstrøm. *Spaces: An Introduction to Real Analysis*. Pure and Applied Undergraduate Texts. American Mathematical Society, 2017. ISBN 9781470440626. URL https://books.google.no/books?id=mL9ADwAAQBAJ.

Andrew Gelman, John B. Carlin, Hal S. Stern, David B. Dunson, Aki Vehtari, and Donald B. Rubin. *Bayesian Data Analysis*. Chapman and Hall/CRC, November 2013. doi: 10.1201/b16018. URL https://doi.org/10.1201/b16018.

W. K. Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, April 1970. doi: 10.1093/biomet/57.1.97. URL https://doi.org/10.1093/biomet/57.1.97.

Christian P. Robert. The metropolis-hastings algorithm, 2015. URL https://arxiv.org/abs/1504.01896.

Christophe Dupuy and Francis Bach. Online but accurate inference for latent variable models with local gibbs sampling. *Journal of Machine Learning Research*, 2016. doi: 10.48550/ARXIV.1603.02644. URL https://arxiv.org/abs/1603.02644.

Diederik P. Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick, 2015. URL https://arxiv.org/abs/1506.02557.

David M. Blei, Alp Kucukelbir, and Jon D. McAuliffe. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112(518):859–877, April 2017. doi: 10.1080/01621459.2017.1285773. URL https://doi.org/10.1080/01621459.2017.1285773.

Aliaksandr Hubin and Geir Storvik. Combining model and parameter uncertainty in bayesian neural networks, 2019. URL https://arxiv.org/abs/1903.07594.

Lorena A. Barba. Terminologies for reproducible research, 2018. URL https://arxiv.org/abs/1802.03311.

William T. Richards. A definition of science. *Journal of Chemical Education*, 5 (7):874, July 1928. doi: 10.1021/ed005p874. URL https://doi.org/10.1021/ed005p874.

Oxford Reference. A definition of science, 2019. URL https://www.oxfordreference.com/page/scienceandtech/science-and-technology.

Jocelyn Kaiser. Potti found guilty of research misconduct. *Science*, November 2015. doi: 10.1126/science.aad7410. URL https://doi.org/10.1126/science.aad7410.

Malcolm R Macleod, Susan Michie, Ian Roberts, Ulrich Dirnagl, Iain Chalmers, John P A Ioannidis, Rustam Al-Shahi Salman, An-Wen Chan, and Paul Glasziou. Biomedical research: increasing value, reducing waste. *The Lancet*, 383(9912): 101–104, January 2014. doi: 10.1016/s0140-6736(13)62329-6. URL https://doi.org/10.1016/s0140-6736(13)62329-6.

Iain Chalmers and Paul Glasziou. Avoidable waste in the production and reporting of research evidence. *The Lancet*, 374(9683):86–89, July 2009. doi: 10.1016/s0140-6736(09)60329-9. URL https://doi.org/10.1016/s0140-6736(09)60329-9.

Git, 2023. URL https://git-scm.com/doc.

Scott Chacon and Ben Straub. *Pro git*. Apress, 2014. URL https://github.com/progit/progit2/releases/download/2.1.392/progit.pdf.

Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Perez, Brian E. Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica B. Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Damian Avila, Safia Abdalla, Carol Willing, and Jupyter Development Team. Jupyter notebooks - a publishing format for reproducible computational workflows. In *International Conference on Electronic Publishing*, 2016. doi: https://doi.org/10.3233/978-1-61499-649-1-87.

Brian E. Granger and Fernando Perez. Jupyter: Thinking and storytelling with code and data. *Computing in Science &amp; Engineering*, 23(2):7–14, March 2021. doi: 10.1109/mcse.2021.3059263. URL https://doi.org/10.1109/mcse.2021.3059263.

Anaconda. Anaconda software distribution, 2020. URL https://docs.anaconda.com/.

Anaconda software distribution, 2020. URL https://mamba.readthedocs.io/en/latest/index.html.

Steve McConnell. *Code Complete, Second Edition*. Microsoft Press, USA, 2004. ISBN 0735619670. URL https://dl.acm.org/doi/10.5555/1096143.

Project Jupyter, Matthias Bussonnier, Jessica Forde, Jeremy Freeman, Brian Granger, Tim Head, Chris Holdgraf, Kyle Kelley, Gladys Nalvarte, Andrew Osheroff, M Pacer, Yuvi Panda, Fernando Perez, Benjamin Ragan-Kelley, and Carol Willing. Binder 2.0 - reproducible, interactive, sharable environments for science at scale. In *Proceedings of the Python in Science Conference*. SciPy, 2018. doi: 10.25080/majora-4af1f417-011. URL https://doi.org/10.25080/majora-4af1f417-011.

Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 06(02):107–116, April 1998. doi: 10.1142/s0218488598000094. URL https://doi.org/10.1142/s0218488598000094.

Yuxin Wu and Kaiming He. Group normalization, 2018. URL https://arxiv.org/abs/1803.08494.

Devansh Arpit, Yingbo Zhou, Bhargava U. Kota, and Venu Govindaraju. Normalization propagation: A parametric technique for removing internal covariate shift in deep networks, 2016. URL https://arxiv.org/abs/1603.01431.

Xiang Li, Shuo Chen, Xiaolin Hu, and Jian Yang. Understanding the disharmony between dropout and batch normalization by variance shift, 2018. URL https://arxiv.org/abs/1801.05134.

Nicholas Polson and Veronika Rockova. Posterior concentration for sparse deep learning, 2018. URL https://arxiv.org/abs/1803.09138.

Tim Salimans and Diederik P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks, 2016. URL https://arxiv.org/abs/1602.07868.

Changyong Oh, Kamil Adamczewski, and Mijung Park. Radial and directional posteriors for bayesian neural networks, 2019. URL https://arxiv.org/abs/1902.02603.

Vincent Fortuin. Priors in bayesian deep learning: A review, 2021. URL https://arxiv.org/abs/2105.06868.

Meet P. Vadera, Adam D. Cobb, Brian Jalaian, and Benjamin M. Marlin. Impact of parameter sparsity on stochastic gradient mcmc methods for bayesian deep learning, 2022. URL https://arxiv.org/abs/2202.03770.

Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks, 2021. URL https://arxiv.org/abs/2102.00554.

Aditya Agrawal, 2023. URL https://www.adityaagrawal.net/blog/deep_learning/bprop_batch_norm.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2022. URL https://doi.org/10.48550/arXiv.1312.6114.

Tim R. Davidson, Luca Falorsi, Nicola De Cao, Thomas Kipf, and Jakub M. Tomczak. Hyperspherical variational auto-encoders. *arXiv*, 2018. doi: 10.48550/ARXIV.1804.00891. URL https://arxiv.org/abs/1804.00891.

Sebastian Farquhar, Michael Osborne, and Yarin Gal. Radial bayesian neural networks: robust variational inference in big models, 2019. URL https://doi.org/10.48550/arXiv.1907.00865.

Minyoung Kim. On pytorch implementation of density estimators for von mises-fisher and its mixture, 2021. URL https://arxiv.org/abs/2102.05340.

Nicola De Cao and Wilker Aziz. The power spherical distribution, 2020. URL https://arxiv.org/abs/2006.04437.

Trevor Hastie, Andreas Buja, and Robert Tibshirani. Penalized discriminant analysis. *The Annals of Statistics*, 23(1), February 1995. doi: 10.1214/aos/1176324456. URL https://doi.org/10.1214/aos/1176324456.
Datalink from UiO:
https://www.uio.no/studier/emner/matnat/math/STK2100/data/phoneme.data.

Machine Learning Repository UCI. Carbon nanotubes dataset, 2018.
https://archive.ics.uci.edu/ml/datasets/Carbon+Nanotubes.

Mehmet ACI, Mutlu AVCI, and Çiğdem ACI. Destek vektör regresyonu yöntemiyle karbon nanotüp benzetim süresinin kisaltilmasi. *Gazi Üniversitesi Mühendislik-Mimarlık Fakültesi Dergisi*, 32(3), September 2017. doi: 10.17341/gazimmfd.337642. URL https://doi.org/10.17341/gazimmfd.337642.

Mehmet Acı and Mutlu Avcı. Artificial neural network approach for atomic coordinate prediction of carbon nanotubes. *Applied Physics A*, 122(7), June 2016. doi: 10.1007/s00339-016-0153-1. URL https://doi.org/10.1007/s00339-016-0153-1.

Yann Lecun, Patrick Haffner, and Y. Bengio. Object recognition with gradient-based learning. 08 2000. doi: 10.1007/3-540-46805-6_19. URL https://doi.org/10.1007/3-540-46805-6_19.

Use torchvision.datasets.MNIST in Python to fetch dataset. This library is provided in the conda environment specified in the .yml file.

Gregory Cohen, Saeed Afshar, Jonathan Tapson, and André van Schaik. Emnist: an extension of mnist to handwritten letters. *arXiv preprint arXiv:1702.05373*, 2017. URL https://doi.org/10.48550/arXiv.1702.05373.

Use torchvision.datasets.MNIST in Python to fetch dataset. This library is provided in the conda environment specified in the .yml file.

Daniel Hernandez-Stumpfhauser, F. Jay Breidt, and Mark J. van der Woerd. The general projected normal distribution of arbitrary dimension: Modeling and bayesian inference. *Bayesian Analysis*, 12(1), March 2017. doi: 10.1214/15-ba989. URL https://doi.org/10.1214/15-ba989.

Fangpo Wang and Alan E. Gelfand. Directional data analysis under the general projected normal distribution. *Statistical Methodology*, 10(1):113–127, January 2013. doi: 10.1016/j.stamet.2012.07.005. URL https://doi.org/10.1016/j.stamet.2012.07.005.

Fangpo Wang, Alan Gelfand, and Giovanna Jona-Lasinio. Joint spatio-temporal analysis of a linear and a directional variable: space-time modeling of wave heights and wave directions in the adriatic sea. *Statistica Sinica*, 2014. doi: 10.5705/ss.2013.204w. URL https://doi.org/10.5705/ss.2013.204w.

Tarmo M. Pukkila and C. Radhakrishna Rao. Pattern recognition based on scale invariant discriminant functions. *Information Sciences*, 45(3):379–389, August 1988. doi: 10.1016/0020-0255(88)90012-6. URL https://doi.org/10.1016/0020-0255(88)90012-6.