École Polytechnique Fédérale de Lausanne

Disentangling the Gauss-Newton Method and Approximate Inference for Neural Networks

by Alexander Immer

Master's Thesis

Dr. Emtiyaz Khan*
Thesis Supervisor

Prof. Dr. Matthias Grossglauser†
Prof. Dr. Patrick Thiran†
Thesis Advisors

*RIKEN Center for Advanced Intelligence Project
Tokyo, Japan

and

†EPFL School of Computer and Communication Sciences
Lausanne, Switzerland

July 20, 2020

Abstract

Deep neural networks achieve state-of-the-art performance in many real-world machine learning problems and alleviate the need to design features by hand. However, their flexibility often comes at a cost. Neural network models are hard to interpret, often overconfident, and do not quantify how probable they are given a dataset. The Bayesian approach to infer neural networks is one way to tackle these issues. However, exact Bayesian inference for neural networks is intractable. Therefore, *Bayesian deep learning* combines approximate inference and optimization methods to design efficient methods that provide an approximate solution. Nonetheless, the combination of both methods is often not well understood.

In this thesis, we disentangle the generalized Gauss-Newton and approximate inference for Bayesian deep learning. The generalized Gauss-Newton method is an optimization method that is used in several popular Bayesian deep learning algorithms. In particular, algorithms that combine the Gauss-Newton method with the Laplace and Gaussian variational approximation have recently led to state-of-the-art results in Bayesian deep learning. While the Laplace and Gaussian variational approximation have been studied extensively, their interplay with the Gauss-Newton method remains unclear. For example, we know that both approximate inference methods compute a Gaussian approximation to the posterior. However, it is not clear how the Gauss-Newton method impacts the underlying probabilistic model or posterior approximation. Additionally, recent criticism of priors and posterior approximations in Bayesian deep learning further urges the need for a deeper understanding of practical algorithms.

The individual analysis of the Gauss-Newton method and Laplace and Gaussian variational approximations for neural networks provides both theoretical insight and new practical algorithms. We find that the Gauss-Newton method simplifies the underlying probabilistic model significantly. In particular, the combination of the Gauss-Newton method with approximate inference can be cast as inference in a linear or Gaussian process model. We find that the Gauss-Newton method turns the original model locally into a linear or Gaussian process model. The Laplace and Gaussian variational approximation can subsequently provide a posterior approximation to these simplified models. This new disentangled understanding of recent Bayesian deep learning algorithms also leads to new methods: first, the connection to Gaussian processes enables new function-space inference algorithms. Second, we present a marginal likelihood approximation of the underlying probabilistic model to tune neural network hyperparameters. Finally, the identified underlying models lead to different methods to compute predictive distributions. In fact, we find that these prediction methods for Bayesian neural networks often work better than the default choice and solve a common issue with the Laplace approximation.

Mathematical Notation and Abbreviations

Symbol	Explanation
$\mathbb{R}, \mathbb{R}_+$	set of real and positive real numbers
x	scalar variable
$\boldsymbol{v}$	vector variable with scalar entries v_i
$\boldsymbol{M}$	matrix variable with scalar entries $\boldsymbol{M}_{ij}$
T	tensor variable with scalar entries T_{ijk}
$f(\boldsymbol{x}; \boldsymbol{\theta})$	function mapping $\boldsymbol{x}$ to some output parameterized by $\boldsymbol{\theta}$, for example a neural network
$\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta})$	the gradient with individual entries $[\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta})]_i = \frac{\partial f(\boldsymbol{\theta})}{\partial \theta_i}$
$\nabla^2_{\boldsymbol{\theta}\boldsymbol{\theta}} f(\boldsymbol{\theta})$	the Hessian with entries $\left[\nabla^2_{\boldsymbol{\theta}\boldsymbol{\theta}} f(\boldsymbol{\theta})\right]_{ij} = \frac{\partial^2 f(\boldsymbol{\theta})}{\partial \theta_i \partial \theta_j}$
$\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}_*)$	gradient evaluated at $\boldsymbol{\theta} = \boldsymbol{\theta}_*$, same applies for Hessian
$\langle \cdot, \cdot \rangle$	scalar, vector, or Frobenius inner product depending on the context
$\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$	multivariate normal distribution with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$
$\mathcal{N}(\boldsymbol{\theta}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$	$\boldsymbol{\theta}$ is distributed according to $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$.

Abbreviation	Meaning
GGN, GN	(generalized) Gauss-Newton
GVA	Gaussian variational approximation
GP	Gaussian process
GLM	generalized linear model
GGPM	generalized Gaussian process model
BLR	Bayesian linear regression (model)
VOGGN	variational online generalized Gauss-Newton
OGGN	online generalized Gauss-Newton
LGVA	linearized Gaussian variational approximation

Contents

Abstract **2**

Mathematical Notation and Abbreviations **3**

1 Introduction **6**
1.1 Probabilistic Models and Inference 7
1.2 Bayesian Deep Learning 8
1.3 Outline of the Thesis 8

2 Background **9**
2.1 Probabilistic Neural Networks for Supervised Learning 10
2.2 Approximate Bayesian Inference 12
2.3 The Generalized Gauss-Newton Method 15

3 From Neural Network to Gaussian Process with Laplace and Gauss-Newton **18**
3.1 From Neural Network to Generalized Linear Model 19
3.2 From Generalized Linear Model to Bayesian Linear Regression 20
3.3 From Generalized Gaussian Process to Gaussian Process Regression 21
3.4 Computing the Laplace-GGN Posterior Approximation 23
3.5 Posterior Predictive Distributions 24
3.6 Marginal Likelihood Approximation 25
3.7 The Prior as Regularizer and Distribution 26

4 The Impact of the Generalized Gauss-Newton in Variational Inference **27**
4.1 Gaussian Natural Gradient Variational Inference 28
4.2 Variational Online Generalized Gauss-Newton 29
4.3 Linearized Gaussian Variational Inference 30
4.4 Deterministic Variational Online Gauss-Newton 30
4.5 Variational GGN Iterations as Exact Inference 31
4.6 Comparison and Posterior Predictive Computation 33

5 Experiments **34**
5.1 Model Selection Using Marginal Likelihood 35
5.2 Posterior Predictive Distributions 38

5.3 Function-Space Neural Network Inference for Explainability 41

6 Discussion and Future Directions **44**
6.1 Related Work 44
6.2 Conclusion 46
6.3 Future Work 48

Bibliography **49**

A Proof of GLM Log Likelihood Derivatives **53**

B Completing the Square **55**

C Natural Gradient Variational Inference **56**

Chapter 1

Introduction

The field of *machine learning* deals with algorithms that teach computers to make predictions or take actions in novel scenarios based on past experience. In the setting of *supervised learning*, the past experience consists of observed data points comprising inputs and labels. For example, the input could be an image of an object and the label its name. A machine learning algorithm can teach, or *train*, the computer to predict the label of unseen images using a *learned* model. In the *probabilistic machine learning* framework, we model uncertainties about our choice of model, i.e., we don't restrict ourselves to a single bet. Therefore, *inference* in this framework is about identifying a distribution over models that explain the past experience well and therefore generalize to future observations. Having a distribution instead of a single model enables uncertainty quantification and model comparison.

Apart from the data, a probabilistic machine learning algorithm comprises two key components: a probabilistic model and an inference algorithm. In this work, we focus on probabilistic models of neural networks and scalable inference algorithms for such models. We disentangle common inference algorithms that combine *approximate inference* with the *generalized Gauss-Newton* (GGN) optimization method and investigate how both affect the underlying probabilistic model. We show that, locally, these techniques simplify the probabilistic model drastically. Investigating these simplified underlying models helps to improve our understanding of Bayesian deep learning. We further exploit this understanding to enhance Bayesian deep learning. In fact, the findings presented here lead to more accurate predictions of neural network models inferred with approximate inference methods. The disentanglement of the Gauss-Newton method and approximate inference further enables novel ways to compute the posterior and posterior predictive of a neural network in the function-space and gives rise to a marginal likelihood approximation. Further, we identify a new variational inference algorithm and provide experimental support for our theoretical results and hypotheses.

1.1 Probabilistic Models and Inference

More formally, the combination of data and a corresponding probabilistic model of the data can be described as follows. In the supervised setting, we are given a dataset $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^N$ of N independent and identically distributed input $\boldsymbol{x}_i$ and label $\boldsymbol{y}_i$ pairs. For now, let both input and output be some abstract quantities. In the previous example, $\boldsymbol{x}_i$ corresponds to an image while $\boldsymbol{y}_i$ denotes the corresponding object name. A probabilistic model consists of *prior* and *likelihood*. In the case of probabilistic neural networks, we usually pose a *prior* over the parameters $\boldsymbol{\theta}$, i.e., $p(\boldsymbol{\theta})$. The *likelihood* of observing a label $\boldsymbol{y}$ for a given parameter $\boldsymbol{\theta}$ and input $\boldsymbol{x}$ can be written as $p(\boldsymbol{y}|\boldsymbol{\theta}, \boldsymbol{x})$. Since all data points are i.i.d., we write $p(\mathcal{D}|\boldsymbol{\theta})$ for the product of all likelihoods over the entire dataset.

Turning to the problem of learning, we deal with the *maximum a posteriori* (MAP) estimate and Bayesian inference. In Bayesian inference, we compute the *posterior distribution* $p(\boldsymbol{\theta}|\mathcal{D})$ over the parameter using Bayes rule

$$p(\boldsymbol{\theta}|\mathcal{D}) = \frac{p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{\int p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})d\boldsymbol{\theta}} = \frac{p(\mathcal{D}, \boldsymbol{\theta})}{p(\mathcal{D})}, \tag{1.1}$$

where the normalization constant in the denominator is called the *marginal likelihood*. In contrast to Bayesian inference, MAP estimation solely captures the mode, i.e., the most probable parameter, of the posterior. Ignoring the normalization constant, this can be achieved by maximizing the joint distribution:

$$\boldsymbol{\theta}_{\text{MAP}} = \arg\max_{\boldsymbol{\theta}} p(\mathcal{D}, \boldsymbol{\theta}). \tag{1.2}$$

MAP estimation is typically much easier and computationally convenient while Bayesian inference is in many cases intractable due to the integration. One could distinguish the two types of inference as follows: Bayesian inference is *learning by integration* while MAP estimation is *learning by optimization*. Computationally, optimization is much more convenient than integration.

Ignoring the computational burden, we would often prefer Bayesian inference since the posterior distribution captures more information than the MAP estimate. While the MAP estimate provides only *local* information at $\boldsymbol{\theta}_{\text{MAP}}$, the posterior distribution carries *global* information due to the integration over the parameter space captured in the marginal likelihood $p(\mathcal{D})$. In the case of linear regression, it is clear that Bayesian inference has advantages over the MAP estimate: we can quantify uncertainty of our predictions and compare the marginal likelihood between models to obtain the one explaining the data best. Nonetheless, computational feasibility is a key requirement of machine learning algorithms. Therefore, the MAP estimator is often the standard choice as it provides good results at fraction of the cost and effort.

1.2 Bayesian Deep Learning

Bayesian neural networks are probabilistic models where the likelihood is parameterized by a neural network and the prior is a distribution over the neural network parameters. Bayesian inference in these models is particularly challenging and therefore MAP estimation has played the major role in the past successes of *deep learning* [8, 24]. In contrast, the alternative strand of research termed *Bayesian deep learning* deals with *approximate Bayesian inference* for probabilistic neural network models. Instead of obtaining an exact posterior of the Bayesian inference problem, an approximation to the posterior is constructed [4, 19, 50, 55]. The key aspect of Bayesian deep learning is to maintain computational and performance advantages of deep learning while providing parameter uncertainties that enable predictive uncertainties.

Bayesian deep learning algorithms rely on approximate inference techniques as well as scalable optimization algorithms. The interplay of both enables to approximate the posterior distribution efficiently. In particular, the combination of the generalized Gauss-Newton method from the optimization literature and Gaussian posterior approximations has recently made Bayesian deep learning competitive with traditional deep learning and enabled new applications based on uncertainty [19, 38, 44, 55]. However, it is unclear how the combination of these approximations impacts the underlying inference problem. In this work, we address this problem by disentangling algorithms that make use of Gaussian posterior approximations and the generalized Gauss-Newton method in Bayesian deep learning. Decoupling the individual methods and analyzing them individually can potentially allow to understand the algorithms better, improve them, and fix existing pathologies.

1.3 Outline of the Thesis

In chapter 2, the necessary background on probabilistic neural network models, approximate inference, and the generalized Gauss-Newton method is introduced. In chapter 3, we disentangle the combination of the Laplace approximation with the generalized Gauss-Newton method for optimization. This allows to understand approximate inference for neural networks with linear and Gaussian process models. Further, we introduce new methods to compute the posterior predictive and marginal likelihood approximation of neural network models. Chapter 4 introduces a new variational inference algorithm and, in the same spirit as Chapter 3, describes approximate inference for neural networks via simpler linear and Gaussian process models. Chapter 5 provides experiments that explain and complement the theoretical connections: we show how to tune hyperparameters using the marginal likelihood and that the posterior predictive is in fact greatly influenced by the application of the Gauss-Newton method. The prediction methods introduced here fix a common problem with the Laplace approximation. The kernel of the identified Gaussian process formulation is further used to explain neural network predictions. Lastly, we discuss related and future work and conclude the thesis in Chapter 6.

Chapter 2

Background

In this chapter, we briefly introduce the necessary concepts and theory to follow the rest of this work. We first introduce probabilistic neural networks as the underlying models that we want to infer. In particular, we introduce likelihoods for supervised learning problems and discuss the choice of prior. Bayesian deep learning combines approximate inference methods and optimization algorithms, both of which we introduce in the end of this chapter. On the inference side, we introduce the Laplace and Gaussian variational approximation and relate them to each other. Lastly, we introduce the *generalized Gauss-Newton* approximation to the Hessian from the optimization literature. This enables scalable approximate inference.

Formally, we denote a neural network as $\boldsymbol{f}(\boldsymbol{x};\boldsymbol{\theta}) : \mathbb{R}^D \times \mathbb{R}^P \to \mathbb{R}^K$ that maps an input $\boldsymbol{x} \in \mathbb{R}^D$ to output $\boldsymbol{f} \in \mathbb{R}^K$ with parameters $\boldsymbol{\theta} \in \mathbb{R}^P$. We do not restrict ourselves to any form of neural network. In fact, any parametric function that is differentiable in $\boldsymbol{\theta}$ at least once can be used. In some cases, we have a scalar output, i.e., $f(\boldsymbol{x};\boldsymbol{\theta}) = f \in \mathbb{R}$.

The dataset in a supervised learning scenario is given as pairs of inputs $\boldsymbol{x}_i \in \mathbb{R}^D$ and labels $\boldsymbol{y}_i \in \mathbb{R}^K$. As introduced in Chapter 1, the entire dataset of size N is then given by $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^N$. We assume that all data points are drawn independently and from the same distribution. Therefore, we have the following likelihood in a neural network model:

$$p(\mathcal{D}|\boldsymbol{\theta}) = \prod_{i=1}^{N} p(\boldsymbol{y}_i|\boldsymbol{f}(\boldsymbol{x}_i;\boldsymbol{\theta})). \tag{2.1}$$

In deep learning, we obtain a MAP estimate by optimizing the corresponding objective in Equation 1.2. For computational reasons, we maximize the log joint distribution which leads to the more convenient objective

$$\boldsymbol{\theta}_{\text{MAP}} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{N} \log p(\boldsymbol{y}_i|\boldsymbol{f}(\boldsymbol{x}_i;\boldsymbol{\theta})) + \log p(\boldsymbol{\theta}), \tag{2.2}$$

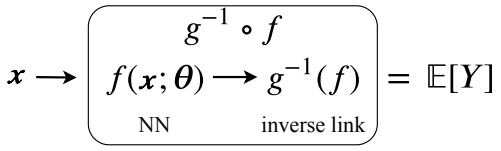


Figure 2.1 – Illustration of the composition of neural network and inverse link function. The inverse link function gives the mean of the modelled response variable Y.

which is also known as the *empirical risk minimization* objective. In the ERM setting, the likelihood acts as a loss per data point and the prior can be understood as a regularizer. Next, we specify a particular family of likelihoods that is sufficient for deep learning and possesses useful theoretical properties. After that, we introduce approximate Bayesian inference and the generalized Gauss-Newton method.

2.1 Probabilistic Neural Networks for Supervised Learning

We have introduced neural networks that map an input data point to an output value. A probabilistic neural network model additionally consists of a likelihood and a prior. We introduce a family of likelihoods, in particular, those of *generalized linear models* (GLMs), that give rise to common losses used in deep learning. These exponential family likelihoods possess simplifying theoretical properties that make later results more interpretable. In the end of the section, we discuss the choice of prior.

Generalized linear model likelihoods specify a distribution over the output labels. In particular, we model a response random variable Y or vector $\boldsymbol{Y}$ for a given input and neural network parameter. GLM likelihoods conveniently express the mean of the response via an invertible *link function* $g(\cdot)$, i.e., $\mathbb{E}[\boldsymbol{Y}] = \boldsymbol{g}^{-1}(\boldsymbol{f})$. Further, the derivatives of the log-likelihood with respect to the function $\boldsymbol{f}$ are directly related to the moments of $\boldsymbol{Y}$. This property makes the theoretical developments more intuitive and insightful. The likelihoods of generalized linear models are restricted to exponential family distributions [33]. Every exponential family can be written in the following *natural form* as

$$p(\boldsymbol{y}|\boldsymbol{f}) = h(\boldsymbol{y}) \exp\{\langle T(\boldsymbol{y}), \boldsymbol{f}\rangle - A(\boldsymbol{f})\}, \tag{2.3}$$

where $h(\boldsymbol{y})$ is the base measure, $T(\boldsymbol{y})$ the sufficient statistics, $A(\boldsymbol{f})$ the log-cumulant, and $\boldsymbol{f}$ the natural parameter. We restrict ourselves to forms where $T(\cdot)$ is the identity, i.e., $T(\boldsymbol{y}) = \boldsymbol{y}$. Therefore, we have

$$p(\boldsymbol{y}|\boldsymbol{f}) = h(\boldsymbol{y}) \exp\{\langle \boldsymbol{y}, \boldsymbol{f}\rangle - A(\boldsymbol{f})\}. \tag{2.4}$$

The derivative of the log likelihood with respect to the natural parameter $\boldsymbol{f}$ is particularly convenient. The first derivative forms a residual between the observed label and the mean of the

Distribution	σ^2	$\mathbb{E}[\boldsymbol{Y}] = \boldsymbol{g}^{-1}(\boldsymbol{f})$	$\boldsymbol{r}(\boldsymbol{y}, \boldsymbol{f})$	$\mathbb{V}[\boldsymbol{Y}]$	$\boldsymbol{\Lambda}(\boldsymbol{f})$
Gaussian	σ^2	f	$\sigma^{-2}(y - f)$	σ^2	σ^{-2}
Multivariate Gaussian	$\boldsymbol{\Sigma}$	$\boldsymbol{f}$	$\boldsymbol{\Sigma}^{-1}(\boldsymbol{y} - \boldsymbol{f})$	$\boldsymbol{\Sigma}$	$\boldsymbol{\Sigma}^{-1}$
Bernoulli	1	$\frac{1}{1+e^{-f}} = \sigma(f)$	$y - \sigma(f)$	$\sigma(f)(1 - \sigma(f))$	$\mathbb{V}[\boldsymbol{Y}]$
Categorical	1	$\mathrm{softmax}(\boldsymbol{f}) = \boldsymbol{p}(\boldsymbol{f})$	$\boldsymbol{y} - \boldsymbol{p}(\boldsymbol{f})$	$\mathrm{diag}(\boldsymbol{p}) - \boldsymbol{p}\boldsymbol{p}^\top$	$\mathbb{V}[\boldsymbol{Y}]$
Poisson	1	$\exp(f)$	$y - \exp(f)$	$\exp(f)$	$\mathbb{V}[\boldsymbol{Y}]$

Table 2.1 – Most common likelihoods of generalized linear models. For each distribution, we list the corresponding *dispersion* parameter, the mean response, the residual, the variance of the response, and the Hessian. In the context of deep learning, the Bernoulli and categorical likelihoods are common due to their application to classification problems.

response variable and therefore specifies the link function. The second derivative directly relates to the variance of the modelled response variable. Further, we identify the mean of the response variable and therefore the inverse link function as the derivative of the log cumulant $A(\boldsymbol{f})$. The following Lemma formalizes these properties.

Lemma 2.1. *Let $p(\boldsymbol{y}|\boldsymbol{f})$ be an exponential family distribution of the form in Equation 2.4 and let $\boldsymbol{Y}$ denote the corresponding random variable. The first and second derivative of the log likelihood take the simple form*

$$\nabla_{\boldsymbol{f}} \log p(\boldsymbol{y}|\boldsymbol{f}) = = \boldsymbol{y} - \nabla_{\boldsymbol{f}} A(\boldsymbol{f}) = \boldsymbol{y} - \mathbb{E}[\boldsymbol{Y}] = \boldsymbol{y} - \boldsymbol{g}^{-1}(\boldsymbol{f}) =: \boldsymbol{r}(\boldsymbol{y}, \boldsymbol{f}), \tag{2.5}$$

$$\nabla^2_{\boldsymbol{f}\boldsymbol{f}} \log p(\boldsymbol{y}|\boldsymbol{f}) = -\nabla^2_{\boldsymbol{f}\boldsymbol{f}} A(\boldsymbol{f}) = -\mathbb{V}[\boldsymbol{Y}] =: -\boldsymbol{\Lambda}(\boldsymbol{f}), \tag{2.6}$$

where we defined the residual $\boldsymbol{r}(\boldsymbol{y}, \boldsymbol{f}) \in \mathbb{R}^K$ and second derivative, or Hessian, of the negative log likelihood $\boldsymbol{\Lambda}(\boldsymbol{f}) \in \mathbb{R}^{K \times K}$. We have thus identified the inverse link that gives the mean of the response variable as the first derivative of the log cumulant. In the case of an overdispersed *exponential family [33], the variance of the response variable is further scaled by the dispersion parameter σ^2 while the derivatives are divided by it. We have this case, for example, in a Gaussian likelihood (see Table 2.1).*

A proof of the Lemma can be found in Appendix A. In Table 2.1, we list the most notable examples of likelihoods. In this work, we mostly deal with the Gaussian univariate and Bernoulli likelihood. Both naturally extend to their multivariate counter-parts, the multivariate Gaussian and categorical distribution. In the empirical risk minimization perspective, a Gaussian likelihood yields a least-squares loss used for regression problems. The Bernoulli and categorical distributions give rise to the commonly used *cross-entropy* loss for classification [12].

As prior we choose a multivariate Gaussian in line with most works on Bayesian neural networks [4, 19, 55]. In the MAP estimation framework, a Gaussian prior corresponds to ℓ_2 regularization commonly known as *weight decay* in deep learning [2, 12, 33]. We therefore have

$$\boldsymbol{\theta} \sim p(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0), \tag{2.7}$$

where $\boldsymbol{\mu}_0 \in \mathbb{R}^P$ is the mean and $\boldsymbol{\Sigma}_0 \in \mathbb{R}^{P \times P}$ the covariance. For neural networks, the mean is typically chosen to be zero and the covariance is either diagonal, i.e., $\boldsymbol{\sigma}_0^2 \in \mathbb{R}_+^P$, or scalar $\sigma_0^2 \in \mathbb{R}_+$. That is because it is nontrivial to specify a prior over the parameters of a neural network and the Gaussian is a comparably agnostic default choice [27, 34, 36]. It is important to note that the prior should be understood as a distribution in Bayesian inference and as a regularizer in MAP estimation. In chapter 3, we pick up on this distinction in the context of approximate Bayesian inference.

2.2 Approximate Bayesian Inference

Exact Bayesian inference in a probabilistic neural network model is typically intractable. Therefore, we need to resort to approximate inference methods that provide scalable and computable alternatives to the marginal likelihood. In the context of networks, the most scalable and practical techniques are variants of the *Laplace* [10, 27, 44] and *Gaussian variational* (GVA) approximation [4, 13, 19, 55]. In this work, we therefore focus on these two approximations. Both, the Laplace approximation and the GVA turn the problem of integration into a problem of optimization followed by a simple closed-form integration. Other approximate inference methods such as *Markov chain Monte Carlo* [35, 36] or *expectation* and *belief propagation* [31, 39] are typically not as scalable and are therefore rarely used for neural networks.

The Laplace and Gaussian variational approximation both construct a Gaussian approximation to the true posterior distribution. The Laplace uses a second-order approximation of the log joint distribution to approximate the marginal likelihood. In contrast, the GVA maximizes the variational lower bound to the marginal likelihood. Both methods give rise to a posterior approximation $q(\boldsymbol{\theta})$ and we have

$$q(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) \approx p(\boldsymbol{\theta}|\mathcal{D}), \tag{2.8}$$

where $\boldsymbol{\mu} \in \mathbb{R}^P$ and $\boldsymbol{\Sigma} \in \mathbb{R}^{P \times P}$ are the free parameters. Typically, the variational approximation is preferred because it can capture the shape of the posterior better [2, 33]. After introducing both methods, we briefly show how they are related.

The Laplace Approximation

The Laplace approximation is a heuristic that fits a Gaussian distribution locally at the MAP estimate (Equation 2.2). Therefore, the Laplace approximation comprises two steps: first, we obtain the MAP estimate $\boldsymbol{\theta}_{\text{MAP}}$ using an optimization method and then we approximate the log joint distribution to the second-order using a Taylor expansion. The second-order approximation allows us to compute the marginal likelihood in closed form. The Laplace approximation therefore provides an approximation to the posterior and the marginal likelihood.

Since the gradient at the MAP is zero, i.e. $\nabla_{\boldsymbol{\theta}} \log p(\mathcal{D}, \boldsymbol{\theta}_{\text{MAP}}) = \mathbf{0}$, the second-order Taylor approximation at this point is simple. We have

$$\log p(\mathcal{D}, \boldsymbol{\theta}) \approx \log p(\mathcal{D}, \boldsymbol{\theta}_{\text{MAP}}) + \frac{1}{2} (\boldsymbol{\theta} - \boldsymbol{\theta}_{\text{MAP}})^\top \nabla^2_{\boldsymbol{\theta}\boldsymbol{\theta}} \log p(\mathcal{D}, \boldsymbol{\theta}_{\text{MAP}}) (\boldsymbol{\theta} - \boldsymbol{\theta}_{\text{MAP}}) . \tag{2.9}$$

Then, computing the marginal likelihood of the approximated model has a closed-form solution. Using the normalization properties of a multivariate normal distribution, we obtain the Laplace approximation to the marginal likelihood

$$p(\mathcal{D}) \approx p(\mathcal{D}, \boldsymbol{\theta}_{\text{MAP}}) (2\pi)^{\frac{P}{2}} \det(-\nabla^2_{\boldsymbol{\theta}\boldsymbol{\theta}} \log p(\mathcal{D}, \boldsymbol{\theta}_{\text{MAP}}))^{-\frac{1}{2}}. \tag{2.10}$$

Exponentiating the approximation to the joint distribution in Equation 2.9 and dividing it by the marginal likelihood, we identify the Laplace approximation as a Gaussian distribution with mean and covariance given by

$$\boldsymbol{\mu} = \boldsymbol{\theta}_{\text{MAP}} \quad \text{and} \quad \boldsymbol{\Sigma} = [-\nabla^2_{\boldsymbol{\theta}\boldsymbol{\theta}} \log p(\mathcal{D}, \boldsymbol{\theta}_{\text{MAP}})]^{-1}. \tag{2.11}$$

The Laplace approximation is a practical method for approximate Bayesian inference because we only need to find a MAP estimate and compute the Hessian of the log joint distribution at that estimate. Deep learning optimizers provide effective means to obtain MAP estimates. For large networks, however, computing the Hessian for a neural network model is permissively complex in terms of storage and computation. Further, it might even give undesirable results: the Hessian at the MAP we obtain is not necessarily a positive definite matrix and might therefore not be invertible [45, 46]. Typically, this problem is tackled by optimization methods that guarantee a positive semi-definite Hessian approximation. For example, Gauss-Newton methods [14, 19, 44] ensure an invertible Hessian approximation. In section 2.3, we introduce the generalized Gauss-Newton method that is commonly employed.

The Gaussian Variational Approximation

In *variational inference*, we minimize the *Kullback-Leibler* (KL) divergence of the true posterior $p(\boldsymbol{\theta}|\mathcal{D})$ from an approximation distribution $q(\boldsymbol{\theta})$ [3, 16]. Let $\mathcal{P}$ be a family of distributions that we choose as our posterior approximating family. Then, the variational approximation is given by the following optimization problem:

$$\begin{aligned} q_*(\boldsymbol{\theta}) &= \arg\min_{q \in \mathcal{P}} D_{\text{KL}} \left[q(\boldsymbol{\theta}) \| p(\boldsymbol{\theta}|\mathcal{D}) \right] = \arg\min_{q \in \mathcal{P}} \int q(\boldsymbol{\theta}) \log \left(\frac{q(\boldsymbol{\theta}) p(\mathcal{D})}{p(\mathcal{D}|\boldsymbol{\theta}) p(\boldsymbol{\theta})} \right) d\boldsymbol{\theta} \\ &= \arg\min_{q \in \mathcal{P}} \int q(\boldsymbol{\theta}) \log \left(\frac{q(\boldsymbol{\theta})}{p(\mathcal{D}, \boldsymbol{\theta})} \right) d\boldsymbol{\theta} + \log p(\mathcal{D}). \end{aligned} \tag{2.12}$$

If the approximating family $\mathcal{P}$ contains the true posterior distribution, the variational approximation is exact and naturally incurs no divergence. Since the KL divergence is non-negative, the

first term in Equation 2.12 stands in special relation to the log marginal likelihood giving rise to the *evidence lower bound* (ELBO):

$$\log p(\mathcal{D}) \geq \mathbb{E}_q \left[\log \frac{p(\mathcal{D}, \boldsymbol{\theta})}{q(\boldsymbol{\theta})} \right] = \text{ELBO}(q). \tag{2.13}$$

Maximizing the ELBO is hence equivalent to minimizing the KL divergence of the true posterior from the approximating distribution.

The Gaussian variational approximation (GVA) is a particular instance of variational inference where $\mathcal{P}$ is the family of multivariate Gaussian distributions. In particular, the approximating distribution $q(\boldsymbol{\theta})$ is restricted to the Gaussian distribution $\mathcal{N}(\boldsymbol{\theta}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$ as stated in Equation 2.8. Therefore, the problem of optimizing over a family of distributions turns into optimizing the parameters of a distribution of fixed form. The ELBO for a probabilistic neural network model with Gaussian prior (section 2.1) can therefore be expressed in terms of the parameters $\boldsymbol{\mu}, \boldsymbol{\Sigma}$. Further, it is convenient to split the ELBO into an expected log likelihood and a KL divergence term:

$$\text{ELBO}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \mathbb{E}_{\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})} \left[\log p(\mathcal{D}|\boldsymbol{\theta}) \right] - D_{\text{KL}} \left[\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \| \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0) \right]. \tag{2.14}$$

The KL divergence has a closed form solution since both prior and approximating distribution are Gaussian. This form of the ELBO provides the basis of variational inference in deep learning [4, 19, 55]. The expected log likelihood term of the ELBO seems to be complicated to optimize. However, a neat relation between derivatives with respect to the parameters $(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ and realized samples $\boldsymbol{\theta}_s$ of the GVA exists [37] and is due to Bonnet and Price [5, 41]. We can simplify the derivative with respect to parameters $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ of the expected log likelihood as follows:

$$\nabla_{\boldsymbol{\mu}} \mathbb{E}_{\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})} \left[\log p(\mathcal{D}|\boldsymbol{\theta}) \right] = \mathbb{E}_{\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})} \left[\nabla_{\boldsymbol{\theta}} \log p(\mathcal{D}|\boldsymbol{\theta}) \right] \tag{2.15}$$

$$\nabla_{\boldsymbol{\Sigma}} \mathbb{E}_{\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})} \left[\log p(\mathcal{D}|\boldsymbol{\theta}) \right] = \frac{1}{2} \mathbb{E}_{\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})} \left[\nabla^2_{\boldsymbol{\theta}\boldsymbol{\theta}} \log p(\mathcal{D}|\boldsymbol{\theta}) \right] \tag{2.16}$$

Therefore, taking gradients with respect to the variational parameters can be as simple as sampling and taking the gradient with respect to individual samples. Further, these equations make clear that the GVA is also limited to a second-order approximation. However, the GVA maintains a global *view* of the loss due to the expectation and is therefore more powerful than the Laplace approximation.

Relation between Laplace and Variational Approximation

Following Opper and Archambeau [37], we compare the optimality criteria of the Laplace and Gaussian variational approximation. For the Laplace approximation, we have the conditions

$$0 = \nabla_{\boldsymbol{\theta}} \log p(\mathcal{D}, \boldsymbol{\mu}) \quad \text{and} \quad \boldsymbol{\Sigma}^{-1} = -\nabla^2_{\boldsymbol{\theta}\boldsymbol{\theta}} \log p(\mathcal{D}, \boldsymbol{\mu}). \tag{2.17}$$

For the Gaussian variational approximation, we can devise very similar conditions. First, we obtain the stationarity conditions by differentiating with respect to the variational parameters in Equation 2.14. The stationarity condition for the mean is simple and for the inverse covariance matrix, we have

$$\nabla_{\boldsymbol{\Sigma}}\text{ELBO}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) = 0 \rightarrow \boldsymbol{\Sigma}^{-1} = 2\nabla_{\boldsymbol{\Sigma}}\mathbb{E}_{\mathcal{N}(\boldsymbol{\mu},\boldsymbol{\Sigma})}\left[-\log p(\mathcal{D}, \boldsymbol{\theta})\right]. \tag{2.18}$$

Now, we apply equalities of Equation 2.15 and Equation 2.16 to both stationarity conditions. Note that these equalities hold not only for the expectation over a likelihood but also the joint distribution [37]. We obtain the GVA stationarity conditions

$$0 = \mathbb{E}_{\mathcal{N}(\boldsymbol{\mu},\boldsymbol{\Sigma})}\left[\nabla_{\boldsymbol{\theta}} \log(\mathcal{D}, \boldsymbol{\theta})\right] \quad \text{and} \quad \boldsymbol{\Sigma}^{-1} = \mathbb{E}_{\mathcal{N}(\boldsymbol{\mu},\boldsymbol{\Sigma})}\left[-\nabla^2_{\boldsymbol{\theta}\boldsymbol{\theta}} \log p(\mathcal{D}, \boldsymbol{\theta})\right]. \tag{2.19}$$

The relation between Equation 2.17 and Equation 2.19 highlights the difference between both approximations: while the Laplace approximation is only defined locally at the MAP, the variational approximation holds globally [37]. The relation suggests that the difference lies in sampling parameters versus fixed parameters. Specifically in practice, the variational inference stationarity in Equation 2.19 does not hold exactly but for S Monte Carlo samples from the variational approximation, i.e., $\boldsymbol{\theta}^{(1)}, \ldots, \boldsymbol{\theta}^{(S)} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$:

$$0 = \frac{1}{S}\sum_{s=1}^{S} \nabla_{\boldsymbol{\theta}} \log(\mathcal{D}, \boldsymbol{\theta}^{(s)}) \quad \text{and} \quad \boldsymbol{\Sigma}^{-1} = \frac{1}{S}\sum_{s=1}^{S} -\nabla^2_{\boldsymbol{\theta}\boldsymbol{\theta}} \log p(\mathcal{D}, \boldsymbol{\theta}^{(s)}). \tag{2.20}$$

The main insight of this section is that both approximations limit themselves to the first two moments of the negative log joint distribution. The variational approximation provides a more global view of the joint distribution by stochasticity in the parameters.

2.3 The Generalized Gauss-Newton Method

For approximate Bayesian inference in neural networks, second-order derivatives of the log likelihood are typically required. In particular, this is the case for the Laplace and Gaussian variational approximations. As mentioned in section 2.2 however, the Hessian is permissively expensive to compute and might even be singular or undefined. Therefore, the current state-of-the-art Bayesian deep learning algorithms rely on approximate second-order optimization methods [19, 38, 55]. These methods are both scalable and guarantee a positive semi-definite approximation to the Hessian [6, 29, 47]. In particular, the *generalized Gauss-Newton approximation* (GGN) is used extensively for the Laplace and Gaussian variational approximation [10, 19, 30, 44, 55]. The GGN is a positive semi-definite approximation to the Hessian. In practice, the diagonal [13, 19] or a Kronecker factorization [44, 55] of the GGN is often used. Here, we work with a full GGN approximation to draw conclusions for the other special cases.

The generalized Gauss-Newton method allows us to compute an approximation to the second derivative of the log-likelihood. The derivative of the log prior in the Laplace approximation and the KL divergence from the prior in the GVA have a closed form and do not require an approximation. The log likelihood takes the following form in the case of probabilistic neural network models. According to Equation 2.1, we have

$$\log p(\mathcal{D}|\boldsymbol{\theta}) = \sum_{i=1}^{N} \log p(\boldsymbol{y}_i|\boldsymbol{f}(\boldsymbol{x}_i;\boldsymbol{\theta})), \tag{2.21}$$

where the likelihood is a generalized linear model likelihood (section 2.1). To take the first and second derivative with respect to the parameter, we apply the chain rule. Figure 2.1 illustrates that we can first differentiate with respect to $\boldsymbol{f}$ and then with respect to the parameters. We define the *Jacobian matrix* $\boldsymbol{J}(\boldsymbol{x};\boldsymbol{\theta}) \in \mathbb{R}^{K\times P}$ of $\boldsymbol{f}(\boldsymbol{x};\boldsymbol{\theta})$ with respect to the parameters, and the *Hessian tensor* $\mathsf{H}(\boldsymbol{x};\boldsymbol{\theta}) \in \mathbb{R}^{K\times P\times P}$ of second derivatives as

$$[\boldsymbol{J}(\boldsymbol{x};\boldsymbol{\theta})]_{ij} = \frac{\partial f_i(\boldsymbol{x};\boldsymbol{\theta})}{\partial \theta_j} \quad \text{and} \quad [\mathsf{H}(\boldsymbol{x};\boldsymbol{\theta})]_{ijk} = \frac{\partial^2 f_i(\boldsymbol{x};\boldsymbol{\theta})}{\partial \theta_j \partial \theta_k}, \tag{2.22}$$

where we assumed the function is twice differentiable for now. Using the properties of the first and second derivative of the GLM log likelihoods in Lemma 2.1 and Table 2.1, we obtain the gradient

$$\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{y}|\boldsymbol{f}(\boldsymbol{x};\boldsymbol{\theta})) = \boldsymbol{J}(\boldsymbol{x};\boldsymbol{\theta})^\top \nabla_{\boldsymbol{f}} \log p(\boldsymbol{y}|\boldsymbol{f}) = \boldsymbol{J}(\boldsymbol{x};\boldsymbol{\theta})^\top \boldsymbol{r}(\boldsymbol{y},\boldsymbol{f}). \tag{2.23}$$

Similarly, the Hessian of the log likelihood can be computed using the chain rule and gives

$$\begin{aligned} \nabla^2_{\boldsymbol{\theta}\boldsymbol{\theta}} \log p(\boldsymbol{y}|\boldsymbol{f}(\boldsymbol{x};\boldsymbol{\theta})) &= \mathsf{H}(\boldsymbol{x};\boldsymbol{\theta})^\top \nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{y}|\boldsymbol{f}) + \boldsymbol{J}(\boldsymbol{x};\boldsymbol{\theta})^\top \nabla^2_{\boldsymbol{f}\boldsymbol{f}} \log p(\boldsymbol{y}|\boldsymbol{f}) \boldsymbol{J}(\boldsymbol{x};\boldsymbol{\theta}) \\ &= \mathsf{H}(\boldsymbol{x};\boldsymbol{\theta})^\top \boldsymbol{r}(\boldsymbol{y},\boldsymbol{f}) - \boldsymbol{J}(\boldsymbol{x};\boldsymbol{\theta})^\top \boldsymbol{\Lambda}(\boldsymbol{f}) \boldsymbol{J}(\boldsymbol{x};\boldsymbol{\theta}). \end{aligned} \tag{2.24}$$

The first derivative is tractable and efficiently implemented in neural networks using *back-propagation* [12]. The second derivative with respect to the parameters is problematic because we need to differentiate the neural network with respect to its large amount of parameters twice. In fact, for some network architectures, for example, `ReLU` activation functions, the second derivative is not defined everywhere [55]. The generalized Gauss-Newton approximation to the Hessian simply removes the term that is intractable and is given by

$$\nabla^2_{\boldsymbol{\theta}\boldsymbol{\theta}} \log p(\boldsymbol{y}|\boldsymbol{f}(\boldsymbol{x};\boldsymbol{\theta})) \approx -\boldsymbol{J}(\boldsymbol{x};\boldsymbol{\theta}) \boldsymbol{\Lambda}(\boldsymbol{f}) \boldsymbol{J}(\boldsymbol{x};\boldsymbol{\theta})^\top. \tag{2.25}$$

This makes the Hessian tractable since we only need first order derivatives with respect to the neural network. Further, it is always positive semi-definite and does not even require the existence of the neural network Hessian.

Assuming the neural network Hessian $\mathsf{H}(\boldsymbol{x};\boldsymbol{\theta})$ exists, the GGN approximation is exact in two cases: either, all residuals are zero, i.e., $\forall(\boldsymbol{x},\boldsymbol{y}) \in \mathcal{D} : \boldsymbol{r}(\boldsymbol{y}, f(\boldsymbol{x};\boldsymbol{\theta})) = \boldsymbol{0}$, or the neural network Hessian $\mathsf{H}(\boldsymbol{x};\boldsymbol{\theta})$ is zero. Although a neural network can potentially achieve zero residuals, it is

both undesirable as it indicates *overfitting* and impractical as this condition does not hold at initialization or during training.[1] The neural network Hessian $\mathbf{H}(\boldsymbol{x};\boldsymbol{\theta})$ can only be zero everywhere if the neural network is linear. Therefore, an alternative derivation of the GGN approximation to the Hessian starts from linearization of the neural network [6, 29]. We define the first order Taylor approximation of the neural network around the expansion point $\boldsymbol{\theta}_*$ as

$$\boldsymbol{f}_{\text{lin}}^{\boldsymbol{\theta}_*}(\boldsymbol{x};\boldsymbol{\theta}) = \boldsymbol{f}(\boldsymbol{x};\boldsymbol{\theta}_*) + \boldsymbol{J}(\boldsymbol{x};\boldsymbol{\theta}_*)(\boldsymbol{\theta} - \boldsymbol{\theta}_*), \tag{2.26}$$

which gives us a linear function in the parameters $\boldsymbol{\theta}$ but not in the input $\boldsymbol{x}$. To compute the Hessian at $\boldsymbol{\theta}_*$, we therefore linearize the network at this parameter and then compute the Hessian. This way, we recover the GGN approximation to the Hessian. This derivation and reasoning for the GGN allows to understand its role in approximate inference better.

[1] For the Bernoulli and categorical likelihood, the residual can theoretically only tend towards zero.

Chapter 3

From Neural Network to Gaussian Process with Laplace and Gauss-Newton

The Laplace approximation is often used as a baseline for Bayesian neural networks and in some cases achieves state of the art results [43, 44, 49]. For neural networks, the diagonal or a Kronecker-factored generalized Gauss-Newton approximation to the Hessian is often used [44]. Here, we will work with the full GGN approximation to the Hessian [10]. We call the combination of Laplace and GGN the Laplace-GGN approximation [20]. In this chapter, we disentangle the Laplace-GGN approximation. In particular, we first apply the generalized Gauss-Newton method and then make use of the Laplace approximation. Going forward, we analyze the individual steps of the Laplace-GGN and their impact on the underlying probabilistic model. Interestingly, the underlying probabilistic model can be cast as a Gaussian process model and enables function-space inference for neural networks.

First, we apply the generalized Gauss-Newton method to the neural network model. This gives rise to a generalized linear model due to the linearizing property of the GGN. Equivalently, we can specify this model as a Gaussian process (GP) model. Subsequently, we apply the Laplace approximation to either of these models. We identify two simple models, the Bayesian linear and the Gaussian process regression model. Inferring these models is then *equivalent* to the Laplace-GGN approximation and facilitates a better understanding of the combination of Laplace and GGN approximation. Figure 3.1 illustrates the steps and relationships established in this chapter.

After analyzing the Laplace and GGN approximation, we compute the posterior, posterior predictive, and marginal likelihood. Due to the equivalence to Gaussian process inference, there are two ways for each of these quantities. This can enable computational advantages in some cases, similar to the *kernel trick*. The Gaussian process viewpoint further enables model inspection and interpretability as we demonstrate in chapter 5.

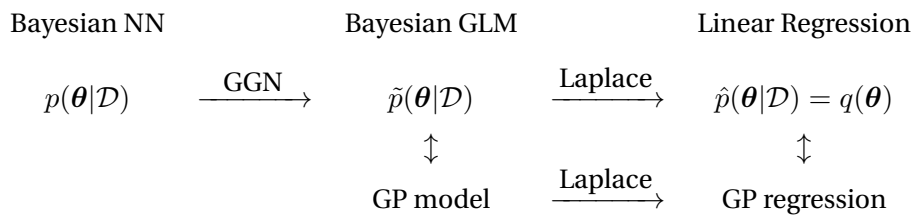


Figure 3.1 – Illustration of approximate inference with the Laplace-GGN method. We start with Bayesian neural network inference and, by applying the GGN and Laplace approximation, ultimately infer a Bayesian linear regression, or equivalently, GP regression model. As an intermediate step, the Bayesian neural network is turned into a generalized linear model due to the GGN. $\tilde{p}(\boldsymbol{\theta}|\mathcal{D})$ is the true posterior of the GLM and $\hat{p}(\boldsymbol{\theta}|\mathcal{D})$ is the true posterior of the Bayesian linear regression model which is equal to the Laplace-GGN approximation.

3.1 From Neural Network to Generalized Linear Model

We start with a probabilistic neural network model as specified in chapter 2. During optimization of the MAP objective, we have the iterate $\boldsymbol{\theta}_*$. Typically, $\boldsymbol{\theta}_*$ is an estimate of the MAP but for our development this is not necessary. The generalized Gauss-Newton approximation performs a linearization around the current parameter. We recall Equation 2.26 from chapter 2:

$$\boldsymbol{f}_{\text{lin}}^{\boldsymbol{\theta}_*}(\boldsymbol{x};\boldsymbol{\theta}) = \boldsymbol{f}(\boldsymbol{x};\boldsymbol{\theta}_*) + \boldsymbol{J}(\boldsymbol{x};\boldsymbol{\theta}_*)(\boldsymbol{\theta} - \boldsymbol{\theta}_*). \tag{3.1}$$

The linearization above changes our model locally. We call $\tilde{p}(\boldsymbol{\theta}|\mathcal{D})$ the posterior of a generalized linear model. Replacing the neural network $\boldsymbol{f}(\boldsymbol{x};\boldsymbol{\theta})$ by its linearized version in the probabilistic model, we then have

$$\tilde{p}(\boldsymbol{\theta}|\mathcal{D}) \propto p(\boldsymbol{\theta}) \prod_{i=1}^{N} p(\boldsymbol{y}_i|\boldsymbol{f}_{\text{lin}}^{\boldsymbol{\theta}_*}(\boldsymbol{x}_i;\boldsymbol{\theta})), \tag{3.2}$$

which is a generalized linear model (GLM) since $\boldsymbol{f}_{\text{lin}}^{\boldsymbol{\theta}_*}(\boldsymbol{x};\boldsymbol{\theta})$ is linear in the parameter $\boldsymbol{\theta}$. The Jacobian can therefore be understood as a *local* feature map of the inputs. Equivalently, we can therefore transform this inference problem from the *weight-space* to the *function-space* [42]. We obtain a Gaussian process prior by taking the expectation of the linearized neural network under the parametric prior $p(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0)$. The mean and covariance function of the Gaussian process prior are given by

$$\boldsymbol{m}(\boldsymbol{x}) = \mathbb{E}_{p(\boldsymbol{\theta})}\left[\boldsymbol{f}_{\text{lin}}^{\boldsymbol{\theta}_*}(\boldsymbol{x};\boldsymbol{\theta})\right] = \boldsymbol{f}_{\text{lin}}^{\boldsymbol{\theta}_*}(\boldsymbol{x};\boldsymbol{\mu}_0), \tag{3.3}$$

$$\boldsymbol{\kappa}(\boldsymbol{x},\boldsymbol{x}') = \text{Cov}_{p(\boldsymbol{\theta})}\left[\boldsymbol{f}_{\text{lin}}^{\boldsymbol{\theta}_*}(\boldsymbol{x};\boldsymbol{\theta}), \boldsymbol{f}_{\text{lin}}^{\boldsymbol{\theta}_*}(\boldsymbol{x}';\boldsymbol{\theta})\right] = \boldsymbol{J}(\boldsymbol{x};\boldsymbol{\theta}_*)^\top \boldsymbol{\Sigma}_0 \boldsymbol{J}(\boldsymbol{x}';\boldsymbol{\theta}_*), \tag{3.4}$$

which gives rise to the Gaussian process prior in common notation [42] as

$$\boldsymbol{f}_{\text{GP}}(\boldsymbol{x}) \sim \mathcal{GP}(\boldsymbol{m}(\boldsymbol{x}), \boldsymbol{\kappa}(\boldsymbol{x},\boldsymbol{x}')). \tag{3.5}$$

We define the posterior Gaussian process distribution by $\tilde{p}(\boldsymbol{f}_{\text{GP}}|\mathcal{D})$ in line with the corresponding GLM. We therefore have the following generalized Gaussian process model (GGPM) as termed by Chan and Dong [7]:

$$\tilde{p}(\boldsymbol{f}_{\text{GP}}|\mathcal{D}) \propto p(\boldsymbol{f}_{\text{GP}}) \prod_{i=1}^{N} p(\boldsymbol{y}_i|\boldsymbol{f}_{\text{GP}}). \tag{3.6}$$

The GLM and the GP model have the same posterior predictive since they specify the same prior over functions [42].

So far, we have applied the GGN optimization method but no particular inference algorithm. Nonetheless, the underlying probabilistic neural network models has turned into a generalized linear model. The Jacobian, which constitutes the features of the GLM, remains fixed after this transformation. The second step is now to apply the Laplace approximation to the GLM. In this context, the Laplace approximation can therefore not really be understood as approximate inference of a neural network model. In the next sections, we show that the Laplace approximation to the GLM or GGPM is equivalent to solving a Bayesian linear regression or GP regression model, respectively. The exact posterior of these models then corresponds to the Laplace-GGN approximation.

3.2 From Generalized Linear Model to Bayesian Linear Regression

The Laplace-GGN approximation is a Laplace approximation to the local generalized linear model in Equation 3.2. In this section, we show that the Laplace approximation to this generalized linear model can be understood as exact inference in a Bayesian linear regression (BLR) model. The form of the Bayesian linear regression model is very intuitive and improves our understanding of the Laplace-GGN approximation. To keep notation simple, we abbreviate all relevant quantities as follows: we write $\boldsymbol{f}(\boldsymbol{x}) := \boldsymbol{f}(\boldsymbol{x};\boldsymbol{\theta}_*)$, $\boldsymbol{g}^{-1}(\boldsymbol{x}) := \boldsymbol{g}^{-1}(\boldsymbol{f}(\boldsymbol{x}_i;\boldsymbol{\theta}_*))$, $\boldsymbol{J}(\boldsymbol{x}) := \boldsymbol{J}(\boldsymbol{x};\boldsymbol{\theta}_*)$, and $\boldsymbol{\Lambda}(\boldsymbol{x}) := \boldsymbol{\Lambda}(\boldsymbol{f}(\boldsymbol{x};\boldsymbol{\theta}_*))$. We obtain the following result:

Theorem 3.1. *The Laplace-GGN approximation to a Bayesian neural network model, i.e., applying the Laplace approximation to the generalized linear model in Equation 3.2, is equivalent to the exact posterior of a Bayesian linear regression model. In particular, we obtain for the Laplace approximation that*

$$q(\boldsymbol{\theta}) = \hat{p}(\boldsymbol{\theta}|\mathcal{D}) \propto p(\boldsymbol{\theta}) \prod_{i=1}^{N} \mathcal{N}\left(\boldsymbol{y}_i|\boldsymbol{g}^{-1}(\boldsymbol{x}_i) + \boldsymbol{\Lambda}(\boldsymbol{x}_i)\boldsymbol{J}(\boldsymbol{x}_i)\left(\boldsymbol{\theta} - \boldsymbol{\theta}_*\right), \boldsymbol{\Lambda}(\boldsymbol{x}_i)\right), \tag{3.7}$$

where the original likelihood of the Bayesian NN model or GLM is replaced by a Gaussian likelihood. The Gaussian likelihood matches its first two moments at $\boldsymbol{\theta}_*$ for likelihoods with dispersion parameter $\sigma^2 = 1$, since $\boldsymbol{\Lambda}(\boldsymbol{x}_i) = \mathbb{V}[\boldsymbol{Y}_i]$ and the inverse link maps to the mean. For the overdispersed Gaussian likelihood, this connection is not useful since the GLM is already a Bayesian linear regression model.

Proof. We apply the Laplace approximation to the model with the linearized neural network in Equation 3.2. We handle the log likelihood and log prior individually: Since the log density of Gaussian is of second order, the prior remains unchanged. We work with a single summand of the log likelihood and use the representative data pair $(\boldsymbol{x}, \boldsymbol{y})$ and drop the dependency on $\boldsymbol{x}$ in the notation to save space, i.e., for any operator $A(\boldsymbol{x})$ we write only A. The residual vector $\boldsymbol{r}(\boldsymbol{y}, \boldsymbol{f}(\boldsymbol{x}; \boldsymbol{\theta}_*))$ is further abbreviated as $\boldsymbol{r}$. Therefore, the second-order Taylor approximation to the log likelihood around $\boldsymbol{\theta}_*$ required for the Laplace approximation can be written as

$$\begin{aligned}
\log p(\boldsymbol{y}|\boldsymbol{f}_{\text{lin}}^{\boldsymbol{\theta}_*}(\boldsymbol{x}; \boldsymbol{\theta})) &\approx \log p(\boldsymbol{y}|\boldsymbol{f}_{\text{lin}}^{\boldsymbol{\theta}_*}(\boldsymbol{x}; \boldsymbol{\theta}_*)) + \boldsymbol{r}\boldsymbol{J}\left(\boldsymbol{\theta} - \boldsymbol{\theta}_*\right) - \frac{1}{2}\left(\boldsymbol{\theta} - \boldsymbol{\theta}_*\right)^\top \boldsymbol{J}^\top \boldsymbol{\Lambda} \boldsymbol{J}\left(\boldsymbol{\theta} - \boldsymbol{\theta}_*\right) \\
&= \log p(\boldsymbol{y}|\boldsymbol{f}) - \left(\boldsymbol{g}^{-1}(\boldsymbol{f}) - \boldsymbol{y}\right)\left(\boldsymbol{J}\left(\boldsymbol{\theta} - \boldsymbol{\theta}_*\right)\right) - \frac{1}{2}\left(\boldsymbol{J}\left(\boldsymbol{\theta} - \boldsymbol{\theta}_*\right)\right)^\top \boldsymbol{\Lambda}\left(\boldsymbol{J}\left(\boldsymbol{\theta} - \boldsymbol{\theta}_*\right)\right) \\
&= \log p(\boldsymbol{y}|\boldsymbol{f}) + \frac{1}{2}\left(\boldsymbol{g}^{-1}(\boldsymbol{f}) - \boldsymbol{y}\right)^\top \boldsymbol{\Lambda}^{-1}\left(\boldsymbol{g}^{-1}(\boldsymbol{f}) - \boldsymbol{y}\right) \\
&\quad - \frac{1}{2}\left(\boldsymbol{g}^{-1}(\boldsymbol{f}) + \boldsymbol{\Lambda}\boldsymbol{J}\left(\boldsymbol{\theta} - \boldsymbol{\theta}_*\right) - \boldsymbol{y}\right)^\top \boldsymbol{\Lambda}^{-1}\left(\boldsymbol{g}^{-1}(\boldsymbol{f}) + \boldsymbol{\Lambda}\boldsymbol{J}\left(\boldsymbol{\theta} - \boldsymbol{\theta}_*\right) - \boldsymbol{y}\right),
\end{aligned}$$

where we have expanded the residual $\boldsymbol{r}$ and completed the square (Appendix B). Exponentiating the term and removing the parts independent of the parameter $\boldsymbol{\theta}$, we obtain the desired result. For the Laplace approximation to the marginal likelihood, the remaining constants will be important (cf. section 3.6). □

The result makes two interesting things apparent: first, the term $\boldsymbol{\Lambda}\boldsymbol{J}$ is equal to the Jacobian of $\boldsymbol{g}^{-1}(f(\boldsymbol{x}; \boldsymbol{\theta}))$ evaluated at $\boldsymbol{\theta}_*$. This follows from the GLM likelihoods since the derivative of the inverse link is $\boldsymbol{\Lambda}$ and therefore we obtain the Jacobian by the chain-rule. This indicates that Laplace-GGN implicitly performs a first-order Taylor approximation *after* the inverse link function. Second, when using the Laplace approximation to a linear model, we essentially turn the model into a solvable Bayesian linear regression model. This Bayesian linear regression model matches the moments of the original likelihood at the expansion point $\boldsymbol{\theta}_*$. In the case of maximum likelihood estimation, Wedderburn [51] has shown something similar: using the generalized Gauss-Newton for generalized linear models requires iterative solutions of least-squares problems that are weighted by the response variances.

3.3 From Generalized Gaussian Process to Gaussian Process Regression

We have explained how the generalized linear model due to the GGN (Equation 3.2) can be cast as a generalized Gaussian process model (Equation 3.6). Here, we will show that the Laplace approximation to this model yields a Gaussian process regression model. Essentially, this allows us to do the Laplace-GGN in the function-space instead of the parameter-space. Therefore, we can trade off computational costs between dimensionality P and dataset size N [42]. This holds for the posterior, posterior predictive, and the marginal likelihood. Therefore, the Laplace-GGN

is the first method that uses Gaussian process inference for finite width neural networks [17, 25]. The relationship to exact Gaussian process regression presented in this section allows to use efficient standard routines developed for these models [42].

We denote the Laplace-GGN posterior approximation in function-space by $q(\boldsymbol{f}_{\text{GP}})$. In the following we show that it can be simply computed by exact Gaussian process inference. Rasmussen [42] derives the Laplace approximation to a Gaussian process model for classification likelihoods. We denote the neural network linearized after the link function by

$$\boldsymbol{g}_{\text{lin}}^{-1}(\boldsymbol{x}_i;\boldsymbol{\theta}) = \boldsymbol{g}^{-1}(\boldsymbol{f}(\boldsymbol{x};\boldsymbol{\theta}_*)) + \boldsymbol{\Lambda}(\boldsymbol{x})\boldsymbol{J}(\boldsymbol{x})(\boldsymbol{\theta}-\boldsymbol{\theta}_*). \tag{3.8}$$

Using this notation, we can relate the Laplace approximation to the generalized Gaussian process:

Theorem 3.2. *We define the mean and covariance function of a GP prior $\hat{\boldsymbol{f}}_{GP} \sim \mathcal{GP}(\hat{\boldsymbol{m}}(\boldsymbol{x}), \hat{\boldsymbol{\kappa}}(\boldsymbol{x},\boldsymbol{x}'))$*

$$\hat{\boldsymbol{m}}(\boldsymbol{x}) = \boldsymbol{g}_{lin}^{-1}(\boldsymbol{x};\boldsymbol{\mu}_0) \quad \textit{and} \quad \hat{\boldsymbol{\kappa}}(\boldsymbol{x},\boldsymbol{x}') = \boldsymbol{\Lambda}(\boldsymbol{x})\boldsymbol{J}(\boldsymbol{x})^\top \boldsymbol{\Sigma}_0 \boldsymbol{J}(\boldsymbol{x}')\boldsymbol{\Lambda}(\boldsymbol{x}'), \tag{3.9}$$

which gives rise to the prior $\hat{p}(\hat{\boldsymbol{f}}_{GP})$. Then, the Laplace approximation to the generalized Gaussian process model in Equation 3.6 at $\boldsymbol{f}(\boldsymbol{x};\boldsymbol{\theta}_*)$ is equal to the posterior of the following Gaussian process regression model:

$$q(\boldsymbol{f}_{GP}) = \hat{p}(\hat{\boldsymbol{f}}_{GP}|\mathcal{D}) \propto \hat{p}(\hat{\boldsymbol{f}}_{GP}) \prod_{i=1}^{N} \mathcal{N}(\boldsymbol{y}_i|\hat{\boldsymbol{f}}_{GP}, \boldsymbol{\Lambda}(\boldsymbol{x}_i)). \tag{3.10}$$

This model complements the Bayesian linear regression model in Theorem 3.1 and has the same marginal likelihood and posterior predictive [42].

Proof. In correspondence with section 3.2, we conduct a Laplace approximation at $\boldsymbol{f}_* := \boldsymbol{f}(\boldsymbol{x};\boldsymbol{\theta}_*)$. Again, the prior is Gaussian and therefore remains unchanged. For the likelihood of an arbitrary label $\boldsymbol{y}$, we have

$$\begin{aligned}
\log p(\boldsymbol{y}|\boldsymbol{f}_{\text{GP}}) &\approx \log p(\boldsymbol{y}|\boldsymbol{f}_*) - (\boldsymbol{g}^{-1}(\boldsymbol{f}_*) - \boldsymbol{y})^\top (\boldsymbol{f}_{\text{GP}} - \boldsymbol{f}_*) - \frac{1}{2}(\boldsymbol{f}_{\text{GP}} - \boldsymbol{f}_*)^\top \boldsymbol{\Lambda} (\boldsymbol{f}_{\text{GP}} - \boldsymbol{f}_*) \\
&= \log p(\boldsymbol{y}|\boldsymbol{f}_*) + \frac{1}{2}(\boldsymbol{g}^{-1}(\boldsymbol{f}_*) - \boldsymbol{y})^\top \boldsymbol{\Lambda}^{-1} (\boldsymbol{g}^{-1}(\boldsymbol{f}_*) - \boldsymbol{y}) \\
&\quad - \frac{1}{2}(\boldsymbol{g}^{-1}(\boldsymbol{f}_*) + \boldsymbol{\Lambda}\boldsymbol{f}_{\text{GP}} - \boldsymbol{\Lambda}\boldsymbol{f}_* - \boldsymbol{y}) \boldsymbol{\Lambda}^{-1} (\boldsymbol{g}^{-1}(\boldsymbol{f}_*) + \boldsymbol{\Lambda}\boldsymbol{f}_{\text{GP}} - \boldsymbol{\Lambda}\boldsymbol{f}_* - \boldsymbol{y}),
\end{aligned}$$

where we expanded around $\boldsymbol{f}_*$ and completed the square. The first two summands of the last term are independent of $\boldsymbol{f}_{\text{GP}}$ and the last term yields a Gaussian likelihood as in the proof of Theorem 3.1. This allows to write a Gaussian likelihood. Again, the remaining constants are necessary for the computation of the marginal likelihood. □

The kernel in this model resembles the *neural tangent kernel* which arises when analyzing neural network training in function space [17]. To see this, we can reparameterize the GP as

$\boldsymbol{\Lambda}(\boldsymbol{x})\hat{\boldsymbol{f}}_{\text{GP}}$ and obtain the kernel $\hat{\boldsymbol{\kappa}}(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{J}(\boldsymbol{x})^\top \boldsymbol{\Sigma}_0 \boldsymbol{J}(\boldsymbol{x}')$. With a spherical prior covariance $\boldsymbol{\Sigma}_0$, we recover the kernel of Jacot et al. [17]. In contrast, we deal with finite width networks in the Bayesian setting and can recover similar properties using the practical Laplace-GGN method presented here. On the practical side, Laplace-GGN in function-space enables inference corresponding to a full posterior covariance for neural networks with huge amounts of parameters but few data. In particular, Laplace-GGN in function-space has the computational complexity $\mathcal{O}(N^3K^3 + NPK)$ for inversion of the kernel and computation of the Jacobians. Potentially, one can approximate this kernel by a low-rank structure, e.g., the Nyström approximation [42], leading to novel approximate inference algorithms for neural networks. In contrast, the complexity of the corresponding Bayesian linear regression inference is $\mathcal{O}(P^3 + NPK)$. In the following, we will derive the posterior, predictive, and marginal likelihood of both models.

3.4 Computing the Laplace-GGN Posterior Approximation

We compute the Laplace-GGN approximation to the posterior distribution of the neural network model. To obtain the parameters of the Gaussian approximation $q(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, we make use of Theorem 3.1. Bayesian linear regression models have a closed form solution [2, 33]. The mean and covariance parameter of the Gaussian posterior $\hat{p}(\boldsymbol{\theta}|\mathcal{D})$ are given by

$$\boldsymbol{\Sigma} = \left(\boldsymbol{\Sigma}_0^{-1} + \sum_{i=1}^{N} \boldsymbol{J}(\boldsymbol{x}_i)\boldsymbol{\Lambda}(\boldsymbol{x}_i)\boldsymbol{J}(\boldsymbol{x}_i)^\top\right)^{-1}, \tag{3.11}$$

$$\boldsymbol{\mu} = \boldsymbol{\Sigma}\left(\boldsymbol{\Sigma}_0^{-1}\boldsymbol{\mu}_0 + \sum_{i=1}^{N} \boldsymbol{J}(\boldsymbol{x}_i)^\top (\boldsymbol{y}_i - \boldsymbol{g}^{-1}\left(\boldsymbol{f}(\boldsymbol{x}_i)\right)) + \boldsymbol{J}(\boldsymbol{x}_i)^\top \boldsymbol{\Lambda}(\boldsymbol{x}_i)\boldsymbol{J}(\boldsymbol{x}_i)\boldsymbol{\theta}_*\right). \tag{3.12}$$

The covariance is simply the inverse of the Hessian of the negative log joint distribution. At first, the mean seems more involved. However, assuming a local minimum, i.e. $\sum_{i=1}^{N} \boldsymbol{J}_*(\boldsymbol{x}_i)(\boldsymbol{y}_i - \boldsymbol{g}_*^{-1}(\boldsymbol{x}_i)) - \boldsymbol{\Sigma}_0^{-1}(\boldsymbol{\theta}_* - \boldsymbol{\mu}_0) = \boldsymbol{0}$, we obtain have $\boldsymbol{\mu} = \boldsymbol{\theta}_*$. In this case, we recover a Laplace approximation that is constructed at the MAP, which we did not need to assume.

Alternatively, we can use the posterior of the generalized Gaussian process and infer in function space. Recall the kernel from the beginning of this chapter in Equation 3.4. Due to the multiple outputs, our kernel maps to a matrix. The kernel $\boldsymbol{K} \in \mathbb{R}^{NK \times NK}$ consists of N $(K \times K)$ *submatrices* along both dimensions. The *submatrix* at the i-th position along the first and j-th position along the second axis is given by $\boldsymbol{\kappa}(\boldsymbol{x}_i, \boldsymbol{x}_j) \in \mathbb{R}^{K \times K}$. We further have the kernel $\boldsymbol{K}_{**} = \kappa(\boldsymbol{x}_*, \boldsymbol{x}_*)$ for data point $\boldsymbol{x}_*$ and the joint kernel with the training data $\boldsymbol{K}_{*n} \in \mathbb{R}^{K \times NK}$. The block-diagonal matrix $\boldsymbol{W} \in \mathbb{R}^{NK \times NK}$ is defined by N $(K \times K)$ blocks where the i-th block is the negative log likelihood Hessian $\boldsymbol{\Lambda}(\boldsymbol{x}_i)$. Assuming a stationary MAP estimate, we have the following distribution on $\boldsymbol{f}_{\text{GP}}$ due to the Laplace approximation:

$$\boldsymbol{f}_{\text{GP}}|\mathcal{D}, x_* \sim \mathcal{N}\left(\boldsymbol{f}(\boldsymbol{x}_*; \boldsymbol{\theta}_*), \boldsymbol{K}_{**} - \boldsymbol{K}_{*n}\left(\boldsymbol{K} + \boldsymbol{W}^{-1}\right)^{-1}\boldsymbol{K}_{*n}^\top\right). \tag{3.13}$$

3.5 Posterior Predictive Distributions

This section manifests how the generalized Gauss-Newton method and approximate inference change our underlying inference model. Due to the two steps depicted in Figure 3.16, we obtain three different posterior predictive models. The original model is a probabilistic neural network model. Due to the GGN, we obtain a generalized linear or Gaussian process model. Finally, the Laplace approximation corresponds to exact inference in a Bayesian linear or Gaussian process regression model. Subsequently, we specify all three (approximate) posterior predictive distributions. We predict the label $\boldsymbol{y}_*$ of a new data point $\boldsymbol{x}_*$ and therefore need to compute the predictive distribution $p(\boldsymbol{y}_*|\mathcal{D}, \boldsymbol{x}_*)$. All models have the same (approximate) posterior distribution but the posterior predictive is naturally different.

NN sampling: to make predictions with the neural network, we need to approximate the posterior predictive integral by sampling. With S Monte Carlo samples $\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_S \sim q(\boldsymbol{\theta})$, we have

$$p(\boldsymbol{y}_*|\mathcal{D}, \boldsymbol{x}_*) \approx \int p(\boldsymbol{y}_*|f(\boldsymbol{x}_*; \boldsymbol{\theta}))q(\boldsymbol{\theta})d\boldsymbol{\theta} \approx \frac{1}{S}\sum_{i=1}^{S} p(\boldsymbol{y}_*|f(\boldsymbol{x}_*; \boldsymbol{\theta}_s)). \tag{3.14}$$

GLM sampling: to predict using the generalized linear or Gaussian process model, we also need samples to approximate the posterior predictive integral unless we have a Gaussian likelihood. Again using S samples, we have

$$\tilde{p}(\boldsymbol{y}_*|\mathcal{D}, \boldsymbol{x}_*) \approx \int p(\boldsymbol{y}_*|\boldsymbol{f}_{\text{lin}}(\boldsymbol{x}_*; \boldsymbol{\theta}))q(\boldsymbol{\theta})d\boldsymbol{\theta} \approx \frac{1}{S}\sum_{i=1}^{S} p(\boldsymbol{y}_*|\boldsymbol{f}_{\text{lin}}^{\boldsymbol{\theta}_*}(\boldsymbol{x}_*; \boldsymbol{\theta}_s)). \tag{3.15}$$

BLR predictive: the Laplace-GGN leads to exact inference in a Bayesian linear regression model. The posterior predictive is available in a closed form but the distribution does not match the original likelihood. We have

$$\begin{aligned} \hat{p}(\boldsymbol{y}_*|\mathcal{D}, \boldsymbol{x}_*) &= \int \mathcal{N}(\boldsymbol{y}_*|\boldsymbol{g}_{\text{lin}}^{-1}(\boldsymbol{x}_*; \boldsymbol{\theta}), \boldsymbol{\Lambda}(\boldsymbol{x}_*))q(\boldsymbol{\theta})d\boldsymbol{\theta} \\ &= \mathcal{N}\left(\boldsymbol{g}_{\text{lin}}^{-1}(\boldsymbol{x}_*; \boldsymbol{\mu}), \boldsymbol{\Lambda}(\boldsymbol{x}_*)\boldsymbol{J}(\boldsymbol{x}_*)\boldsymbol{\Sigma}\boldsymbol{J}(\boldsymbol{x}_*)^\top\boldsymbol{\Lambda}(\boldsymbol{x}_*) + \boldsymbol{\Lambda}(\boldsymbol{x}_*)\right). \end{aligned} \tag{3.16}$$

At the MAP, the mean simply is the MAP prediction of the neural network with additional uncertainty. In the case of a Gaussian likelihood, the last step is not needed. For the other likelihoods, the Hessian $\boldsymbol{\Lambda}(\boldsymbol{x}_*)$ corresponds to the variance of the GLM response variable. For the BLR and GLM sampling predictive, we can equivalently use the Gaussian process view under use of the posterior Gaussian process introduced in the previous section.

Only the first posterior predictive approximation is used in practice [4, 19, 44, 55]. However for a Bayesian neural network with scalar Gaussian likelihood, Foong et al. [9] have recently shown empirically that the third version often works better. The development of this chapter

theoretically underline this result and extends it to other likelihoods. Applying the generalized Gauss-Newton approximation changes our underlying inference model to a generalized linear model. Therefore, this is the model we should predict with when using the Laplace-GGN. The closed form Bayesian linear regression version is the second choice since it maintains linearity. However, the likelihood of the original model is replace by a Gaussian. In chapter 5, we support this hypothesis experimentally and show that the first predictive can fail spectacularly.

3.6 Marginal Likelihood Approximation

In this section, we derive the Laplace-GGN marginal likelihood approximation. We can derive it from the Laplace approximation to the marginal likelihood of the generalized linear or Gaussian process model. Using the exact marginal likelihoods as given by the BLR and GP regression models, we only have to derive a *correction* term. In the proofs of Theorem 3.1 and 3.2, the second-order approximation to the likelihood is the same and we can use it to derive the following result:

Theorem 3.3. *Let $\hat{p}(\mathcal{D})$ be the marginal likelihood of the Bayesian linear or Gaussian process regression model (Theorem 3.1 and 3.2). Then, the Laplace approximation to the marginal likelihood of the generalized linear model denoted by $\tilde{p}(\mathcal{D})$ is given by*

$$\log \tilde{p}(\mathcal{D}) \approx \log \hat{p}(\mathcal{D}) + \sum_{i=1}^{N} \Big[\log p(\boldsymbol{y}_i | \boldsymbol{f}(\boldsymbol{x}_i; \boldsymbol{\theta}_*)) - \log \mathcal{N}(\boldsymbol{y}_i | \boldsymbol{g}^{-1}(\boldsymbol{f}(\boldsymbol{x}_i; \boldsymbol{\theta}_*)), \boldsymbol{\Lambda}(\boldsymbol{x}_i)) \Big]. \tag{3.17}$$

For the Gaussian likelihood, this leads to $\hat{p}(\mathcal{D})$ while other likelihoods maintain a correction term.

Proof. The Laplace approximation to the generalized linear and GP model marginal likelihood arises from a second-order approximation. We can simply start off from the proofs of Theorem 3.1 and 3.2. For the prior, a second order approximation of the log density is exact since it is a Gaussian. For the likelihood, we obtain the following term for a single input-output pair:

$$\begin{aligned} \log p(\boldsymbol{y}|\boldsymbol{f}) \approx \log p(\boldsymbol{y}|\boldsymbol{f}_*) &+ \frac{1}{2} \left(\boldsymbol{g}^{-1}(\boldsymbol{f}_*) - \boldsymbol{y} \right)^\top \boldsymbol{\Lambda}^{-1} \left(\boldsymbol{g}^{-1}(\boldsymbol{f}_*) - \boldsymbol{y} \right) \\ &- \frac{1}{2} \left(\boldsymbol{g}^{-1}(\boldsymbol{f}_*) + \boldsymbol{\Lambda} \boldsymbol{J} \left(\boldsymbol{\theta} - \boldsymbol{\theta}_* \right) - \boldsymbol{y} \right)^\top \boldsymbol{\Lambda}^{-1} \left(\boldsymbol{g}^{-1}(\boldsymbol{f}_*) + \boldsymbol{\Lambda} \boldsymbol{J} \left(\boldsymbol{\theta} - \boldsymbol{\theta}_* \right) - \boldsymbol{y} \right). \end{aligned}$$

Let us add and subtract the term $\frac{1}{2} \log \left((2\pi)^k \det \boldsymbol{\Lambda} \right)$ to obtain a Gaussian log likelihood in the second term. The second term becomes $-\log \mathcal{N}(\boldsymbol{y}|\boldsymbol{g}^{-1}(\boldsymbol{f}(\boldsymbol{x}; \boldsymbol{\theta}_*)), \boldsymbol{\Lambda}(\boldsymbol{x}))$. The last term is also properly normalized to a Gaussian. The first and second term are independent of the parameter $\boldsymbol{\theta}$ and give us the correction terms. The remaining term applied to the entire dataset and combined with the prior gives us the BLR or GP regression models defined earlier. We denote the marginal likelihood of these models as $\hat{p}(\mathcal{D})$. □

Having established this connection, we are only left with the computation of $\hat{p}(\mathcal{D})$. For the

term $\hat{p}(\mathcal{D})$, we have again two ways to compute it due to weight and function space equivalence. Estimating $\hat{p}(\mathcal{D})$ using the Bayesian linear regression model of Theorem 3.1 is widely known and there exist numerically robust implementations for it [2, 33]. We have

$$\log \hat{p}(\mathcal{D}) = \sum_{i=1}^{N} \log \mathcal{N}(\boldsymbol{y}_i | \boldsymbol{g}_{\boldsymbol{\mu}}^{-1}(\boldsymbol{x}_i)) - \frac{1}{2} \log \frac{\det \boldsymbol{\Sigma}_0}{\det \boldsymbol{\Sigma}} - \frac{1}{2} (\boldsymbol{\mu} - \boldsymbol{\mu}_0)^\top \boldsymbol{\Sigma}_0^{-1} (\boldsymbol{\mu} - \boldsymbol{\mu}_0) . \tag{3.18}$$

Alternatively, we can use the Gaussian process variant. The key difference lies in the computational complexity shifted into the number of samples N as opposed to the number of parameters P. In contrast to the kernel for the generalized GP in section 3.5, we have the kernel of the GP regression model: $\hat{\boldsymbol{K}} \in \mathbb{R}^{NK \times NK}$ consists of N^2 $(K \times K)$ submatrices. The submatrix at i-th position along the first dimension and j-th position along the second axis is given by $\hat{\boldsymbol{\kappa}}(\boldsymbol{x}_i, \boldsymbol{x}_j)$ defined in Theorem 3.2. $\boldsymbol{W} \in \mathbb{R}^{NK \times NK}$ is a block-diagonal matrix of the N Hessians $\boldsymbol{\Lambda}(\boldsymbol{x}_i)$ for $i \in [N]$. Lastly, the mean function $\boldsymbol{m} \in R^{NK}$ is a concatenation of the individual mean functions of the N data points and $\boldsymbol{y}$ a concatenation of the corresponding labels. Then, we have for the marginal likelihood of the GP regression model

$$\log \hat{p}(\mathcal{D}) = -\frac{1}{2} (\boldsymbol{m} - \boldsymbol{y})^\top (\boldsymbol{K} + \boldsymbol{\Lambda})^{-1} (\boldsymbol{m} - \boldsymbol{y}) - \frac{1}{2} \det [\boldsymbol{K} + \boldsymbol{\Lambda}] - \frac{NK}{2} \log 2\pi. \tag{3.19}$$

With a single output model, we recover the form derived by Rasmussen [42, Sec. 2].

3.7 The Prior as Regularizer and Distribution

In recent discussions, the prior in Bayesian neural network models and its obscure meaning have been criticized [11, 53]. Here, we discuss the role of the prior in a Bayesian deep learning algorithm. In particular, we discuss the role of the prior in an approximate as opposed to exact posterior.

This chapter shows that, in the case of the Laplace-GGN approximation, the prior acts in form of a distribution in a GLM and as a regularizer in the MAP objective. The regularizer impacts the learned feature map, i.e., Jacobian, as well as the linearization point that both give rise to the GLM that we infer. The GLM with fixed feature map and a Gaussian prior is simple to infer and has a unimodal posterior. While it might be complicated to pose a prior on neural network parameters, the Laplace-GGN approximation simplifies the model to a GLM where a simple Gaussian prior is very common and justified [33]. Therefore, it is not straightforward to criticize the priors used in Bayesian deep learning since the prior impacts an approximate and not exact posterior. For the Laplace-GGN, we rather have to ask if the MAP estimation and linearization at the MAP is reasonable. The error of the linearization can be quantified while MAP estimation forms the foundation of deep learning itself. If we find both to be reasonable, inference in a GLM using the Laplace approximation is a minor remaining problem.

Chapter 4

The Impact of the Generalized Gauss-Newton in Variational Inference

The Gaussian variational approximation is the direct competitor of the Laplace approximation: it is equally scalable and uses the same approximating family. The difference is that optimization and approximate inference steps are not sequential but combined into one *variational inference* algorithm that maximizes the ELBO (Equation 2.13). Nonetheless, we can disentangle the respective influence of the generalized Gauss-Newton method and approximate inference for each step of such an algorithm. This understanding again suggests different posterior predictive distributions, updates in function-space due to a Gaussian process formulation, and leads to the identification of a new algorithm. To maximize the ELBO, we make use of *natural gradient variational inference* (NGVI). NGVI uses the information geometry to improve convergence [1] and is responsible for recent successes in the field of Bayesian deep learning [19, 38, 55]. Most algorithms further rely on the generalized Gauss-Newton approximation. Zhang et al. [55] use a Kronecker-factored approximation and Khan et al. [19] use a diagonal approximation to the GGN. Here, we analyze the case of the full GGN approximation.

First, we specify the parameter updates of a natural gradient variational inference method for the GVA [18, 19]. A short derivation of NGVI for the GVA can be found in Appendix C. We introduce three algorithms derived from the NGVI update to a Gaussian posterior approximation. Two of these algorithms, *variational online generalized Gauss-Newton* (VOGGN) and *online generalized Gauss-Newton* (OGGN), have been introduced before [20]. Additionally, we derive a new algorithm, the *linearized Gaussian variational approximation* (LGVA) algorithm. It is derived as a compromise between VOGGN and OGGN. OGGN is similar to an iterative Laplace approximation as it does not sample. The difference between VOGGN and LGVA lies in the order of operations: VOGGN samples first and then uses the GGN while LGVA applies the GGN and then samples. An illustration of the order of operations of the three algorithms is given in Table 4.1. Finally, we show how the Gaussian variational approximation at every step of these NGVI algorithms can be cast as a Bayesian linear or Gaussian process regression problem.

Algorithm	Step 1	Step 2
VOGGN	sample $\boldsymbol{\theta}_1, .., \boldsymbol{\theta}_S \sim \mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$	S lin. networks $\boldsymbol{f}_{\text{lin}}^{\boldsymbol{\theta}_1}(\boldsymbol{x}; \boldsymbol{\theta}), \ldots, \boldsymbol{f}_{\text{lin}}^{\boldsymbol{\theta}_S}(\boldsymbol{x}; \boldsymbol{\theta})$
LGVA	1 lin. network $\boldsymbol{f}_{\text{lin}}^{\boldsymbol{\mu}_t}(\boldsymbol{x}; \boldsymbol{\theta})$	sample $\boldsymbol{\theta}_1, .., \boldsymbol{\theta}_S \sim \mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$
OGGN	1 lin. network $\boldsymbol{f}_{\text{lin}}^{\boldsymbol{\mu}_t}(\boldsymbol{x}; \boldsymbol{\theta})$	use the mean $\boldsymbol{\mu}_t$

Table 4.1 – Illustration of three variational generalized Gauss-Newton algorithms. The order of approximating the expectation by samples and applying the linearization of the GGN yields different algorithms. Sampling is part of the variational inference algorithm while linearization comes from the GGN. Only VOGGN samples many neural networks. OGGN is a crude version of LGVA and VOGGN since it uses only the mean instead of sampling.

4.1 Gaussian Natural Gradient Variational Inference

We denote by $q_t(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$ the Gaussian variational approximation to the posterior at iteration t. In this chapter, we will work with the natural parameters. For the Gaussian distribution, the natural parameters at iteration t are given by

$$\boldsymbol{\eta}_t^{(1)} = \boldsymbol{\Sigma}_t^{-1}\boldsymbol{\mu}_t \quad \text{and} \quad \boldsymbol{\eta}_t^{(2)} = -\frac{1}{2}\boldsymbol{\Sigma}_t^{-1}. \tag{4.1}$$

It is mathematically convenient to work with this parameterization for NGVI. NGVI updates the first and second natural parameter of the Gaussian as

$$\boldsymbol{\Sigma}_{t+1}^{-1}\boldsymbol{\mu}_{t+1} = (1-\gamma)\boldsymbol{\Sigma}_t^{-1}\boldsymbol{\mu}_t + \gamma\boldsymbol{\Sigma}_0^{-1}\boldsymbol{\mu}_0 + \gamma\left[\nabla_{\boldsymbol{\mu}}\mathbb{E}\left[\log p(\mathcal{D}|\boldsymbol{\theta})\right] - 2\nabla_{\boldsymbol{\Sigma}}\mathbb{E}\left[\log p(\mathcal{D}|\boldsymbol{\theta})\right]\boldsymbol{\mu}_t\right], \tag{4.2}$$

$$-\frac{1}{2}\boldsymbol{\Sigma}_{t+1}^{-1} = (1-\gamma)\left[-\frac{1}{2}\boldsymbol{\Sigma}_t^{-1}\right] + \gamma\left[-\frac{1}{2}\boldsymbol{\Sigma}_0^{-1}\right] + \gamma\nabla_{\boldsymbol{\Sigma}}\mathbb{E}\left[\log p(\mathcal{D}|\boldsymbol{\theta})\right], \tag{4.3}$$

where the expectation is taken over the posterior approximation q_t at iteration t. The update tells us that we combine the current posterior approximation q_t with the prior using a convex combination (usually $\gamma \leq 1$). The data dependency is only due to the gradients with respect to mean and covariance of the expected log likelihood terms. Due to the linearity of expectation, the expected log likelihood can be written as

$$\mathbb{E}\left[\log p(\mathcal{D}|\boldsymbol{\theta})\right] = \mathbb{E}\left[\sum_{i=1}^{N}\log p(\boldsymbol{y}_i|\boldsymbol{f}(\boldsymbol{x}_i;\boldsymbol{\theta}))\right] = \sum_{i=1}^{N}\mathbb{E}\left[\log p(\boldsymbol{y}_i|\boldsymbol{f}(\boldsymbol{x}_i;\boldsymbol{\theta}))\right], \tag{4.4}$$

which allows us to restrict ourselves to a single representative data pair $(\boldsymbol{x}, \boldsymbol{y})$.[1] Therefore, the problem reduces to estimation of the derivatives $\nabla_{\boldsymbol{\mu}}\mathbb{E}\left[\log p(\boldsymbol{y}|\boldsymbol{f}(\boldsymbol{x};\boldsymbol{\theta}))\right]$ and $\nabla_{\boldsymbol{\Sigma}}\mathbb{E}\left[\log p(\boldsymbol{y}|\boldsymbol{f}(\boldsymbol{x};\boldsymbol{\theta}))\right]$. To enable the computation of these gradients, we use the identities made popular by Opper and Archambeau [37] that we introduced in section 2.2 (see Equation 2.15 and 2.16). This allows to express the gradients of the expectation as the expectation of gradients for individual samples from q_t. To derive different Gaussian NGVI algorithms, different approximations to these deriva-

[1] In a practical scenario, one can use doubly-stochastic variational inference by sampling subsets of data to obtain an unbiased estimate[16].

tives have been proposed based on the GGN [19, 20, 55]. After introducing the VOGGN algorithm, we will add one more variant to this family of Gaussian NGVI algorithms. Recall that $\boldsymbol{f}_{\text{lin}}^{\boldsymbol{\theta}_*}(\boldsymbol{x};\boldsymbol{\theta})$ denotes the neural network function linearized at $\boldsymbol{\theta}_*$ due to the GGN. Further, we denote S Monte Carlo samples from the approximating distribution $q_t(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$ by $\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_S$.

In all the following derivations, we first apply the identity of Opper and Archambeau [37]. This allows us to estimate the derivative with respect to the mean and covariance by sampling individual gradients. Recall the equalities from section 2.2

$$\nabla_{\boldsymbol{\mu}} \mathbb{E}_{\boldsymbol{\theta}_s \sim \mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)} \left[\log p(\boldsymbol{y}|\boldsymbol{f}(\boldsymbol{x};\boldsymbol{\theta}_s))\right] = \mathbb{E}_{\boldsymbol{\theta}_s \sim \mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)} \left[\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{y}|\boldsymbol{f}(\boldsymbol{x};\boldsymbol{\theta}_s))\right] \tag{4.5}$$

$$\nabla_{\boldsymbol{\Sigma}} \mathbb{E}_{\boldsymbol{\theta}_s \sim \mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)} \left[\log p(\boldsymbol{y}|\boldsymbol{f}(\boldsymbol{x};\boldsymbol{\theta}_s))\right] = \frac{1}{2} \mathbb{E}_{\boldsymbol{\theta}_s \sim \mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)} \left[\nabla^2_{\boldsymbol{\theta}\boldsymbol{\theta}} \log p(\boldsymbol{y}|\boldsymbol{f}(\boldsymbol{x};\boldsymbol{\theta}_s))\right]. \tag{4.6}$$

All following algorithms vary only in their approximation to these expectations and the log likelihood or its gradient and Hessian. We now show three different variants.

4.2 Variational Online Generalized Gauss-Newton

The variational online generalized Gauss-Newton algorithm uses the GGN *after* sampling parameters from the approximating distribution. For the first and second derivative, we have

$$\mathbb{E}_{\boldsymbol{\theta}_s \sim \mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)} \left[\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{y}|\boldsymbol{f}(\boldsymbol{x};\boldsymbol{\theta}_s))\right] \approx \frac{1}{S} \sum_{i=1}^{S} \boldsymbol{J}(\boldsymbol{x};\boldsymbol{\theta}_s)^\top \boldsymbol{r}(\boldsymbol{y}, \boldsymbol{f}(\boldsymbol{x};\boldsymbol{\theta}_s)), \tag{4.7}$$

$$\begin{aligned}
\mathbb{E}_{\boldsymbol{\theta}_s \sim \mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)} \left[\nabla^2_{\boldsymbol{\theta}\boldsymbol{\theta}} \log p(\boldsymbol{y}|\boldsymbol{f}(\boldsymbol{x};\boldsymbol{\theta}_s))\right] &\approx \frac{1}{2} \sum_{i=1}^{S} \nabla^2_{\boldsymbol{\theta}\boldsymbol{\theta}} p(\boldsymbol{y}|\boldsymbol{f}(\boldsymbol{x};\boldsymbol{\theta}_s)) \\
&\approx \frac{1}{2} \sum_{i=1}^{S} \nabla^2_{\boldsymbol{\theta}\boldsymbol{\theta}} p(\boldsymbol{y}|\boldsymbol{f}_{\text{lin}}^{\boldsymbol{\theta}_s}(\boldsymbol{x};\boldsymbol{\theta}_s)) \\
&= -\frac{1}{2} \sum_{i=1}^{S} \boldsymbol{J}(\boldsymbol{x};\boldsymbol{\theta}_s)^\top \boldsymbol{\Lambda}(\boldsymbol{f}(\boldsymbol{x};\boldsymbol{\theta}_s)) \boldsymbol{J}(\boldsymbol{x};\boldsymbol{\theta}_s).
\end{aligned} \tag{4.8}$$

For the first derivative, simply approximate the expected gradient using S samples. For the Hessian, we first sample S neural network models and then linearize these models individually at the sampled parameters $\boldsymbol{\theta}_s$. We consider this expansion point as a constant and therefore can compute the Hessian of the linearized neural network log likelihood with respect to individual samples $\boldsymbol{\theta}_s$. This is like simultaneously sampling the linearization point and parameter. Similar algorithms proposed before, have used either a diagonal, low-rank, or Kronecker factored approximation to the Hessian [4, 19, 32, 55]. Next, we introduce a new algorithm that does not apply the GGN per sample but rather before sampling.

4.3 Linearized Gaussian Variational Inference

The linearized GVA applies the generalized Gauss-Newton approximation *before* sampling. That means, we linearize the neural network at some point $\boldsymbol{\theta}_*$ and then compute the gradients. Here, we choose to linearize at $\boldsymbol{\mu}_t$. Therefore, we have

$$\begin{aligned}\mathbb{E}_{\boldsymbol{\theta}_s\sim\mathcal{N}(\boldsymbol{\mu}_t,\boldsymbol{\Sigma}_t)}\left[\nabla_{\boldsymbol{\theta}}\log p(\boldsymbol{y}|\boldsymbol{f}(\boldsymbol{x};\boldsymbol{\theta}_s))\right] &\approx \mathbb{E}_{\boldsymbol{\theta}_s\sim\mathcal{N}(\boldsymbol{\mu}_t,\boldsymbol{\Sigma}_t)}\left[\nabla_{\boldsymbol{\theta}}\log p(\boldsymbol{y}|\boldsymbol{f}_{\text{lin}}^{\boldsymbol{\mu}_t}(\boldsymbol{x};\boldsymbol{\theta}_s))\right] \\ &= \frac{1}{S}\sum_{i=1}^{S}\boldsymbol{J}(\boldsymbol{x};\boldsymbol{\mu}_t)^\top \boldsymbol{r}(\boldsymbol{y},\boldsymbol{f}_{\text{lin}}^{\boldsymbol{\mu}_t}(\boldsymbol{x};\boldsymbol{\theta}_s)),\end{aligned} \tag{4.9}$$

$$\begin{aligned}\mathbb{E}_{\boldsymbol{\theta}_s\sim\mathcal{N}(\boldsymbol{\mu}_t,\boldsymbol{\Sigma}_t)}\left[\nabla^2_{\boldsymbol{\theta}\boldsymbol{\theta}}\log p(\boldsymbol{y}|\boldsymbol{f}(\boldsymbol{x};\boldsymbol{\theta}_s))\right] &\approx \mathbb{E}_{\boldsymbol{\theta}_s\sim\mathcal{N}(\boldsymbol{\mu}_t,\boldsymbol{\Sigma}_t)}\left[\nabla^2_{\boldsymbol{\theta}\boldsymbol{\theta}}\log p(\boldsymbol{y}|\boldsymbol{f}_{\text{lin}}^{\boldsymbol{\mu}_t}(\boldsymbol{x};\boldsymbol{\theta}_s))\right] \\ &= \mathbb{E}_{\boldsymbol{\theta}_s\sim\mathcal{N}(\boldsymbol{\mu}_t,\boldsymbol{\Sigma}_t)}\left[-\boldsymbol{J}(\boldsymbol{x};\boldsymbol{\mu}_t)^\top\boldsymbol{\Lambda}(\boldsymbol{f}_{\text{lin}}^{\boldsymbol{\mu}_t}(\boldsymbol{x};\boldsymbol{\theta}_s))\boldsymbol{J}(\boldsymbol{x};\boldsymbol{\mu}_t)\right] \\ &= -\frac{1}{S}\sum_{i=1}^{S}\boldsymbol{J}(\boldsymbol{x};\boldsymbol{\mu}_t)^\top\boldsymbol{\Lambda}(\boldsymbol{f}_{\text{lin}}^{\boldsymbol{\mu}_t}(\boldsymbol{x};\boldsymbol{\theta}_s))\boldsymbol{J}(\boldsymbol{x};\boldsymbol{\mu}_t).\end{aligned} \tag{4.10}$$

In contrast to VOGGN, we only sample in the first order of the neural network. LGVA has two potential advantages over VOGGN: we need to compute only one Jacobian no matter how many samples we take and linearization might stabilize the training. LGVA can be seen the variational twin of the Laplace-GGN since it constructs a GLM in each step and not only at the MAP. In this GLM, we take a step of natural gradient variational inference. Therefore, we should predict with this model using the GLM sampling method. The reason is easy to see: in the above derivation, we only work with the linearized neural network.

4.4 Deterministic Variational Online Gauss-Newton

The last algorithm we introduce is called online generalized Gauss-Newton (OGGN). It is motivated as a deep learning optimizer derived from natural gradient variational inference [20]. Instead of sampling to compute expectations, we simply take the current mean $\boldsymbol{\mu}_t$. Therefore, this method is related to the Laplace approximation.We have the derivatives

$$\begin{aligned}\mathbb{E}_{\boldsymbol{\theta}_s\sim\mathcal{N}(\boldsymbol{\mu}_t,\boldsymbol{\Sigma}_t)}\left[\nabla_{\boldsymbol{\theta}}\log p(\boldsymbol{y}|\boldsymbol{f}(\boldsymbol{x};\boldsymbol{\theta}_s))\right] &\approx \nabla_{\boldsymbol{\mu}}\log p(\boldsymbol{y}|\boldsymbol{f}(\boldsymbol{x};\boldsymbol{\mu}_t)) \\ &= \boldsymbol{J}(\boldsymbol{x};\boldsymbol{\mu}_t)^\top\boldsymbol{r}(\boldsymbol{y},\boldsymbol{f}(\boldsymbol{x};\boldsymbol{\mu}_t),\end{aligned} \tag{4.11}$$

$$\begin{aligned}\mathbb{E}_{\boldsymbol{\theta}_s\sim\mathcal{N}(\boldsymbol{\mu}_t,\boldsymbol{\Sigma}_t)}\left[\nabla^2_{\boldsymbol{\theta}\boldsymbol{\theta}}\log p(\boldsymbol{y}|\boldsymbol{f}(\boldsymbol{x};\boldsymbol{\theta}_s))\right] &\approx \nabla^2_{\boldsymbol{\mu}\boldsymbol{\mu}}\log p(\boldsymbol{y}|\boldsymbol{f}(\boldsymbol{x};\boldsymbol{\mu}_t)) \\ &\approx -\boldsymbol{J}(\boldsymbol{x};\boldsymbol{\mu}_t)^\top\boldsymbol{\Lambda}(\boldsymbol{f}(\boldsymbol{x};\boldsymbol{\mu}_t))\boldsymbol{J}(\boldsymbol{x};\boldsymbol{\mu}_t),\end{aligned} \tag{4.12}$$

where we approximate the expectation using the mean. For the second derivative, we use the GGN to get the last line. Note also that we can obtain above algorithm starting from LGVA and using the mean $\boldsymbol{\mu}_t$ instead of sampling parameters $\boldsymbol{\theta}_s$.

Algorithm	samples	$\hat{\boldsymbol{f}}_s(\boldsymbol{x})$	$\hat{\boldsymbol{J}}_s(\boldsymbol{x})$
VOGGN	S	$\boldsymbol{f}(\boldsymbol{x};\boldsymbol{\theta}_s)$	$\boldsymbol{J}(\boldsymbol{x};\boldsymbol{\theta}_s)$
LGVA	S	$\boldsymbol{f}_{\text{lin}}^{\boldsymbol{\mu}_t}(\boldsymbol{x};\boldsymbol{\theta}_s)$	$\boldsymbol{J}(\boldsymbol{x};\boldsymbol{\mu}_t)$
OGGN	1	$\boldsymbol{f}_{\text{lin}}^{\boldsymbol{\mu}_t}(\boldsymbol{x};\boldsymbol{\mu}_t) = \boldsymbol{f}(\boldsymbol{x};\boldsymbol{\mu}_t)$	$\boldsymbol{J}(\boldsymbol{x};\boldsymbol{\mu}_t)$

Table 4.2 – Values of the parameters in Theorem 4.1 for the three algorithms. $\boldsymbol{\theta}_s$ is a sample from the posterior approximation q_t at iteration t and $\boldsymbol{\mu}_t$ its mean. Only VOGGN samples S neural networks and obtains individual Jacobians.

4.5 Variational GGN Iterations as Exact Inference

In line with the proofs for the Laplace-GGN approximation, we will show that all above NGVI algorithms solve local Bayesian linear regression models. Again, we can interpret these models in the function-space and characterize them as Gaussian processes. This analysis also explains why VOGGN is the most powerful algorithm and can be expected to predict well when we sample from the neural network. LGVA and OGGN are therefore expected to require linearization to predict accurately.

The natural parameter updates of the Gaussian variational approximation in Equation 4.2 and 4.3 can simply be written into the Gaussian posterior approximation q_{t+1} by multiplying with the Gaussian sufficient statistics (see Appendix C). The first step is to combine the prior and the posterior approximation at step t to obtain an intermediary prior. We take the terms independent of the data from the natural parameter updates and define the natural parameters of the Gaussian $p_t(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{m}, \boldsymbol{S})$ as $\eta^{(1)} = (1-\gamma)\boldsymbol{\Sigma}_t^{-1}\boldsymbol{\mu}_t + \gamma\boldsymbol{\Sigma}_0^{-1}\boldsymbol{\mu}_0$ and $\eta^{(2)} = -\frac{1}{2}\left[(1-\gamma)\boldsymbol{\Sigma}_t^{-1} + \gamma\boldsymbol{\Sigma}_0^{-1}\right]$. Resolving the data-dependent term requires more steps and is shown in the proof of the following theorem.

Theorem 4.1. *The VOGGN, LGVA, and OGGN algorithms perform exact Bayesian linear regression in each update. In the most general case, we can characterize the updated posterior approximation $q_{t+1}(\boldsymbol{\theta})$ as*

$$q_{t+1}(\boldsymbol{\theta}) \propto p_t(\boldsymbol{\theta}) \prod_{i=1}^{N} \prod_{i=1}^{S} \mathcal{N}\left(\boldsymbol{y}_i \middle| \boldsymbol{g}^{-1}(\hat{\boldsymbol{f}}_s(\boldsymbol{x}_i)) + \boldsymbol{\Lambda}(\hat{\boldsymbol{f}}_s(\boldsymbol{x}_i))\hat{\boldsymbol{J}}_s(\boldsymbol{x}_i)(\boldsymbol{\theta} - \boldsymbol{\mu}_t), \frac{S}{\gamma}\boldsymbol{\Lambda}(\hat{\boldsymbol{f}}_s(\boldsymbol{x}_i))\right), \tag{4.13}$$

where S is the number of Monte Carlo samples, γ the step size and the function and Jacobian values $\hat{\boldsymbol{f}}_s$ and $\hat{\boldsymbol{J}}_s$ depend on the algorithm. For the particular values, see Table 4.2. The key difference to Theorem 3.1 lies in the fact that we sample $\boldsymbol{\theta}_s$ from $q_t(\boldsymbol{\theta})$ as opposed to taking the mean. For the overdispersed Gaussian likelihood, this also holds but needs to be written slightly differently since $\boldsymbol{\Lambda}(\hat{\boldsymbol{f}}_s(\boldsymbol{x}))$ does not correspond to the noise variance (see end of proof below).

Proof. The prior $p_t(\boldsymbol{\theta})$ arises from the natural parameter update. Additionally, we plug the data-dependent terms into the Gaussian natural parameterization (cf. section 2.1 and Appendix C). In particular, we need to plug into $e^{\gamma\langle T(\boldsymbol{\theta}), \widetilde{\boldsymbol{\eta}}\rangle}$ where $\widetilde{\boldsymbol{\eta}}$ denotes the data-dependent natural parameter

summands of Equation 4.2 and 4.3 as follows

$$e^{\gamma\langle T(\boldsymbol{\theta}),\widetilde{\boldsymbol{\eta}}\rangle} = \exp\left\{\gamma\langle\boldsymbol{\theta}, \nabla_{\boldsymbol{\mu}}\mathbb{E}\left[\log p(\mathcal{D}|\boldsymbol{\theta})\right] - 2\nabla_{\boldsymbol{\Sigma}}\mathbb{E}\left[\log p(\mathcal{D}|\boldsymbol{\theta})\right]\boldsymbol{\mu}_t\rangle + \gamma\langle\boldsymbol{\theta}\boldsymbol{\theta}^\top, \nabla_{\boldsymbol{\Sigma}}\mathbb{E}\left[\log p(\mathcal{D}|\boldsymbol{\theta})\right]\rangle\right\}.$$

Next, we can use the linearity of expectation and write $\mathbb{E}\left[\log p(\mathcal{D}|\boldsymbol{\theta})\right]$ as a sum over the N data points. Since the inner product is linear, we can pull the sum outside of the exponent and get a product over N data points instead:

$$\prod_{i=1}^{N}\exp\left\{\gamma\langle\boldsymbol{\theta}, \nabla_{\boldsymbol{\mu}}\mathbb{E}\left[\log p(\boldsymbol{y}_i|\boldsymbol{\theta})\right] - 2\nabla_{\boldsymbol{\Sigma}}\mathbb{E}\left[\log p(\boldsymbol{y}_i|\boldsymbol{\theta})\boldsymbol{\mu}_t\right]\rangle + \gamma\langle\boldsymbol{\theta}\boldsymbol{\theta}^\top, \nabla_{\boldsymbol{\Sigma}}\mathbb{E}\left[\log p(\boldsymbol{y}_i|\boldsymbol{\theta})\right]\rangle\right\},$$

where we have abbreviated $\log p(\boldsymbol{y}|\boldsymbol{f}(\boldsymbol{x};\boldsymbol{\theta}))$ as $\log p(\boldsymbol{y}|\boldsymbol{\theta})$. Taking S samples $\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_S \sim q_t(\boldsymbol{\theta})$ leads to a sum over S divided by S. The sum can again be pulled outside to obtain a product over these samples and we pull the gradient inside the expectation to obtain

$$\prod_{i=1}^{N}\prod_{s=1}^{S}\exp\left\{\frac{\gamma}{S}\langle\boldsymbol{\theta}, \nabla_{\boldsymbol{\theta}}\log p(\boldsymbol{y}_i|\boldsymbol{\theta}_s) - \nabla^2_{\boldsymbol{\theta}\boldsymbol{\theta}}\log p(\boldsymbol{y}_i|\boldsymbol{\theta}_s)\boldsymbol{\mu}_t\rangle + \frac{\gamma}{2S}\langle\boldsymbol{\theta}\boldsymbol{\theta}^\top, \nabla^2_{\boldsymbol{\theta}\boldsymbol{\theta}}\log p(\boldsymbol{y}_i|\boldsymbol{\theta}_s)\rangle\right\}.$$

We continue with the exponent for a single data and MC sample. We use $\hat{f}_s(\boldsymbol{x})$ for the function and $\hat{\boldsymbol{J}}_s(\boldsymbol{x})$ for the Jacobian for some data point $(\boldsymbol{x}, \boldsymbol{y})$ and parameter sample $\boldsymbol{\theta}_s$. For brevity, we write $\hat{\boldsymbol{\Lambda}}_s := \boldsymbol{\Lambda}(\hat{\boldsymbol{f}}_s(\boldsymbol{x}))$. Then, we have for a single exponent

$$\begin{aligned}
&\frac{\gamma}{S}\langle\boldsymbol{\theta}, \boldsymbol{J}_s(\boldsymbol{x})^\top\boldsymbol{r}(\boldsymbol{y}, \hat{\boldsymbol{f}}_s(\boldsymbol{x})) + \boldsymbol{J}_s(\boldsymbol{x})^\top\hat{\boldsymbol{\Lambda}}_s\boldsymbol{J}_s(\boldsymbol{x})\boldsymbol{\mu}_t\rangle - \frac{\gamma}{2S}\langle\boldsymbol{\theta}\boldsymbol{\theta}^\top, \boldsymbol{J}_s(\boldsymbol{x})^\top\hat{\boldsymbol{\Lambda}}_s\boldsymbol{J}_s(\boldsymbol{x})\rangle \\
=&\frac{\gamma}{S}\boldsymbol{\theta}^\top\boldsymbol{J}_s(\boldsymbol{x})^\top\left(\boldsymbol{y} - \boldsymbol{g}^{-1}(\boldsymbol{f}_s(\boldsymbol{x})) + \hat{\boldsymbol{\Lambda}}_s\boldsymbol{J}_s(\boldsymbol{x})\boldsymbol{\mu}_t\right) - \frac{\gamma}{2S}\boldsymbol{\theta}^\top\boldsymbol{J}_s(\boldsymbol{x})^\top\hat{\boldsymbol{\Lambda}}_s\boldsymbol{J}_s(\boldsymbol{x})\boldsymbol{\theta} \\
=&-\frac{1}{2}\left(\boldsymbol{g}^{-1}(\hat{\boldsymbol{f}}_s(\boldsymbol{x})) + \hat{\boldsymbol{\Lambda}}_s\boldsymbol{J}_s(\boldsymbol{x})\left(\boldsymbol{\theta} - \boldsymbol{\mu}_t\right) - \boldsymbol{y}\right)\left(\frac{S}{\gamma}\hat{\boldsymbol{\Lambda}}_s\right)^{-1}\left(\boldsymbol{g}^{-1}(\hat{\boldsymbol{f}}_s(\boldsymbol{x})) + \hat{\boldsymbol{\Lambda}}_s\boldsymbol{J}_s(\boldsymbol{x})\left(\boldsymbol{\theta} - \boldsymbol{\mu}_t\right) - \boldsymbol{y}\right) \\
&+\frac{1}{2}\left(\boldsymbol{y} - \boldsymbol{g}^{-1}(\hat{\boldsymbol{f}}_s(\boldsymbol{x}))\right)^\top\left(\frac{S}{\gamma}\hat{\boldsymbol{\Lambda}}_s\right)^{-1}\left(\boldsymbol{y} - \boldsymbol{g}^{-1}(\hat{\boldsymbol{f}}_s(\boldsymbol{x}))\right)
\end{aligned}$$

where we first used the inner product properties and then completed the square. The first term in the individual exponent yields a Gaussian density with the desired structure and therefore concludes the proof. Note that for OGGN, we do not sample so the proof is simpler but follows the same steps. For the overdispersed Gaussian likelihood, we can set $\hat{\boldsymbol{\Lambda}}_s = \boldsymbol{I}_K$ above and then divide all terms by the dispersion parameter, i.e., variance. That is only possible because both residual and Hessian are scaled by σ^{-2} (see. Table 2.1). Then, we maintain a Gaussian likelihood. □

In comparison to Theorem 3.1 obtained for the Laplace-GGN approximation, this theorem characterizes the steps of an approximate inference algorithm as opposed to the stationary point. Therefore, we additionally have the step size γ in our model. The deterministic OGGN algorithm ($S = 1$) is similar to the Laplace-GGN approximation. This is apparent if we set the step size γ of OGGN to 1 at a stationary point: The linear regression model in Theorem 3.1 and Theorem 4.1 become equivalent. OGGN has the advantage that it is an online algorithm and

provides a posterior approximation in every step and not only at a MAP estimate. All derived quantities in chapter 3 can also be applied to the OGGN posterior approximation: in particular, we should predict using GLM sampling in Equation 3.15.

The Bayesian linear regression model corresponding to VOGGN and LGVA varies significantly due to the S samples. Each sample *augments* the model with N new predictive models that always predict the same N targets $\boldsymbol{y}$ from observations $\boldsymbol{x}$. In section 2.2, we have characterized the stationarity of the Gaussian variational approximation, which depends on an expectation. Here, we have taken samples to approximate this expectation and observe that this indeed leads to a more *global* characterization due to an augmented linear regression model. Therefore, comparing OGGN with VOGGN and LGVA is similar to the relation between Laplace and Gaussian variational approximation presented in section 2.2 but for iterations instead of stationarity. Following the developments of chapter 3, we can equivalently turn above Bayesian linear regression model into a Gaussian process regression model with a kernel of size $NKS \times NKS$. Potentially, these models are useful to identify a good step size γ and sample-size S since these parameters are represented in the model.

4.6 Comparison and Posterior Predictive Computation

In chapter 3, we have introduced three ways to compute the posterior predictive of the Bayesian neural network model. Based on the derivation of the algorithms and their relation to a Bayesian linear Regression model, we can argue for the right choice of posterior predictive. LGVA linearizes the neural network *before* taking samples and can therefore be also characterized as a GLM in each inference step. This can be seen in the updates of LGVA (Equation 4.9 and 4.10) where we use a linearized neural network for the log likelihood and therefore have a GLM. Since OGGN can be derived from LGVA by a crude approximation of the expectation, the same argument holds for OGGN. Further, OGGN is tightly connected to the Laplace approximation for which we proposed to use the GLM sampling predictive. Therefore, both LGVA and OGGN should make use of the GLM sampling predictive.

VOGGN works very differently from both LGVA and OGGN since we first sample and then linearize. That means, during update steps we sample non-linear neural networks and then linearize them individually at the sampled parameter. This allows the computation of the Hessian approximation for different neural networks in Equation 4.8. Therefore, we obtain S different Jacobians instead of a single one as in LGVA. In the predictive setting, this would allow to use the NN sampling method. Notably, VOGGN is the only approximate inference algorithm where we can arguably expect the NN sampling method to work well. Nonetheless, the NN sampling method is the only one used for prediction with Bayesian neural networks in the past.

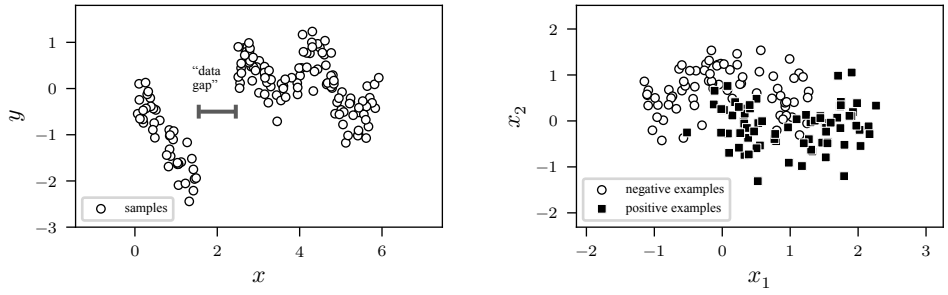
Chapter 5

Experiments

In this chapter, we investigate the behavior of the analyzed and proposed algorithms and validate our hypotheses. One of the key propositions of this work is that the computation of the predictive distribution should be aligned with the inference algorithm. Therefore, we investigate experimentally how the combination of approximate inference and the generalized Gauss-Newton method impacts the posterior predictive. Further, we use the identified generalized linear and Gaussian process models to approximate the marginal likelihood of a neural network, and use the Gaussian process posterior predictive for explainability. We conduct our experiments on toy regression and classification datasets that allow detailed visualizations. For the explainability experiment, we use a binary handwritten digit classification task.

In Figure 5.1, we illustrate both datasets $\mathcal{D}$ each with $N = 150$ data points. The two-dimensional classification problem is known as "two moons" [40]. Here, we have inputs $\boldsymbol{x}_i \in \mathbb{R}^2$ and targets $y_i \in \{0, 1\}$. The one-dimensional regression task is known as "Snelson" named after its inventor [48]. In this case, we have inputs and outputs $x_i, y_i \in \mathbb{R}$. We further add an artificial gap in this data set to make it more complicated and observe overfitting in line with [9, 20]. Both datasets are standard toy problems to benchmark non-linear models like Gaussian processes and neural networks. For both the regression and classification dataset, we proceed as follows: first, we select our models using the marginal likelihood. That means, we find appropriate parameters for the prior and, in the regression example, for the likelihood. Next, we compare the three posterior predictive distributions for the different inference algorithms (cf. section 3.4). Finally, we make use of the Gaussian process model to understand the predictions of our models.

For both tasks, we use a standard *multilayer perceptron* with 5 layers and 25 hidden units per layer and the `tanh` activation function. All layers have bias parameters. We have parameter vectors $\boldsymbol{\theta} \in \mathbb{R}^P$ with $P = 2676$ for the 1-D regression task and $P = 2701$ for the 2-D classification task. The parameter vectors define our neural network function $f(\boldsymbol{x}; \boldsymbol{\theta})$ mapping inputs to outputs as depicted in Figure 2.1. For the binary classification task, we use a Bernoulli likelihood, i.e., we model $Y \sim \text{Bernoulli}(f(\boldsymbol{x}; \boldsymbol{\theta}))$ where the neural network f parameterizes the natural



(a) Regression dataset. (b) Classification dataset.

Figure 5.1 – Visualization of the two toy example datasets used in the experimental study. Figure (a) shows the univariate regression dataset Snelson [48] with an additional "data gap". Figure (b) shows the two moons classification dataset that has a noisy decision boundary and can therefore lead to severe overfitting.

parameter (cf. Table 2.1). In the regression case, we use a Gaussian likelihood with dispersion parameter σ^2, i.e., we model the response $Y \sim \mathcal{N}(f(\boldsymbol{x};\boldsymbol{\theta}),\sigma^2)$. In line with the literature, we use a spherical Gaussian prior with precision δ on the parameters, i.e., $\boldsymbol{\theta} \sim \mathcal{N}(\mathbf{0},\delta^{-1}\boldsymbol{I}_P)$. Using the marginal likelihood, we can then optimize the hyperparameter δ for both problems. In the regression case, we additionally have the observation noise σ^2 as hyperparameter.

5.1 Model Selection Using Marginal Likelihood

We use the Laplace-GGN algorithm introduced in Chapter 3 to compute an approximation to the marginal likelihood. The marginal likelihood gives evidence to prefer one model over another. Therefore, it allows us to find suitable hyperparameters δ and σ^2. This procedure is called *empirical Bayes*. Empirical Bayes is uncommon for neural networks and usually cross-validation schemes are applied [12], even in Bayesian deep learning [19, 55]. While empirical Bayes is uncommon for neural networks, it has been explored before in the context of Laplace and Gaussian variational approximations [10, 54]. In particular, the method of Foresee and Hagan [10] is the same as the one presented here in the case of a Gaussian likelihood.

For both datasets, we have $N = 150$ training samples and a test set $\mathcal{D}_{\text{test}}$ with 1000 additional input output pairs. This allows us to estimate the generalization error. For the generalization error, we use the average log likelihood on the test data at the MAP. Let p be the likelihood of the corresponding model. Then, we have for the average log likelihood at the MAP

$$\ell\ell = \frac{1}{|\mathcal{D}_{\text{test}}|} \sum_{(\boldsymbol{x}_i,y_i)\in\mathcal{D}_{\text{test}}} \log p(y_i|f(\boldsymbol{x}_i;\boldsymbol{\theta}_{\text{MAP}})), \tag{5.1}$$

where $\boldsymbol{x}_i$ is a scalar in the regression dataset. Ideally, the marginal likelihood approximation

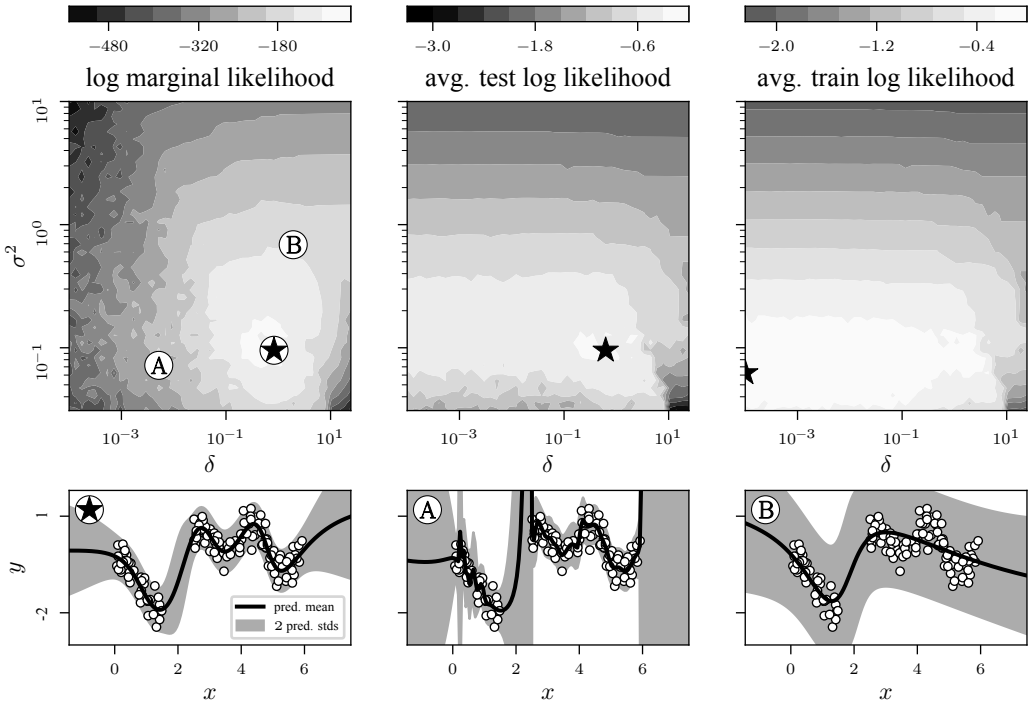


Figure 5.2 – Marginal likelihood with respect to observation noise σ^2 and prior precision δ of a neural network model on a toy regression problem with example posterior predictive distributions. In the top, the marginal likelihood approximation of a neural network model due to the Laplace-GGN in comparison to average test and train log likelihood is displayed. The marginal likelihood provides a robust way to choose hyperparameters and is in line with the test log likelihood. In contrast, the optimal likelihood on the training data can go to zero which leads to an overfitting model. In the bottom, the optimal posterior predictive due to the marginal likelihood ($\star$) and examples of overfitting (A) and underfitting (B) are visualized. We show the posterior predictive mean and two standard deviations.

suggests the same optimal parameters as the test log likelihood. In the regression problem, we choose the hyperparameter ranges $\sigma^2 \in [0.001, 10]$ and $\delta \in [0.0001, 100]$. For classification, we choose $\delta \in [0.01, 100]$. Using this range of hyperparameters, we train the one neural networks for each parameter setting until convergence to a MAP estimate. For training the MAP objective, we use the Adam optimizer [21]. At the MAP, we compute the Laplace-GGN approximation to the marginal likelihood, i.e. , to the local generalized linear model. For the predictive distribution, we therefore also choose the GLM sampling variant.

Figure 5.2 shows the marginal likelihoods for different hyperparameters on the regression problem. The optimal hyperparameters found using the marginal likelihood are very close those that generalize the best according to the test log likelihood. In contrast, we can see that the neural network can become too expressive and overfit when we have weak regularization. Weak regularization corresponds to a small prior precision δ and can lead to a complex predictive func-

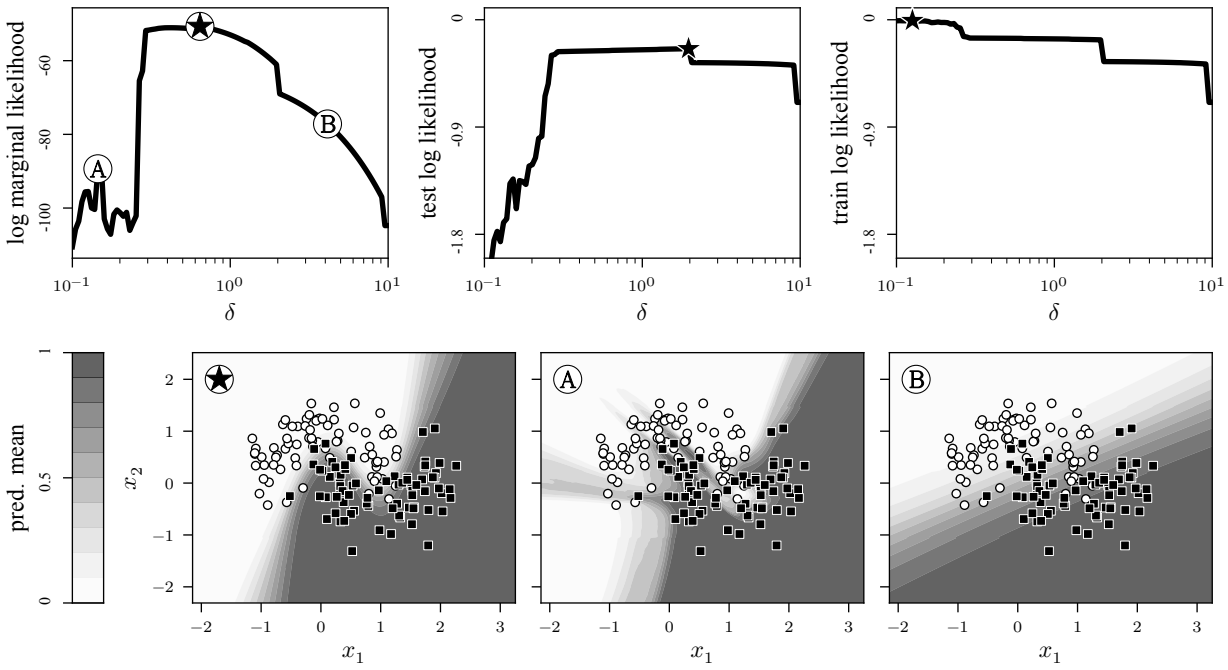


Figure 5.3 – Marginal likelihood with respect to prior precision δ of a neural network model for classification with examples of the corresponding predictive mean. In the top, we compare the marginal likelihood to the log likelihood on the testing and training dataset. The marginal likelihood identifies the optimal range of prior precision between 0.3 and 2 in line with the test log likelihood. In the figure displaying the log likelihood on the training data, the model overfits clearly for small prior precision values. In the bottom, the mean of the posterior predictive due to sampling from the generalized linear model is visualized for the model with optimal marginal likelihood ($\star$) and an overfitting (A) and underfitting (B) model. The overfitting model achieves almost zero misclassifications in the training data due to an overly complex decision boundary.

tion that overfits to individual training data points. Figure 5.2 further depicts three generalized linear model predictives: the optimal, an overfitting, and an underfitting model. We see that the optimal model also visually trades off between complexity and simplicity while the overfitting model is overly complex and fits the noise. In contrast, the underfitting model fails to match the shape of the underlying data generating function. We identify the optimal hyperparameters $\delta = 0.63$ and $\sigma^2 = 0.1$. Notably, the observations are generated with a similar noise variance of 0.09.

In Figure 5.3, we conduct the same analysis for the classification problem and obtain similar results: According to the marginal likelihood, the range of optimal hyperparameters lies between 0.02 and 2 which matches the plateau of the test log likelihood accurately. The marginal likelihood identifies the optimal hyperparameter $\delta = 0.13$. The training log likelihood goes to zero for weak regularization indicating a perfect fit and correct prediction of each training data point. However, both the marginal likelihood and test likelihood reject such an overfitting model. Figure 5.3 further displays such a model in comparison to the optimal model according to the

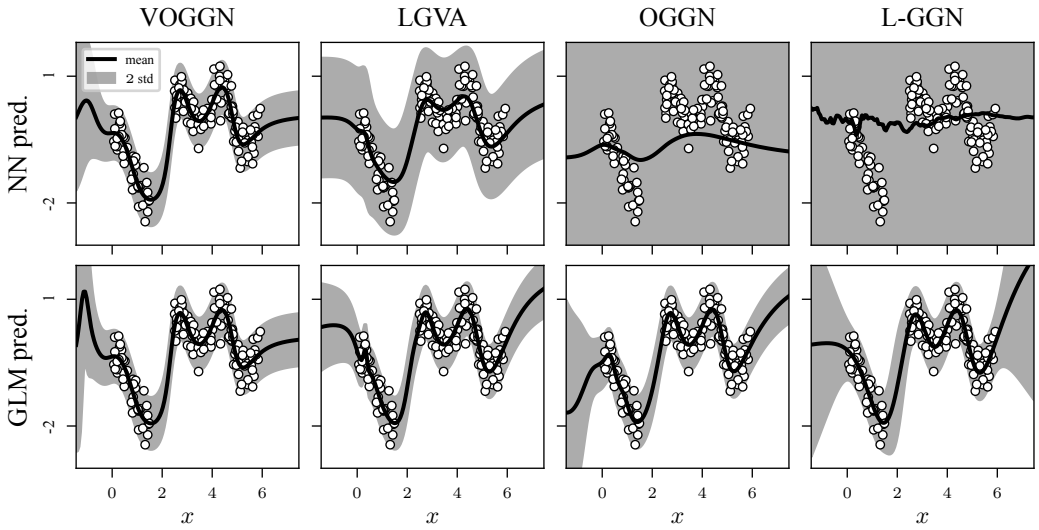
marginal likelihood and an underfitting model that only represents a linear decision boundary. As in the regression problem, the optimal model also exhibits the best uncertainty around the decision boundary as the boundary becomes wider away from the data. In the next section, we focus especially on the properties of the posterior predictive distribution. We use the optimal hyperparameters identified in this section.

5.2 Posterior Predictive Distributions

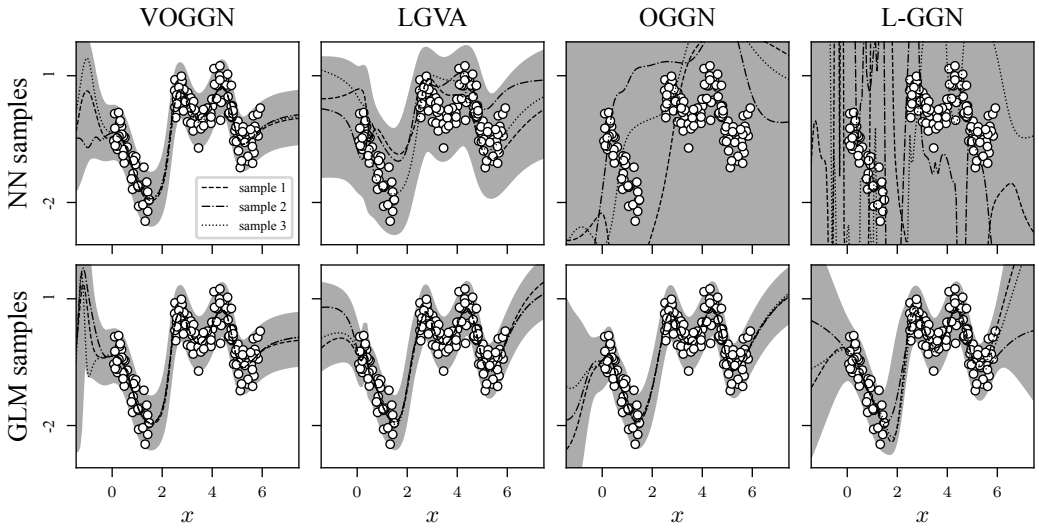
We have introduced three ways to obtain an approximate posterior predictive distribution for a Bayesian neural network: NN sampling, GLM sampling, and BLR inference (cf. section 3.5). Disentangling the GGN and approximate inference, we posed the hypothesis that only VOGGN can lead to a stable predictive using NN sampling. We investigate this hypothesis here. We use the optimal hyperparameters found in the previous section.

We train the neural network models using the three natural-gradient variational inference algorithms introduced in Chapter 4. We use the step size $\beta = 0.999$, a single Monte Carlo sample from the posterior approximation for VOGGN and LGVA per step, and initialize the posterior covariance to $\boldsymbol{\Sigma} = 0.1\boldsymbol{I}_P$. We use the same randomly initialized mean $\boldsymbol{\mu}$ for all algorithms and train until convergence. For the Laplace-GGN (L-GGN) algorithm, we again use Adam to obtain a MAP estimate and then apply the Laplace-GGN posterior approximation. For the NN and GLM sampling posterior predictive approximations, we use 1000 Monte Carlo samples. In the regression case, we have a Gaussian likelihood and therefore the GLM coincides with the BLR model. In the classification case, the GLM corresponds to a Bayesian logistic regression model.

In Figure 5.4, we display the posterior predictive distributions on the regression task for all algorithms using the NN sampling and GLM/BLR inference method. Additionally, we display posterior predictive samples to understand not only the marginal mean and variance but also the joint predictive distribution. The BLR posterior predictive works consistently across all approximate inference methods and provides reasonable uncertainty estimates. Between the methods, there is no significant difference using the BLR predictive. In contrast, the NN sampling posterior predictive fails for the Laplace-GGN and the Laplace-like OGGN since the predictive mean is inaccurate and the variance exceedingly high. The posterior approximation due to LGVA also leads to overestimated predictive uncertainties and an inaccurate mean. Only VOGGN exhibits a posterior predictive that is reasonable and similar to the GLM variant of the VOGGN posterior approximation. On this toy example, we can clearly see that we should predict based on the underlying model that we infer.

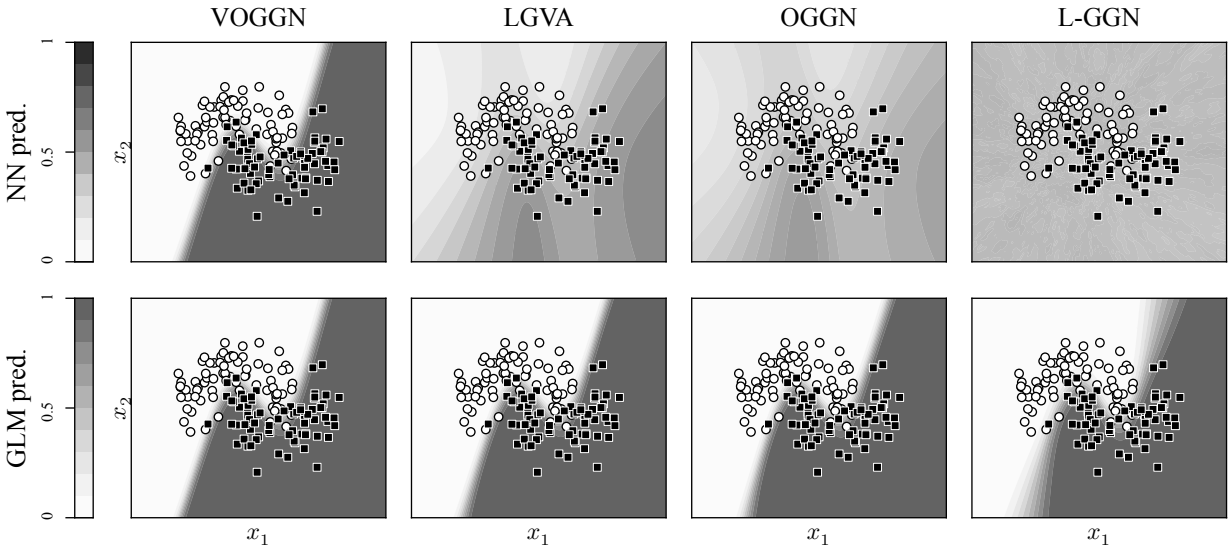


(a) Posterior predictive distribution (NN vs. GLM).

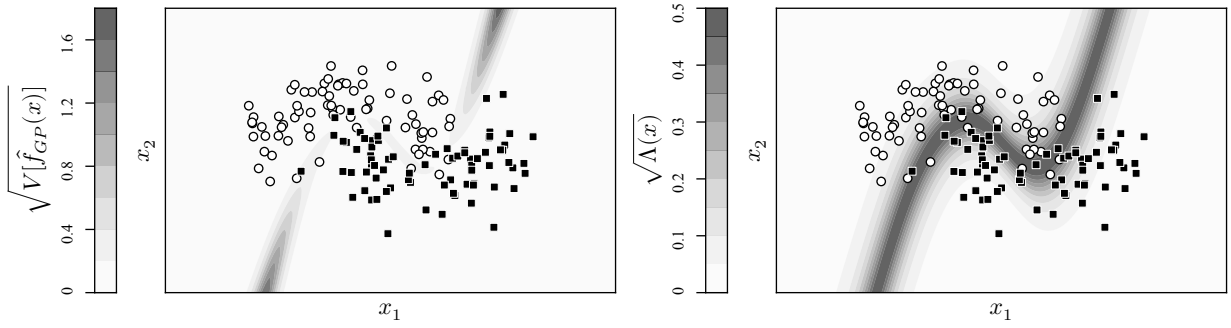


(b) Posterior predictive samples (NN vs. GLM).

Figure 5.4 – Comparison of posterior predictive by NN sampling and GLM sampling for four approximate inference algorithms. The top row shows prediction due to NN sampling and the bottom row shows GLM sampling. In (a), we show the predictive mean and standard deviation. Figure (b) displays three posterior predictive samples. For the Gaussian likelihood used here, GLM sampling is equivalent to the exact Bayesian linear regression predictive. Using the GLM, the predictions are reliable for all algorithms. In contrast, only VOGGN gives reasonable results when predicting by sampling neural networks. Since LGVA samples like VOGGN, it can still give reasonable predictions using NN sampling. OGGN and L-GGN do not work in this case.



(a) Posterior predictive mean (NN vs. GLM).



(b) Posterior predictive uncertainty due to Bayesian linear regression.

Figure 5.5 – Comparison of NN sampling, GLM sampling, and Bayesian linear regression posterior predictive. Figure (a) shows the posterior predictive mean by NN sampling in the top and GLM sampling in the bottom. Only VOGGN maintains good performance using NN sampling. All four algorithms show optimal performance when we use GLM sampling to predict. In Figure (b), we look into the model uncertainty and observation noise due to the Bayesian linear or GP regression model. While the observation noise has simply the variance of the Bernoulli response variables, the model uncertainty on the left grows further away from the data and is relatively low where the decision boundary is supported by data.

In the classification example, we make similar observations: Figure 5.5a depicts the posterior predictives of all methods using NN and GLM sampling. Again, only VOGGN leads to a posterior works for both NN and GLM sampling. In line with the regression problem, the posterior predictive due to the GLM performs consistently across all methods and provides similar predictive distributions for posterior approximations. Prediction using NN sampling is again only viable using the VOGGN posterior approximation. LGVA and OGGN show signs of a decision boundary but the quality is worse than that of the GLM predictive. For the Laplace-GGN posterior approximation, NN sampling leads to uniform predictions and therefore no decision boundary. In stark

contrast, the corresponding GLM sampling predictive yields an optimal predictive model.

In Figure 5.5b, we analyze the Bayesian linear regression model that is inferred exactly when we use the Laplace or Gaussian variational approximation. In particular, we look into the uncertainty in the posterior Gaussian process and the observation noises. The observation noise corresponds to the variance of the modelled Bernoulli random variable and therefore highlights the decision boundary. However, the uncertainty in the Gaussian process is low around the decision boundary and instead grows away from the data. This property could be useful to extend the decision boundary as it is desired for example in *active learning* or *Bayesian optimization*. Interestingly, the model has high model certainty outside of the data as long as it is far away from the inferred decision boundary. This is expected since the kernel of neural networks is typically not stationary [25, 35, 52].

The experiments on posterior predictive distributions show clearly that it is important to understand the approximations used in Bayesian deep learning. Having understood the impact of the GGN on individual approximate inference algorithms, we can choose the right posterior predictive and substantially improve the performance. In fact, the pathology of prediction with NN sampling using the Laplace-GGN posterior approximation has been already pointed out in the literature [9, 44]. Ritter et al. [44] argued that conditioning and the ratio of data points and parameters leads to this problem. The GLM sampling method fixes this problem and it turns out that it works reliably for few data points ($N = 150$) and comparatively many parameters ($P \geq 2000$). The problem is therefore not due to conditioning but a predictive procedure that does not align with the inference and approximation methods used.

5.3 Function-Space Neural Network Inference for Explainability

In this section, we use the Gaussian process formulation of the Laplace-GGN approximation to explain neural network predictions. In chapter 3, we have shown that the generalized Gauss-Newton gives rise to a generalized linear or generalized Gaussian process model. Gaussian process models are instance-based learning algorithms, i.e., they make predictions directly based on the training data. Therefore, we can understand predictions by looking into the training data points responsible for them. In particular, we try to understand predictions of a convolutional neural network on the binary classification task of distinguishing handwritten digits 4 and 9.

The predictive mean of a Gaussian process regression model can be written as an inner product of the kernel between a test point and the training data and an importance vector [42]. We have a vector $\boldsymbol{a} \in \mathbb{R}^N$ that gives an importance factor to each training data point that depends on the likelihood of the generalized GP model. Further, we have the kernel vector $\boldsymbol{k} \in \mathbb{R}^N$ with entries computed by the kernel $\kappa(\boldsymbol{x}_*, \boldsymbol{x}_i)$ between a test data point $\boldsymbol{x}_*$ and the training dataset.

Then, the predictive mean of the GP posterior mean on a new data point can be written as

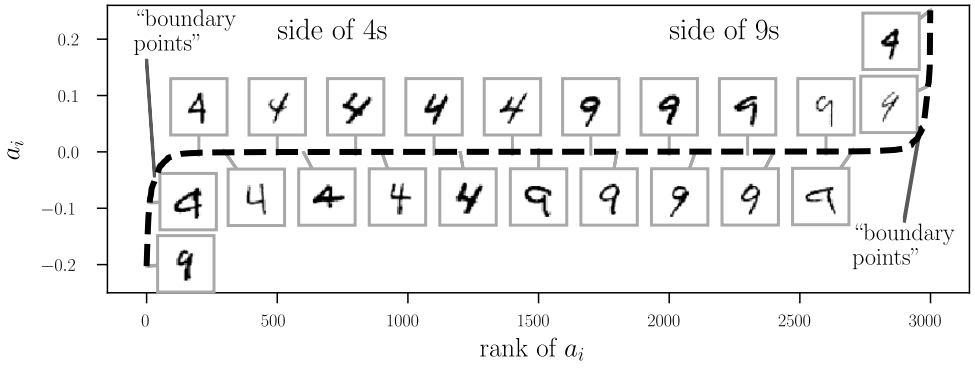
$$f_* = \sum_{i=1}^{N} a_i k(\boldsymbol{x}_*, \boldsymbol{x}_i). \tag{5.2}$$

This allows us to understand the prediction by looking into individual entries of $\boldsymbol{a}$ and $k(\boldsymbol{x}_*, \boldsymbol{x}_i)$. In particular, for a generalized Gaussian process, the *importances* $\boldsymbol{a}$ can be related to the residuals $\boldsymbol{r}(\boldsymbol{y}, \boldsymbol{f})$ of the log likelihood, i.e., the first derivative. In the classification case, we have $a_i = \nabla_{\boldsymbol{f}} \log p(\boldsymbol{y}|\boldsymbol{f}(\boldsymbol{x}_i))$ where $\boldsymbol{f}$ is, for example, our Gaussian process formulation of the neural network and p denotes a Bernoulli likelihood. The kernel $k(\boldsymbol{x}_*, \boldsymbol{x}_i)$ quantifies the *similarity* between a test and training data point according to the feature map or kernel function. Therefore, it allows to identify similar training data points that lead to a particular prediction.

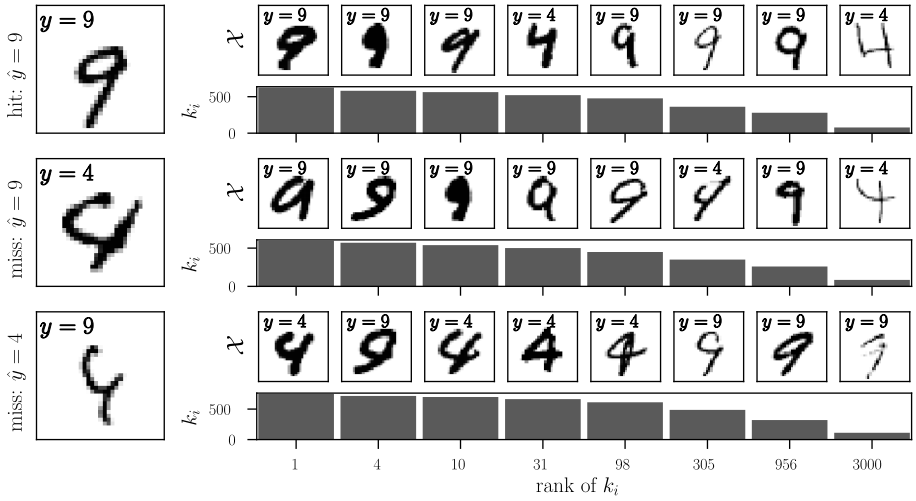
We apply the Laplace-GGN to a convolutional neural network. We train the neural network on the digits 4 and 9 which constitutes the hardest binary classification task on the MNIST dataset [23]. In particular, we randomly select 3000 samples for training. The network has 2 convolutional layers each followed by a `ReLU` activation function and `MaxPooling`. The last three layers are linear and also use the `ReLU` activation function. In total, the network has $P = 4587$ parameters. We use hyperparameter $\delta = 10$. Since the Laplace-GGN approximation can be equivalently cast as the Laplace approximation in a generalized Gaussian process model, we obtain for $\boldsymbol{a}$ the residuals $r(y_i, f(\boldsymbol{x}_i; \boldsymbol{\theta}_{\text{MAP}}))$. The kernel between a new test and a training data point is given by $\kappa(\boldsymbol{x}_*, \boldsymbol{x}) = \delta^{-1} \boldsymbol{J}(\boldsymbol{x}_*; \boldsymbol{\theta}_*) \boldsymbol{J}(\boldsymbol{x}; \boldsymbol{\theta}_*)$.

In Figure 5.6, we analyze both quantities for the binary MNIST problem. Since the model achieves perfect classification on the training data, the residuals are not bigger than 0.2 in Figure 5.6a. The data points with the highest absolute residuals can be understood as boundary points and depict particularly notable examples: 4s that look like 9s and vice versa. These data points have a high impact in decisions since their residuals play an important role in the predictive formulation in Equation 5.2. In the middle range, we have ordinary examples that are easy to distinguish. In Figure 5.6b, we display one correct and two incorrect predictions along with examples of training data points and the kernel value between these samples and the test input. For the incorrect predictions, we find that an example of the opposite class strongly aligns with the test input (see rank 1). In both cases, these samples are also boundary points that highly influence the decision. This ultimately leads to a misclassification.

Understanding predictions of neural networks using an instance-based approach can potentially help make decisions more robust, improve explainability, and identify problematic training data. It would further be interesting to understand how different neural network architectures induce different feature maps and therefore Jacobians. Potentially, some choices lead to good inductive biases for Bayesian neural networks.



(a) Sorted training data importances $\boldsymbol{a}$ with examples.



(b) Classification on test inputs and similarity to training data.

Figure 5.6 – Understanding neural network predictions using a Gaussian process view: in Figure (a), the elements of the importance vector $\boldsymbol{a}$ are displayed in order and with corresponding examples. In the left and right border, we can identify boundary points, i.e., data points that are nearly misclassified: in fact, on the left we have 4s that look like 9s and vice versa on the right. In Figure (b), the similarity between training and test data points is used to understand predictions. On the left, test data points and their predictions $\hat{y}$ are displayed. On the right, we list 8 training data points and their labels together with the corresponding similarity to the test image. The kernel vector $\boldsymbol{k}_*$ for a test image $\boldsymbol{x}_*$ determines the decision together with the vector $\boldsymbol{a}$. We can see that misclassified examples correlate highly with examples of the wrong class due to the learned feature map of the convolutional neural network. Further, the misclassified inputs both show high similarity to points at the decision boundary that highly impact the final prediction.

Chapter 6

Discussion and Future Directions

In this chapter, we discuss the work related to this thesis and conclude the results with a future outlook. Throughout this work, we have referred to related literature where appropriate. Here, all references are summarized in a single place. The conclusion contains a short review of the presented results and discusses possible directions for future work.

6.1 Related Work

This thesis complements and extends the paper "Approximate Inference Turns Deep Networks into Gaussian Processes" [20]. The focus and standpoint of the present work is different. We try to disentangle the generalized Gauss-Newton method and approximate inference in Bayesian deep learning to gain theoretical understanding and practical advantages. In contrast, the prior work focuses on obtaining a Gaussian process representation of neural networks directly from the Bayesian deep learning algorithm [20]. In contrast, the present work can therefore identify an intermediary generalized linear and Gaussian process model that proves to be useful to derive new posterior predictive, marginal likelihood, and inference algorithms for Bayesian neural networks. The Bayesian linear and Gaussian process regression models of both works are equivalent up to reparameterization. The reparameterization plays a major role to practically apply the identified connection. Khan et al. [20] identify a linear and GP regression model in a transformed data space, which, in its original form, cannot replace the neural network model. Next to the two algorithms VOGGN and OGGN introduced by the prior work, the present work additionally introduces the LGVA algorithm that lies between Laplace and variational approximation. The experiments presented here are different since they serve the purpose of particularly showing the impact of the GGN on approximate inference and are based mostly on the novel generalized linear and Gaussian process model formulations. The prior work focuses on the Gaussian process regression formulation that is obtained. Therefore, their marginal likelihood formulation only works for a Gaussian likelihood.

Recently, there has been a surge of interest in the connection of neural network training or inference and Gaussian processes. Williams [52] and Neal [36] already connected Bayesian neural networks to Gaussian processes in the 90s. In particular for a single hidden layer of infinite width, one could show that, under a Gaussian prior, the neural network function can be characterized as a non-stationary Gaussian process [52]. This result has recently been extended to other architectures, activations functions, and depths [25]. The derivation of the neural tangent kernel that characterizes the training of a neural network in function space [17] has again sparked the interest in relating neural networks and Gaussian processes. Based on the work of Jacot et al. [17], Lee et al. [26] showed that infinite width neural networks can be understood as linearized neural networks and therefore be connected to Gaussian process inference if we pose a prior on the parameters. They further found empirical evidence that this connection even holds in the finite setting. In this work, we do not analyze the probabilistic neural network model theoretically but rather the combination of this model with a corresponding practical algorithm. Notably, this gives us similar results and allows to relate neural network inference with Gaussian process inference. In contrast, we do not need to take the limits but obtain similar results due to the combination of the GGN and a Gaussian posterior approximation.

The combination of the generalized Gauss-Newton and approximate inference for Bayesian deep learning is very common. The GGN is mostly applied to approximate the Hessian of the log likelihood. In some cases, it is however used to approximate the Fisher information matrix required for natural gradient descent. For a discussion this, we refer the reader to the recent work of Kunstner et al. [22]. The combination of Laplace approximation and Gauss-Newton is popular and has already been suggested for least-squares regression with neural networks in the 90s [10]. Modern large-scale Bayesian deep learning algorithms based on the Laplace approximation do not use the full GGN approximation but rather diagonal or factorized variants [44] and have successfully been applied to continual learning with neural networks [43]. The Gaussian variational posterior approximation is more popular than the Laplace approximations as it promises to be more precise [2]. In particular, the combination with natural gradients [1], efficient and numerically stable inference algorithms are possible [19, 38, 55]. Prior to their work, the GVA for neural networks mostly relied on backpropagation and the reparameterization trick. The posterior approximation of these algorithms is often unstable and depends heavily on the hyperparameters [4, 9]. Notably, the state-of-the-art results obtained in Bayesian deep learning all rely on a combination of Gaussian posterior approximation and generalized Gauss-Newton method [19, 32, 38, 44, 55]. This work is the first to analyze the interplay of the generalized Gauss-Newton and approximate inference in detail. All above works start from the motivation of a Bayesian deep learning algorithm and use approximations along the derivation of a new algorithm. In contrast, we discuss the impact of individual approximation choices and the impact on the underlying probabilistic model. Further, prior work exclusively relies on NN sampling to approximate the posterior predictive which we find to be unreliable in many cases. The detailed present discussion allows to fix this problem due to two new posterior predictive distributions that are in line with the inferred model.

Problems with the posterior predictive due to a Laplace-GGN approximation to a neural

network are known [44, 9]. Recently, Foong et al. [9] discovered that the BLR predictive works consistently better than NN sampling for univariate Gaussian likelihoods. This work supports this observation theoretically and further extends the result to other likelihoods using the stable GLM sampling prediction method. Further, we justify the same predictive procedure for Gaussian variational approximations. The form of the marginal likelihood based on the Bayesian linear regression model is equivalent to the evidence approximation due to *Occam's razor* [28]. The difference is that we propose to compute the model evidence of the underlying GLM and obtain an additional Gaussian process variant.

The seminal work by Wedderburn [51] studies the impact of the generalized Gauss-Newton method on maximum likelihood estimation for generalized linear models. In particular, he finds that an optimization step using the Gauss-Newton method of a non-linear model with GLM likelihood can be cast as a least-squares regression problem. This least-squares regression problem specifies an adjusted noise variance similar to the one we find here. He was the first to generalize the Gauss-Newton algorithm to GLM likelihoods and therefore defined the generalized Gauss-Newton algorithm. He shows that the generalized Gauss-Newton method applied to maximum-likelihood estimation requires iterative solutions of a least-squares regression problem. In our case we have a similar Bayesian linear regression model that further allows computation of other quantities and can be related to Gaussian processes. While we have specifically focused on Bayesian neural networks, the results presented here also hold for general parametric functions $\boldsymbol{f}(\boldsymbol{x};\boldsymbol{\theta})$ that are at least once differentiable in the parameter $\boldsymbol{\theta}$. This is precisely the case Wedderburn [51] worked with.

6.2 Conclusion

In this thesis, we have disentangled the generalized Gauss-Newton and approximate inference methods in Bayesian deep learning algorithms. The individual analysis of both methods has shed new light on these algorithms: the generalized Gauss-Newton algorithm turns the neural network model into a local generalized linear model. Further, approximating the posterior of the GLM requires the exact solution to a Bayesian linear regression problem which gives us the posterior approximation to the neural network model. We have made use of this new understanding to improve the posterior predictive of Bayesian neural networks, enable empirical Bayes for tuning hyperparameters using the generalized linear model formulation, and identify a function-space posterior approximation for neural networks. The theoretical findings are supported by experiments on simple toy datasets that enable detailed investigation. The present work enables future research in quantifying the accuracy of approximate Bayesian inference using the two identified stages and in applying the derived quantities based on the underlying generalized linear and Gaussian process models to real and larger datasets.

For the Laplace approximation, we have shown that the common combination with the generalized Gauss-Newton optimization method can be understood in two stages: the first stage

consists of obtaining a MAP estimate of the neural network and using the generalized Gauss-Newton that implicitly linearizes the neural network at the MAP. This turns the neural network model into a generalized linear model. The second step is to apply the Laplace approximation. We have found that the Laplace approximation implicitly moment-matches the generalized linear model likelihood to a Gaussian likelihood. This turns the GLM into a Bayesian linear regression model which can be solved exactly and gives us the neural network posterior approximation. Both intermediary models, the generalized linear and Bayesian linear regression model, have Gaussian process counterparts that have the same marginal likelihood and posterior predictive. This connection allows a function-space Laplace-GGN posterior approximation which is useful when the number of parameter greatly exceeds the number of data points. Approximate inference is therefore conducted in the underlying generalized linear or Gaussian process model. Hence, we have argued that these models should be used for the posterior predictive and marginal likelihood computation. The regression models that we need to solve exactly also provide robust predictions but have the wrong likelihood. In particular, we presented the GLM sampling and closed-form BLR predictive distributions that complement the common NN sampling method. The experiments provide practical evidence that the GLM sampling method provides more consistent posterior distributions than NN sampling. Further, selecting hyperparameters using the marginal likelihood approximation to the GLM provides a way to optimize neural network hyperparameters only based on the training data. Lastly, we have shown that the Gaussian process view can be useful for understanding the predictions of neural networks.

The generalized Gauss-Newton method is used in many recent variational inference algorithms for Bayesian deep learning and it is therefore critical to understand its impact. Variational inference differs from the Laplace approximation in one critical point: we have to compute an expectation to obtain the updates or characterize stationarity (cf. section 2.2). In practice, we approximate this expectation by sampling. Therefore, we identified two ways to make use of the GGN: we either sample first and then apply the GGN or vice versa. Applying the GGN first led to the newly introduced LGVA algorithm. Sampling first led to the VOGGN algorithm that we analyzed as a prototype for most variational algorithms for neural networks. The third algorithm, OGGN, is a crude approximation to the LGVA that avoids sampling. We obtained the following understanding of the LGVA algorithm: using the GGN first leads to a generalized linear model as for the Laplace approximation. A variational inference step can then be formulated as a Bayesian linear or Gaussian process regression model. This model made clear that the LGVA algorithm does not sample neural networks but only generalized linear models. We have therefore argued for the GLM sampling method for the posterior predictive. For the VOGGN algorithm we found that that underlying regression problem we solve in each step is specified by different neural network feature maps. This corresponds to sampling neural networks and led to the conclusion that the posterior approximation inferred with VOGGN can make use of the NN sampling posterior predictive. Experimentally, we found results that support this hypothesis. In fact, VOGGN provides the only posterior approximation that allowed to use NN sampling and obtain reasonable predictions consistently. Interestingly, we could not determine the single best algorithm in the experiments. Both the Laplace and variational algorithms provide equally good predictions and uncertainty estimates using the GLM sampling method introduced here.

6.3 Future Work

The results presented in this thesis can be used for future theoretical or applied research. On the theoretical side, the isolation of the generalized Gauss-Newton method and approximate inference in Bayesian deep learning could be useful to investigate convergence, priors, and feature maps. The experiments presented here only serve the purpose of understanding the theoretical results and validate hypotheses. An interesting future direction would therefore be to investigate these results on real or larger-scale data and for other applications.

The individual understanding of the generalized Gauss-Newton and approximate inference for Bayesian deep learning could be useful to jointly specify how accurate these methods really are. In particular, the disentangled understanding also sheds new light on the prior in Bayesian deep learning. In the Laplace approximation, it serves as a regularizer during MAP estimation and as a distribution in approximate inference. However, since we infer a GLM due to the GGN, the prior potentially plays another role during inference. Since the MAP plays an important role, research on the *loss landscape* of neural networks could be useful to understand Bayesian deep learning methods better. Another future direction is to analyze the forms of Jacobians that different architectures and activation functions give rise to. This could help to better understand the GLM or GP kernel that we obtain during approximate inference. Lastly, it is important to understand the further diagonal or factorized approximation applied to the GGN or posterior covariance approximation. The impact of this further approximation is topic of many studies but has not been fully solved yet.

The transformation from probabilistic neural networks to simpler linear and Gaussian process models is potentially useful for many applications. For linear models, many interesting quantities can be computed efficiently and in a numerically stable way. One particularly exciting avenue is to use the marginal likelihood of the underlying linear model to tune neural network hyperparameters. The marginal likelihood is traditionally one of the arguments for a Bayesian approach but its application to model selection has been rarely used in Bayesian deep learning. Here, we have shown that this method works well on toy problems but it is important to scale it to more data points and more parameters. This is also where the Gaussian process variant might be useful: for a reasonably sized dataset ($\sim 10^4$), we can conduct a function-space posterior approximation for an arbitrarily large neural network. This function space approximation leads to the same posterior predictive as a full posterior covariance in the parametric space. I believe that this could also enable interesting applications in *transfer learning*. Lastly, the GLM sampling and exact BLR posterior predictives for Bayesian neural networks introduced in this work seem to provide robust uncertainty estimates and, across methods, work better than traditional NN sampling. It is important to investigate this behavior on other larger datasets. If this observation is consistent, it could enable better performance on applications that require good uncertainty estimates like *active learning* or *Bayesian optimization* using neural networks. The closed-form functional form of the posterior due to the Gaussian process connection could also be useful for regularizing neural networks in the function space as opposed to the parameter space.

Bibliography

[1] S.-I. Amari. Natural gradient works efficiently in learning. *Neural computation*, 10(2): 251–276, 1998.

[2] C. M. Bishop. *Pattern recognition and machine learning*. springer, 2006.

[3] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe. Variational inference: A review for statisticians. *Journal of the American statistical Association*, 112(518):859–877, 2017.

[4] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra. Weight uncertainty in neural network. In *International Conference on Machine Learning*, pages 1613–1622, 2015.

[5] G. Bonnet. Transformations des signaux aléatoires a travers les systemes non linéaires sans mémoire. In *Annales des Télécommunications*, volume 19, pages 203–220. Springer, 1964.

[6] L. Bottou, F. E. Curtis, and J. Nocedal. Optimization methods for large-scale machine learning. *Siam Review*, 60(2):223–311, 2018.

[7] A. B. Chan and D. Dong. Generalized gaussian process models. In *2011 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2011*, pages 2681–2688, 2011.

[8] L. Deng, D. Yu, et al. Deep learning: methods and applications. *Foundations and Trends® in Signal Processing*, 7(3–4):197–387, 2014.

[9] A. Y. Foong, Y. Li, J. M. Hernández-Lobato, and R. E. Turner. 'in-between'uncertainty in bayesian neural networks. *arXiv preprint arXiv:1906.11537*, 2019.

[10] F. D. Foresee and M. T. Hagan. Gauss-newton approximation to bayesian learning. In *Proceedings of International Conference on Neural Networks (ICNN'97)*, volume 3, pages 1930–1935. IEEE, 1997.

[11] C. Gelada and J. Buckman. Bayesian neural networks need not concentrate. `https://jacobbuckman.com/2020-01-22-bayesian-neural-networks-need-not-concentrate/`, 2020.

[12] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.

[13] A. Graves. Practical variational inference for neural networks. In *Advances in neural information processing systems*, pages 2348–2356, 2011.

[14] H. O. Hartley. The modified gauss-newton method for the fitting of non-linear regression functions by least squares. *Technometrics*, 3(2):269–280, 1961.

[15] J. Hensman, M. Rattray, and N. D. Lawrence. Fast variational inference in the conjugate exponential family. In *Advances in neural information processing systems*, pages 2888–2896, 2012.

[16] M. D. Hoffman, D. M. Blei, C. Wang, and J. Paisley. Stochastic variational inference. *The Journal of Machine Learning Research*, 14(1):1303–1347, 2013.

[17] A. Jacot, F. Gabriel, and C. Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in neural information processing systems*, pages 8571–8580, 2018.

[18] M. Khan and W. Lin. Conjugate-computation variational inference: Converting variational inference in non-conjugate models to inferences in conjugate models. In *Artificial Intelligence and Statistics*, pages 878–887, 2017.

[19] M. Khan, D. Nielsen, V. Tangkaratt, W. Lin, Y. Gal, and A. Srivastava. Fast and scalable bayesian deep learning by weight-perturbation in adam. In *International Conference on Machine Learning*, pages 2611–2620, 2018.

[20] M. E. E. Khan, A. Immer, E. Abedi, and M. Korzepa. Approximate inference turns deep networks into gaussian processes. In *Advances in Neural Information Processing Systems*, pages 3088–3098, 2019.

[21] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[22] F. Kunstner, P. Hennig, and L. Balles. Limitations of the empirical fisher approximation for natural gradient descent. In *Advances in Neural Information Processing Systems*, pages 4158–4169, 2019.

[23] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[24] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[25] J. Lee, Y. Bahri, R. Novak, S. S. Schoenholz, J. Pennington, and J. Sohl-Dickstein. Deep neural networks as gaussian processes. 2018.

[26] J. Lee, L. Xiao, S. Schoenholz, Y. Bahri, R. Novak, J. Sohl-Dickstein, and J. Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. In *Advances in neural information processing systems*, pages 8570–8581, 2019.

[27] D. J. MacKay. Bayesian model comparison and backprop nets. In *Advances in neural information processing systems*, pages 839–846, 1992.

[28] D. J. MacKay. Probable networks and plausible predictions—a review of practical bayesian methods for supervised neural networks. *Network: computation in neural systems*, 6(3): 469–505, 1995.

[29] J. Martens. New insights and perspectives on the natural gradient method. *arXiv preprint arXiv:1412.1193*, 2014.

[30] J. Martens and R. Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, pages 2408–2417, 2015.

[31] T. P. Minka. Expectation propagation for approximate bayesian inference. In *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*, pages 362–369. Morgan Kaufmann Publishers Inc., 2001.

[32] A. Mishkin, F. Kunstner, D. Nielsen, M. Schmidt, and M. E. Khan. Slang: Fast structured covariance approximations for bayesian deep learning with natural gradient. In *Advances in Neural Information Processing Systems*, pages 6245–6255, 2018.

[33] K. P. Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[34] E. T. Nalisnick. *On priors for bayesian neural networks*. PhD thesis, UC Irvine, 2018.

[35] R. M. Neal. *Probabilistic inference using Markov chain Monte Carlo methods*. Department of Computer Science, University of Toronto Toronto, ON, Canada, 1993.

[36] R. M. Neal. *Bayesian Learning for Neural Networks*. Springer-Verlag, Berlin, Heidelberg, 1996. ISBN 0387947248.

[37] M. Opper and C. Archambeau. The variational gaussian approximation revisited. *Neural computation*, 21(3):786–792, 2009.

[38] K. Osawa, S. Swaroop, M. E. E. Khan, A. Jain, R. Eschenhagen, R. E. Turner, and R. Yokota. Practical deep learning with bayesian principles. In *Advances in Neural Information Processing Systems*, pages 4289–4301, 2019.

[39] J. Pearl. Fusion, propagation, and structuring in belief networks. *Artificial intelligence*, 29 (3):241–288, 1986.

[40] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[41] R. Price. A useful theorem for nonlinear devices having gaussian inputs. *IRE Transactions on Information Theory*, 4(2):69–72, 1958.

[42] C. E. Rasmussen. Gaussian processes in machine learning. In *Summer School on Machine Learning*, pages 63–71. Springer, 2003.

[43] H. Ritter, A. Botev, and D. Barber. Online structured laplace approximations for overcoming catastrophic forgetting. In *Advances in Neural Information Processing Systems*, pages 3738–3748, 2018.

[44] H. Ritter, A. Botev, and D. Barber. A scalable laplace approximation for neural networks. In *6th International Conference on Learning Representations*, 2018.

[45] L. Sagun, L. Bottou, and Y. LeCun. Singularity of the hessian in deep learning. 2016.

[46] L. Sagun, U. Evci, V. U. Güney, Y. Dauphin, and L. Bottou. Empirical analysis of the hessian of over-parametrized neural networks. *arXiv preprint arXiv:1706.04454*, 2017.

[47] N. N. Schraudolph. Fast curvature matrix-vector products for second-order gradient descent. *Neural computation*, 14(7):1723–1738, 2002.

[48] E. L. Snelson. *Flexible and efficient Gaussian process models for machine learning*. PhD thesis, UCL (University College London), 2007.

[49] D. Titterington et al. Bayesian methods for neural networks and related models. *Statistical Science*, 19(1):128–139, 2004.

[50] H. Wang and D.-Y. Yeung. Towards bayesian deep learning: A framework and some existing methods. *IEEE Transactions on Knowledge and Data Engineering*, 28(12):3395–3408, 2016.

[51] R. W. Wedderburn. Quasi-likelihood functions, generalized linear models, and the gauss—newton method. *Biometrika*, 61(3):439–447, 1974.

[52] C. K. Williams. Computation with infinite neural networks. *Neural Computation*, 10(5): 1203–1216, 1998.

[53] A. G. Wilson. The case for Bayesian deep learning. *NYU Courant Technical Report*, 2019. Accessible at `https://cims.nyu.edu/~andrewgw/caseforbdl.pdf`.

[54] A. Wu, S. Nowozin, E. Meeds, R. Turner, J. Hernández-Lobato, and A. Gaunt. Deterministic variational inference for robust bayesian neural networks. In *7th International Conference on Learning Representations*, 2019.

[55] G. Zhang, S. Sun, D. Duvenaud, and R. Grosse. Noisy natural gradient as variational inference. In *International Conference on Machine Learning*, pages 5852–5861, 2018.

Appendix A

Proof of GLM Log Likelihood Derivatives

Lemma 2.1. *Let $p(\boldsymbol{y}|\boldsymbol{f})$ be an exponential family distribution of the form in Equation 2.4 and let $\boldsymbol{Y}$ denote the corresponding random variable. The first and second derivative of the log likelihood take the simple form*

$$\nabla_{\boldsymbol{f}} \log p(\boldsymbol{y}|\boldsymbol{f}) = = \boldsymbol{y} - \nabla_{\boldsymbol{f}} A(\boldsymbol{f}) = \boldsymbol{y} - \mathbb{E}[\boldsymbol{Y}] = \boldsymbol{y} - \boldsymbol{g}^{-1}(\boldsymbol{f}) =: \boldsymbol{r}(\boldsymbol{y}, \boldsymbol{f}), \tag{2.5}$$

$$\nabla^2_{\boldsymbol{f}\boldsymbol{f}} \log p(\boldsymbol{y}|\boldsymbol{f}) = -\nabla^2_{\boldsymbol{f}\boldsymbol{f}} A(\boldsymbol{f}) = -\mathbb{V}[\boldsymbol{Y}] =: -\boldsymbol{\Lambda}(\boldsymbol{f}), \tag{2.6}$$

where we defined the residual $\boldsymbol{r}(\boldsymbol{y}, \boldsymbol{f}) \in \mathbb{R}^K$ and second derivative, or Hessian, of the negative log likelihood $\boldsymbol{\Lambda}(\boldsymbol{f}) \in \mathbb{R}^{K\times K}$. We have thus identified the inverse link that gives the mean of the response variable as the first derivative of the log cumulant. In the case of an overdispersed *exponential family [33], the variance of the response variable is further scaled by the dispersion parameter σ^2 while the derivatives are divided by it. We have this case, for example, in a Gaussian likelihood (see Table 2.1).*

Proof. We will present a short proof for continuous distributions with scalar natural parameter and label of the form in Equation 2.4. The applied steps directly extend to discrete distributions and become more tedious for multi-dimensional distributions but are analogous. We being with the first derivative with respect to the natural parameter: since $p(y|f)$ is a probability density that integrates to 1, we can rewrite Equation 2.4 as $A(f) = \log \int h(y) \exp\left\{y^\top f\right\} dy$. We start with

the first derivative:

$$\begin{aligned}\frac{\partial \log p(y|f)}{\partial f} &= y - \frac{\partial A(f)}{\partial f} = y - \frac{\partial}{\partial f}\log \int h(y)\exp\{yf\}\,dy \\ &= y - \frac{\frac{\partial}{\partial f}\int h(y)\exp\{yf\}\,dy}{\exp A(f)} = y - \frac{\int \frac{\partial}{\partial f} h(y)\exp\{yf\}\,dy}{\exp A(f)} \\ &= y - \frac{\int yh(y)\exp\{yf\}\,dy}{\exp A(f)} = y - \mathbb{E}[Y].\end{aligned}$$

We used the dominated convergence Theorem to exchange integral and differentiation in the second line. For the second derivative, we have

$$\begin{aligned}\frac{\partial^2 \log p(y|f)}{\partial f^2} &= \frac{\partial}{\partial f}(y - \mathbb{E}[Y]) = -\frac{\partial}{\partial f}\int yh(y)\exp\{yf - A(f)\}dy \\ &= -\int \frac{\partial}{\partial f} yh(y)\exp\{yf - A(f)\}dy = -\int yh(y)\exp\{yf - A(f)\}\left(y - \frac{\partial A(f)}{\partial f}\right)dy \\ &= \left(\int yh(y)\exp\{yf - A(f)\}dy\right)^2 - \int y^2 h(y)\exp\{yf - A(f)\}dy \\ &= \mathbb{E}[Y]^2 - \mathbb{E}\left[Y^2\right] = -\mathbb{V}[Y].\end{aligned}$$

□

Appendix B

Completing the Square

In the proofs, one of the main techniques used is to simply compute the square which leads to a simplification or useful relation. Here, we quickly elaborate on what that means. We assume a symmetric matrix $\boldsymbol{A} \in \mathbb{R}^{D \times D}$ that is positive semi-definite, i.e., we can use a pseudo-inverse if necessary. Further, we have vectors $\boldsymbol{x}, \boldsymbol{b} \in \mathbb{R}^D$. Then, we have the following identity:

$$\frac{1}{2}\boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{x} + \boldsymbol{x}^\top \boldsymbol{b} = \frac{1}{2}\left(\boldsymbol{x} + \boldsymbol{A}^{-1}\boldsymbol{b}\right)^\top \boldsymbol{A}\left(\boldsymbol{x} + \boldsymbol{A}^{-1}\boldsymbol{b}\right) - \frac{1}{2}\boldsymbol{b}^\top \boldsymbol{A}^{-1}\boldsymbol{b}. \tag{B.1}$$

This is exactly the form it shows up in the proofs. We can always use the pseudo-inverse here. Let $\boldsymbol{A}^+$ be the pseudo-inverse of $\boldsymbol{A}$. Then, this holds because of the two properties (1) $\boldsymbol{A}\boldsymbol{A}^+\boldsymbol{A} = \boldsymbol{A}$ and (2) $\boldsymbol{A}^+\boldsymbol{A}\boldsymbol{A}^+ = \boldsymbol{A}^+$.

Appendix C

Natural Gradient Variational Inference

For this section, we will make use of some equivalences in NGVI. To do so, we need two tightly connected parameterizations of the Gaussian posterior approximation: the natural and expectation parameterization. We denote the first and second natural parameter by $\boldsymbol{\eta}^{(1)}, \boldsymbol{\eta}^{(2)}$ and the mean parameters $\boldsymbol{\phi}^{(1)}, \boldsymbol{\phi}^{(2)}$, respectively:

$$\boldsymbol{\eta}^{(1)} = \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu} \quad \text{and} \quad \boldsymbol{\eta}^{(2)} = -\frac{1}{2}\boldsymbol{\Sigma}^{-1}, \tag{C.1}$$

$$\boldsymbol{\phi}^{(1)} = \boldsymbol{\mu} \quad \text{and} \quad \boldsymbol{\phi}^{(2)} = \boldsymbol{\mu}\boldsymbol{\mu}^{\top} + \boldsymbol{\Sigma}. \tag{C.2}$$

The ELBO can equivalently be rewritten in terms of these parameters.

In contrast to vanilla gradient descent, natural gradient descent uses the underlying information geometry as opposed to the euclidean geometry to update parameters. That means, we adjust the parameters of our Gaussian posterior approximation in the space of distributions and not in the euclidean space of the parameters of our Gaussian. Here, we will derive natural variational inference in the natural parameter space. Let $\boldsymbol{\eta}$ the natural and $\boldsymbol{\phi}$ the expectation parameters and $\boldsymbol{F}(\boldsymbol{\eta}) = \mathrm{Cov}_q\left[\nabla_{\boldsymbol{\eta}} \log q(\boldsymbol{\theta})\right]$ the Fisher information matrix of our approximating distribution q [15, 55]. Then, natural gradient variational inference in natural parameter space with step size γ is governed by the following dynamics:

$$\boldsymbol{\eta}_{t+1} = \boldsymbol{\eta}_t + \gamma \boldsymbol{F}(\boldsymbol{\eta})^{-1} \nabla_{\boldsymbol{\eta}} \mathcal{L}(\boldsymbol{\eta}) = \boldsymbol{\eta}_t + \gamma \nabla_{\boldsymbol{\phi}} \mathcal{L}(\boldsymbol{\phi}). \tag{C.3}$$

Since the natural gradient in one parameterization provides the gradient in the other [15], we have the second equality. This allows us to derive the updates for our models in natural parameter space without computing the Fisher information. However, computing gradients with respect to expectation and natural parameters can be inconvenient (especially for backpropagation). Therefore, it is useful to further use the chain-rule and express the expectation parameter gradients in terms of gradients wrt. $\boldsymbol{\mu}, \boldsymbol{\Sigma}$. We have $\boldsymbol{\mu} = \boldsymbol{\phi}^{(1)}$ and $\boldsymbol{\Sigma} = \boldsymbol{\phi}^{(2)} - \left[\boldsymbol{\phi}^{(1)}\right]^2$. Therefore, we can

simply write using the chain-rule:

$$\nabla_{\boldsymbol{\phi}^{(1)}} \mathcal{L}(\boldsymbol{\phi}^{(1)}, \boldsymbol{\phi}^{(2)}) = \nabla_{\boldsymbol{\mu}} \mathcal{L}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) - 2\nabla_{\boldsymbol{\Sigma}} \mathcal{L}(\boldsymbol{\mu}, \boldsymbol{\Sigma})\boldsymbol{\phi}^{(1)} \tag{C.4}$$

$$\nabla_{\boldsymbol{\phi}^{(2)}} \mathcal{L}(\boldsymbol{\phi}^{(1)}, \boldsymbol{\phi}^{(2)}) = \nabla_{\boldsymbol{\Sigma}} \mathcal{L}(\boldsymbol{\mu}, \boldsymbol{\Sigma}). \tag{C.5}$$

We are therefore left with the two gradients with respect to the original parameters to obtain the final update. Recall the form of the ELBO in Equation 2.14 and use the closed-form derivatives of the KL-divergence of the prior p from q:

$$\nabla_{\boldsymbol{\mu}} \mathcal{L}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \nabla_{\boldsymbol{\mu}} \mathbb{E}_q \left[\log p(\mathcal{D}|\boldsymbol{\theta})\right] + \boldsymbol{\Sigma}_0^{-1}\boldsymbol{\mu}_0 - \boldsymbol{\Sigma}_0^{-1}\boldsymbol{\mu} \tag{C.6}$$

$$\nabla_{\boldsymbol{\Sigma}} \mathcal{L}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \nabla_{\boldsymbol{\Sigma}} \mathbb{E}_q \left[\log p(\mathcal{D}|\boldsymbol{\theta})\right] + \frac{1}{2}\boldsymbol{\Sigma}^{-1} - \frac{1}{2}\boldsymbol{\Sigma}_0^{-1}. \tag{C.7}$$

The gradients with respect to the expected log-likelihood can be rewritten using Bonnet's and Price's Theorems as shown in section 2.2. However, especially for the gradient with respect to $\boldsymbol{\Sigma}$, different further approximations are possible and yield different algorithms. Therefore, the next sections will deal with the remaining term and analyze the update in detail.

We obtain the final NGVI in natural parameter space updates by plugging Equation C.4 and Equation C.5 into Equation C.3. We take the gradient at iterate $\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t$ and write the natural parameter updates:

$$\begin{aligned}
\boldsymbol{\Sigma}_{t+1}^{-1}\boldsymbol{\mu}_{t+1} &= \boldsymbol{\Sigma}_t^{-1}\boldsymbol{\mu}_t + \gamma\left[\boldsymbol{\Sigma}_0^{-1}\boldsymbol{\mu}_0 - \boldsymbol{\Sigma}_t^{-1}\boldsymbol{\mu}_t + \nabla_{\boldsymbol{\mu}}\mathbb{E}\left[\log p(\mathcal{D}|\boldsymbol{\theta})\right] - 2\nabla_{\boldsymbol{\Sigma}}\mathbb{E}\left[\log p(\mathcal{D}|\boldsymbol{\theta})\right]\boldsymbol{\mu}_t\right] \\
&= (1-\gamma)\boldsymbol{\Sigma}_t^{-1}\boldsymbol{\mu}_t + \gamma\boldsymbol{\Sigma}_0^{-1}\boldsymbol{\mu}_0 + \gamma\left[\nabla_{\boldsymbol{\mu}}\mathbb{E}\left[\log p(\mathcal{D}|\boldsymbol{\theta})\right] - 2\nabla_{\boldsymbol{\Sigma}}\mathbb{E}\left[\log p(\mathcal{D}|\boldsymbol{\theta})\right]\boldsymbol{\mu}_t\right]
\end{aligned} \tag{C.8}$$

$$\begin{aligned}
-\frac{1}{2}\boldsymbol{\Sigma}_{t+1}^{-1} &= -\frac{1}{2}\boldsymbol{\Sigma}_t^{-1} + \gamma\left[\frac{1}{2}\boldsymbol{\Sigma}_t^{-1} - \frac{1}{2}\boldsymbol{\Sigma}_0^{-1} + \nabla_{\boldsymbol{\Sigma}}\mathbb{E}\left[\log p(\mathcal{D}|\boldsymbol{\theta})\right]\right] \\
&= (1-\gamma)\left[-\frac{1}{2}\boldsymbol{\Sigma}_t^{-1}\right] + \gamma\left[-\frac{1}{2}\boldsymbol{\Sigma}_0^{-1}\right] + \gamma\nabla_{\boldsymbol{\Sigma}}\mathbb{E}\left[\log p(\mathcal{D}|\boldsymbol{\theta})\right].
\end{aligned} \tag{C.9}$$

This form makes apparent that we *combine* the current posterior approximation with the prior usually with a convex combination ($\gamma \leq 1$) of their natural parameters. The data dependency comes solely via first and second derivative of the expected log likelihood under the approximating distribution. Another way of writing above update is therefore to identify natural parameters of p, q with parameters at iteration t, and those arising from gradients of the expected log likelihood [18]. We denote by $q_t(\boldsymbol{\theta})$ the posterior approximation with parameters at iteration t. Then, we can write with another natural parameter $\widetilde{\boldsymbol{\eta}}$ and sufficient statistics $T(\boldsymbol{\theta})$

$$q_{t+1}(\boldsymbol{\theta}) \propto q_t(\boldsymbol{\theta})^{(1-\gamma)} p(\boldsymbol{\theta})^{\gamma} e^{\gamma T(\boldsymbol{\theta})^{\top}\widetilde{\boldsymbol{\eta}}}. \tag{C.10}$$

Clearly, the natural parameter $\widetilde{\boldsymbol{\eta}}$ is given by the gradient terms in above updates. We can easily write the product of q_t and p as another Gaussian and in fact this will be our intermediary prior. The last exponential term will vary depending on how we compute the gradients.