Bayesian convolutional neural networks: a probabilistic approach to address uncertainty in convolutional neural networks with an application in image processing

by

Carl Frederick Steyn

Submitted in partial fulfillment of the requirements for the degree

Magister Scientiae

In the Department of Statistics

In the Faculty of Natural and Agricultural Sciences

Univeristy of Pretoria

November 2018

I, *Carl Frederick Steyn,* declare that this mini-dissertation (100 credits), which I hereby submit for the degree Magister Scientiae in Mathematical Statistics at the Univeristy of Pretoria, is my own work and has not previously been submitted by me for a degree at this or any other tertiary institution.

Signature:

Date:

Abstract

Neural Networks (NNs) play an integral role in modern machine learning development. Recent advances in NN research have led to a wide array of applications, ranging from medical diagnosis [1] to complex problems such as facial and object recognition [2] [3]. However, despite the increasingly powerful predictive capabilities of NNs, some limitations exist which could cause more traditional methods to become the preferred alternative. Most of these limitations result from the "black box" nature of the NN in which the estimated model parameters are not interpretable. The output of traditional NNs also contain no measure of uncertainty in its predictions, causing decision-making to become challenging when NN output plays an important role such as in automatic medical imaging and autonomous vehicles. To address these challenges, we investigate a probabilistic approach to NNs through Bayesian inference and discuss different methods in approximating the posterior distributions of NN parameters. We investigate results when extending the NN structure to deeper architectures such as Convolutional Neural Networks and discuss the advantage of extracting additional information from the posterior predictive distribution to measure prediction uncertainty.

Acknowledgements

I would like to thank my supervisor Dr Alta de Waal, whose continual mentorship and keen interest helped to inspire the work done in this dissertation. Her willingness to hear and explore new ideas encouraged a steady pace throughout the writing of this research document. To the NRF who provided generous financial support during my two years as an MSc student, I express my gratitude. Lastly, I am grateful for the tremendous support my parents have provided throughout my life, without which none of my achievements would've been possible.

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

In recent years, a surge in deep learning interest has taken place and changed the way we see and implement artificial intelligence. Many industries have started to investigate the use of deep learning to harness its powerful predictive capability for a wide variety of applications. From marketing tools such as product recommendation to medical diagnosis via computer vision algorithms, deep learning models have secured their place among the most powerful predictive methods in the modern world.

Deep learning is a term that refers to the use of a Neural Network (NN) designed with an architecture that contains many hidden layers and nodes. However, many different kinds of NNs exist for different applications. For example, a Recurrent Neural Network (RNN) has been shown to work well with sequential data [4] while a Convolutional Neural Network (CNN) was initially designed for image classification and displayed superior performance in comparison to more traditional approaches [5] [3]. A Feedforward Neural Network is known as the oldest design for NN architecture and is still widely used to this day for multiple purposes ranging from simple image classification problems to non-linear regression or classification. In a Feedforward NN, the nodes between layers are all connected and do not form a cycle such as in a RNN. As will be illustrated in this dissertation, the flexible design of NNs can be chosen to suit the complexity of the problem at hand. From the simplicity of a Rosenblatt Perceptron or logistic regression to a network architecture deep enough to classify handwritten digits, an NN is able to show competitive performance for a wide array of problems given the appropriate structure [6].

1.1 Motivation

With deep learning models emerging in industries such as medical imaging and autonomous vehicles, it becomes clear that predictions from such models are important factors that contribute to the decision

making process in research or business. Since NNs are discriminative models and therefore produce only point estimates, we are in many cases uncertain whether the output from the model carries high confidence or is merely a random guess for an observation close to the decision boundary or far away from the training data. We need to better understand the data and model predictions in order to produce an uncertainty estimate which could help stimulate more informed decision making.

This could be achieved by changing our approach from a discriminative model to a generative one. Bayesian inference allows us to define a generative model via a probabilistic framework and infer a posterior distribution over all the unknown weight parameters. By integrating over these parameters for new unseen data, we can compute a predictive distribution from the posterior and use it to understand the uncertainty surrounding each prediction.

Bayesian Neural Networks are a result of applying Bayesian inference to an NN by assigning a prior distribution over the unknown weight parameters and defining a likelihood function from the output of the NN. Since most NN architectures contain a large number of parameters, calculating the posterior distribution could become computationally intractable, as will be explained in more detail in this dissertation. To account for this, we investigate two modern approximation methods in detail. With a focus on image processing, we apply Bayesian inference to a CNN and demonstrate how we achieve state-of-the-art performance on the MNIST handwritten digits dataset through the use of a posterior predictive distribution. Using this distribution we are then able to extract uncertainty measures for new predictions.

1.2 Outline of dissertation

Chapter 2 - Background A literature review covering a history of NN research such as early implementations and training algorithms that made them practically possible. A short discussion follows of how deep learning came to light in earlier years and what caused the sudden surge of interest in these models in recent years as well as limitations regarding modern implementations of deep NNs. Lastly, early literature on the Bayesian approach to NNs is discussed with a final section on important preliminaries that would aid the reader in this dissertation.

Chapter 3 - Bayesian Learning This chapter explores fundamental theory on Bayesian statistics. We reflect on the effect of non-informative vs informative priors and the use of conjugacy, the role of the likelihood function in Bayesian inference and how the posterior distribution provides us with important information about the data using the above-mentioned.

Chapter 4 - Linear Classifier A review on classical linear models and how they can be trained using a Bayesian approach. Doing so, we identify the challenge in performing Bayesian inference on more complex models and the steps needed when extending these models to deeper NN architectures.

Chapter 5 - Approximation Methods This chapter is a comprehensive review of two types of methods used to approximate a probability distribution: Markov Chain Monte Carlo (MCMC) and Variational Inference (VI). The advantages and disadvantages of each of these methods will be discussed and how they can be used to approximate the posterior distribution over the unknown weight parameters in an NN.

Chapter 6 - Neural Networks The theory behind a Neural Network's training algorithm is formally introduced with a derivation of Backpropagation. Various methods to provide scalability to large data and parameter regularisation to prevent over-fitting are discussed with a simple example to illustrate the effectiveness of these methods. With a focus on image classification, Convolutional Neural Networks (CNNs) and their training/regularisation techniques are discussed. Finally, Bayesian NNs (BNNs) and Bayesian CNNs are introduced and we employ the techniques derived in earlier chapters to train a simple model for illustration.

Chapter 7 - Application to Image Classification The predictive power and usefulness of the uncertainty measure from a BNN is demonstrated on the MNIST handwritten character dataset. A Bayesian Convolutional Neural Network (BCNN) is introduced and the increase in predictive power as well as uncertainty measurement is investigated.

Chapter 8 - Conclusion To conclude, we discuss the results from chapter 7, how deep learning models could benefit from a Bayesian approach and the challenges that could arise from such an implementation.

Chapter 2

Background

The goal of designing an intelligent system that learns from observation stems from a long and disputed history of scientific and philosophical research. Inspired by the brain's ability to process information, researchers have been simulating NN systems for decades. In this chapter we do not focus on the biological or philosophical background of the NN, but instead on the mathematical methods that led to the current NN research and the research being conducted to introduce a more probabilistic approach to NNs as well as different training methods associated with it.

2.1 A brief history of Neural Networks

2.1.1 The Rosenblatt Perceptron

One of the most important early designs of an NN was the perceptron, introduced by Franck Rosenblatt in 1958. Developed to imitate a single neuron in a hypothetical nervous system, the perceptron contained a single node that "fires" when the weighted sum of inputs exceeded a threshold, determined by weight parameters in the model [7]. This node (shown in Eq. 2.1 below) is called a unit step function, also known as an "activation function" and is one of the core components in NN architecture.

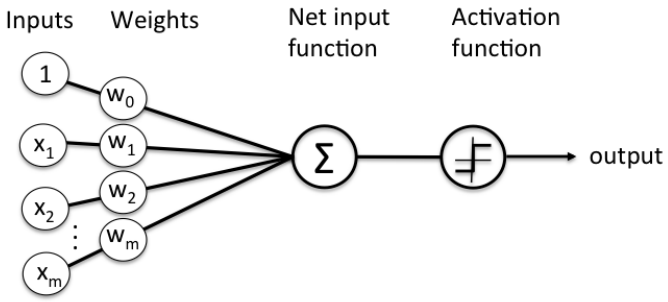


Figure 2.1: Rosenblatt's Perceptron diagram

Structurally, the perceptron bears resemblance to a logistic regression model. The key difference is in the activation function where logistic regression uses a sigmoid function and predicts a continuous value in the range $[0, 1]$, and the perceptron uses a step function which returns a binary value for classification purposes. The step function used in the perceptron is defined below:

$$\hat{y} = f(\underline{w}^\intercal \underline{x}),$$
$$\text{where } f = \begin{cases} 1 & \text{if } \underline{w}^\intercal \underline{x} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

The perceptron iteratively learns the value of its weight parameters by observing labelled examples, known as supervised learning. In the absence of an intercept term, the threshold in the above activation function changes as the values of the weight parameters change with each iteration.

Algorithm 1 Perceptron learning algorithm

- For a set of training data $X = \{\underline{x}_1, ..., \underline{x}_n\}$ with corresponding response variables $\{y_1, ..., y_n\}$ where each $y_i \in \{0, 1\}$:

1. Initialise weight and bias parameters as small numbers (usually chosen as 0)
2. For i = 1 to n:
   - Calculate $\hat{y}_i = f(\underline{w}^\intercal \underline{x}_i)$ from (1.1)
   - Update $\underline{w}_{new} = \underline{w} + (y_i - \hat{y}_i)\underline{x}_i$
3. The error from Step 2 is measured as $\frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i|$. Convergence is met as soon as the error is less than a user-specified threshold, or until a specific number of iterations have been performed.

The weight updates in Step 2 are immediately applied with each consecutive iteration until convergence. Convergence of the perceptron algorithm is guaranteed for a linearly-separable training set [8]. In the case of the training set not being linearly-separable, the algorithm is not guaranteed to converge to an approximate solution.

Following a decade of hype but lack of academic rigor in Neural Network research, Minsky and Papert published a book called Perceptrons in 1969 [9] in which they presented limitations of the perceptron model, specifically its inability to solve non-linear classification tasks such as the popular XOR problem. The XOR problem (short for exclusive-or) shown in Table 2.1 is a non-linearly separable problem that consists of two binary features that generate an outcome based on a simple rule. An XOR function returns a true value (or a 1, for convenience) if the two inputs are not equal and a false value (zero) if they are equal.

x	y	result
0	0	0
1	0	1
0	1	1
1	1	0

Table 2.1: XOR problem

The publication by Minsky et al. contributed to the so-called "AI Winter" during which NN research had difficulty being taken seriously for more than a decade. In essence, they have shown that NN architecture had not yet allowed for any hidden layers or additional nodes, rendering them less capable of learning more complex functions than traditional methods.

2.1.2 Multilayer perceptrons and backpropagation

In 1986, Rumelhart et al. showed that a simple method called backpropagation [10] could be used to learn the weights of a perceptron model with more than one hidden layer (also called a multilayer perceptron), which inspired the well-known NN structure as used in present-day research and application. The gradient-descent algorithm (shown in Algorithm 2) uses backpropagation to incrementally adjust the weight parameters in each layer, in the negative direction of the loss function's derivative with respect to the weight parameters. Backpropagation allows us to utilise a multilayer perceptron's hierarchical structure to find the derivatives of weight parameters throughout different layers within the model.

Through this method, the multilayer perceptron (or NN) was able to learn non-linear functions through

various differentiable activation functions such as the sigmoid activation function $\sigma(x) = \frac{1}{1+e^{-x}}$ used in logistic regression or hyperbolic tangents $\sigma(x) = tanh(x)$. These alternative activation functions play a major role when fitting an NN to data generated by a nonlinear function. Rumelhart's findings proved to be a breakthrough for neural network research, as the improved performance greatly reduced scepticism around the topic.

Algorithm 2 Gradient descent

- Let X be a set of training data $X = \{\underline{x}_1, ..., \underline{x}_n\}$ with corresponding response variables $\{y_1, ..., y_n\}$
- The error function is defined as $S = \frac{1}{2n}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$ where $\hat{y}_i$ is the output from the multilayer perceptron for training example $\underline{x}_i$
- The multilayer perceptron contains L hidden layers, each layer containing its own set of weight parameters $W^{(\ell)}$
- Starting values for weight and bias parameters are chosen at random
- The learning rate $\lambda \in [0, 1]$ determines the step size of parameter updates
- The following steps are followed until convergence:
  1. The data X passes through the multilayer perceptron with current parameter settings to obtain $\hat{y}$
  2. Calculate the partial derivative of S with respect to each weight matrix $\frac{\partial S}{\partial W^{(\ell)}}$
  3. Update each weight matrix as $W^{(\ell)}_{new} = W^{(\ell)}_{old} - \lambda\frac{\partial S}{\partial W^{(\ell)}_{old}}$
- Convergence is met when the decrease in error from each iteration becomes minimal and the error function reaches a local optimum as shown in Figure 2.2. However, the definition of "minimal" is up to the user and choosing an appropriate value can become challenging.

Following Rumelhart's publication and an increasing amount of research into the multilayer perceptron, powerful new ideas for NNs emerged such as recurrent neural networks (RNN) which considers the data's behaviour over time and is used for tasks such as handwriting [11] or speech recognition [12]. Convolutional Neural Networks (CNNs) [5] imitate the visual cortex of the brain by shifting focus over a receptive field (or small sample of a dataset), and was shown to perform well when applied to tasks such as image classification and object recognition [3]. However, these methods used in high dimensional models did not scale well to large sets of data as a result of hardware limitations during the 80s and 90s and were outperformed by cheaper, more traditional algorithms such as non-linear regression and decision trees.

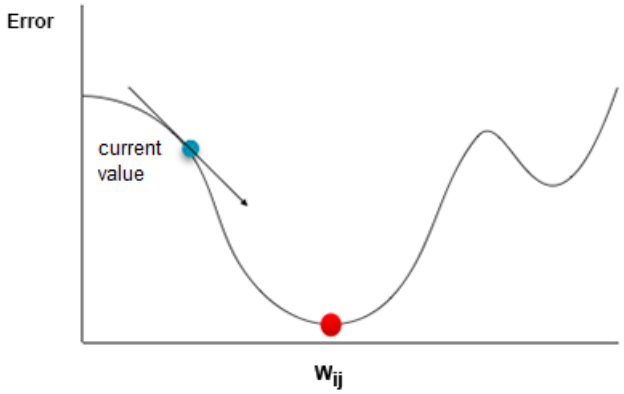


Figure 2.2: Gradient descent on neural network weights

2.1.3 Deep Neural Networks

The scalability issue surrounding NNs was alleviated by Geoffrey Hinton in 2006 by training shallow NNs and stacking them together to create a deep network [13]. Although deep learning in NNs has been researched for decades prior to Hinton's publication, it was not practically viable since computationally efficient solutions were limited. Following Hinton's publication, deep learning saw an increase in attention from researchers, leading to improved algorithms for complex classification tasks such as optical character recognition on the well-known MNIST handwritten digits dataset [14] [15].

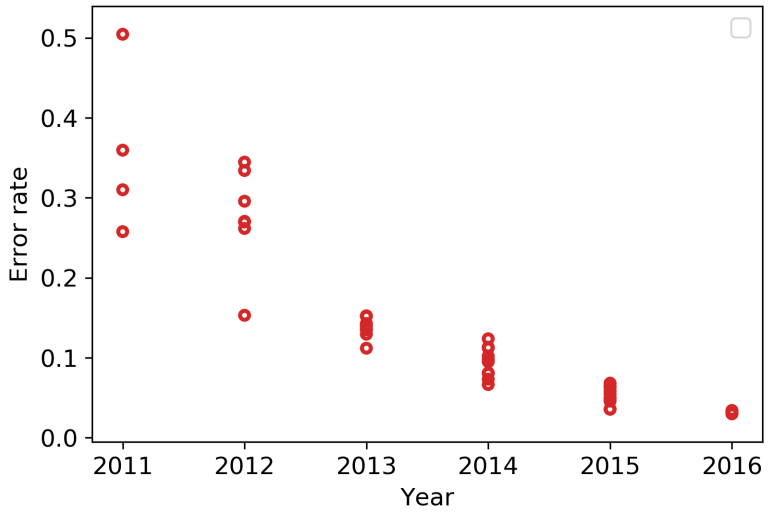


Figure 2.3: Imagenet competition results

One of the largest contributions to deep learning's more recent popularity was the 2012 ImageNet Large Scale Visual Recognition Challenge (ILSVRC). The competition evaluates the performance of algorithms

trained on a large dataset created in 2010 [16]. A team called Supervision employed a deep CNN called AlexNet [3] designed by Alex Krizhevsky that significantly outperformed image classification methods used in earlier years, sparking the fascination around deep learning as it is today. As shown in Figure 2.3, Imagenet results from consecutive years after 2012 has shown significant improvements in performance.

Open-source software such as Python and R have seen to an increasing use of deep learning in industry-related practice as well as research. The cost-free availability of packages such as Scikit-Learn [17] used in Python and rpart [18] used in R provide easy-to-use machine-learning algorithms which include a wide family of NN-based methods. A research team at Google called Google Brain formed in 2010 has also been on the forefront of open-source NN software. Led by artificial intelligence pioneers such as Geoffrey Hinton and Andrew Ng, Google Brain produced numerous breakthroughs using deep learning applied to various fields such as healthcare [19] and robotics [20]. Released early 2015, Google Brain developed an open-source platform in Python called Tensorflow [21], which is the primary Python package used for distributed computing with deep neural networks. This package allows multiple processing units such as CPU's or GPU's to work in parallel for much faster computing in a deep learning environment.

2.2 Limitations in Neural Networks

Destpite its recent popularity, deep NNs using frequentist estimation techniques such as backpropagation have been shown to suffer from several disadvantages. Examples of such disadvantages include gradient-based training being susceptible to coverging on local minima [22] and that frequentist models in general have difficulty measuring uncertainty when predicting future observations [23]. Deep NNs are also not immune to overfitting without the use of additional methods such as L1 and L2 regularisation or dropout in the case of limited training data [24].

The problem of overfitting is inherent to most supervised machine learning algorithms, especially when the model includes too many parameters and/or the training data is limited. Regularisation methods such as L1 and L2 exist to encourage values of smaller scale in parameters in favour of the model becoming less complex and more generalised. This is achieved by adding an extra term to the loss function:

$$L = \sum_{i=1}^{N}(y_i - \underline{w}^\intercal \underline{x})^2 + \lambda \sum_{j=1}^{p} |w_j| \qquad \text{(L1 regularisation)}$$

$$L = \sum_{i=1}^{N}(y_i - \underline{w}^\intercal \underline{x})^2 + \lambda \sum_{j=1}^{p} w_j^2 \qquad \text{(L2 regularisation)}$$

In the case of NNs, more advanced methods have been researched to counter overfitting. In addition to dropout [24], the use of mixed Gaussian priors over weight parameters for feature selection has shown su-

perior performance to the traditional regularisation methods [25]. These methods, however, do not allow any uncertainty in the model to be measured. With no consideration of uncertainty in model predictions, an NN risks making an overly confident decision.

Put differently, an NN in a classification scenario would classify an input to a class with the highest corresponding output probability. No additional information (such as standard deviation) regarding the true distribution of the output or weight parameters is known. In cases where important decisions need to be made from NN output, (such as autonomous vehicles [26]) uncertainty plays a critical role and could be used to prevent adverse results.

2.3 Bayesian Neural Networks

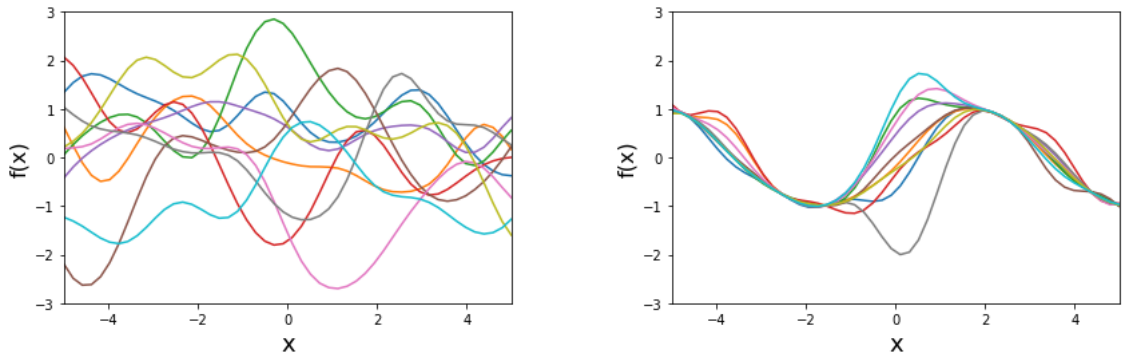
The above problem is an important concept in Bayesian learning, which uses a probabilistic framework to uncover posterior distributions over variables of importance. The demand to introduce a probabilistic approach to account for the above issues has led to Bayesian Neural Networks (BNNs) gaining more attention in neural network research [26] [22]. Early research on this topic suggested fitting a prior distribution over the weight parameters and invoking Bayes' theorem to derive the posterior distribution [27] [28]. Different prior distributions for the weight parameters have been investigated [29], leading to interesting results as the number of hidden components approach infinity. However, due to the hardware limitations present at the time, the posterior distribution needed to be approximated using sampling or variational techniques [27] [30]. Modern approximation methods used for Bayesian inference in NNs include Markov Chain Monte Carlo (MCMC) [29], variational inference [26] and various hybrid methods often combining both MCMC and variational inference [31].

As part of the MCMC family of algorithms, the Metropolis-Hastings algorithm developed in 1953 by Metropolis et al. [32] and later generalised by Hastings [33] has created an enormous impact on artificial intelligence by allowing high dimensional distributions to be approximated. The Metropolis-Hastings algorithm, however, was not widely used by statisticians until the early 90's where Tierney [34] investigated the use of Markov Chains to explore posterior distributions. As pointed out by Gelman in 1992, the well-known Gibbs sampler is a special case of the Metropolis-Hastings algorithm [35] as is widely used in machine learning today, including BNNs.

Although MCMC produces asymptotically exact samples from the posterior distribution [36], it does not scale well to very large datasets due to the computational intensity inherent in MCMC methods. As an alternative, variational inference [37] (VI) produces a faster approximation of the posterior distribution. Since VI produces an approximation through optimization, methods such as stochastic optimization have

been shown to scale well to large sets of data [38]. Therefore, there exists a trade-off between MCMC and VI. For smaller sets of data, MCMC is usually preferable over VI which is best suited for large sets of data [39].

In his paper [29], Radford Neal points out that using Bayesian learning with NNs not only provides a measure of uncertainty for parameter settings and predictions, but allows the use of infinitely many hidden components regardless of the size of the training set without overfitting the data. He further shows that, in the limit of an NN containing an infinitely wide single hidden layer, the outputs can be compared to a Gaussian Process (GP) prior over functions. This enables the use of exact Bayesian inference for regression tasks with neural networks [40] [29] by gaining the posterior distribution over different functions by GPs (as shown in Figure 2.4) while enjoying the scalability and flexibility of an NN. By extending this result, Lee et al. shows that a deep NN containing infinitely wide hidden layers shows the same property [41], allowing BNNs to easily extend to deep NN architectures. BNNs with deep architectures may provide insight to more complex relationships in the data without penalizing performance on future, unseen data by overfitting.



(a) GP prior distribution over functions

(b) GP posterior distribution over functions

Figure 2.4: Illustration of the prior and posterior of a Gaussian Process

2.4 Review of important concepts

2.4.1 Generative vs. Discriminative models

An important choice in statistical modeling is whether a generative or discriminative model should be used. It is therefore important to clarify the difference between the two classes of models. For this section, let $\mathcal{D} = \{(x_1, y_1), ..., (x_n, y_n)\}$ be a set of pairs of observed data points, each consisting of an independent variable x with a corresponding dependent variable y.

Discriminative models (such as linear regression, logistic regression or Support Vector Machines) model the conditional probability distribution of the response variable on the input variable, i.e. $P(y|x)$. For example, in a linear regression model our hypothesis is $\mathbb{E}[Y|X] = \beta_0 + \beta_1 X$, i.e. the predicted values $\hat{y}$ are obtained by calculating the mean of the conditional distribution.

In a classification environment such as logistic regression, the discriminative model focusses on the boundary between classes (i.e. *discriminating* between the classes of x) and not on the distribution of each class as generative models do. As opposed to generative models, the data $\mathcal{D}$ cannot be generated from a discriminative model - only the output y can be generated for a given example x. It is generally known that discriminative models tend to learn more complex relationships as the dataset increases [42]. However, for smaller or missing data, generative models are more capable of generalising to prevent over-fitting of data where discriminative models may learn irregular patterns which aren't representative of the data.

Generative models (such as Naïve Bayes or Gaussian mixture models) describe the way in which the data was generated through the use of a joint probability distribution $P(X, y)$. These models typically require stronger assumptions than discriminative models do. For example, Naïve Bayes assumes that all the variables contained in X are conditionally independent on y, i.e. $P(x_1, .., x_p|y) = \prod_{i=1}^{p} P(x_i|y)$. This is a strong assumption which is seldom satisfied in practice. Generative models become the better choice when the objective isn't only to predict future data, but to draw information from underlying distribution of the data and be able to generate additional data from said underlying distributions.

The estimated parameters $\hat{\theta}$ of an NN are usually obtained through maximum likelihood estimation, satisfying the following expression:

$$\hat{\theta}^{MLE} = argmax_\theta \sum_{i=1}^{N} logp(y_i|x_i, \theta).$$

Therefore, NN's are mostly considered discriminative models. However, this dissertation will focus on NNs from a generative perspective through means of Bayesian inference.

2.4.2 Linear vs non-linear

When a model is said to be "linear", the linearity occurs within the parameters of the model. It is then assumed that the dependent variable is linearly related to the independent variables. This distinction becomes important when considering polynomial regression such as

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2^2.$$

Even though a non-linearity occurs in the independent variable, the dependent variable remains a linear combination between the data and the parameters. Using a linear model is a good option when the data

exhibit suitable patterns, usually found through visual inspection of the variables. Training linear models is faster and computationally cheaper than non-linear models (usually done by ordinary least squares).

When data exhibits more complex patterns not suitable for linear models, a non-linear model is preferred. Non-linear models are capable of explaining complex relationships between the dependent and independent variables. However, the optimization of non-linear models are generally harder and usually requires numerical approximation when the problem cannot be transformed to a linear domain. When optimizing the objective function of a non-linear model though iteration, appropriate initial values of the parameters become a critical factor. Furthermore, for non-linear models containing a large number of parameters, the training algorithm becomes computationally expensive.

NNs with non-linear activation functions are considered a class of non-linear models and are capable of learning highly complex patterns in some datasets, especially in a deep learning architecture.

2.4.3 Parametric vs non-parametric

The number of parameters found in non-parametric methods increase as the amount of data increases, meaning that we may have a potentially infinite number of parameters in a non-parametric model. These models usually require fewer assumptions than parametric methods and are preferred over parametric methods in unsupervised learning environments such as cluster analysis. Examples of non-parametric learning algorithms include:

- k-nearest-neighbors
- k-means clustering
- decision trees

When using parametric methods to model data, we decide on the number of parameters included in the model beforehand. There are often strong assumption we need to make about the distribution of the data for the model to perform well, for example in a linear regression we assume a linear relationship between the dependent and independent variables as well as normality of the error terms. Examples of parametric learning algorithms include:

- linear and logistic regression
- Naïve Bayes classification
- linear support vector machines

NNs are considered parametric learning algorithms since we decide beforehand on the number of hidden layers and nodes contained in each layer. However, an advantage of an NN is that there is no need for any assumptions regarding the distribution of the data. This makes the NN a particularly useful algorithm when we have a large amount of data but lack information on the behaviour of the data.

2.5 Summary

Although deep learning is often considered a modern field of research rapidly advancing as computational power and data volumes grow, this literature review served as an account of important past research which contributed to the field's recent success.

Arguably the oldest implementation of the feedforward NN is the Rosenblatt perceptron introduced in Section 2.1.1. Built upon this design but trained differently, the multilayer perceptron allows the use of deeper architectures to solve non-linear problems. Although possible in theory, practical implementation of a multilayer perceptron containing deep architectures were computationally infeasible during those years. Section 2.1.2 discussed the origin of the multilayer perceptron and how it became possible to train models comprised of deeper architectures on computers from previous eras.

Section 2.2 mentioned possible shortcomings of NNs and Section 2.3 discussed previous research done on the approach we have chosen to address these challenges. Lastly, Section 2.4 provided a brief review of important concepts discussed in this dissertation and brought this chapter to an end.

Chapter 3

Bayesian Learning

Bayesian inference is a statistical technique where we employ Bayes' theorem to update our belief of a population, given new evidence obtained from a sample. Using a Bayesian approach to draw inference from data means that we adopt a probabilistic framework. Our knowledge about an outcome is expressed as a probability distribution (called a posterior distribution) and is obtained by combining our prior belief with observed evidence from data in the form of a likelihood function.

A key difference between Bayesian and frequentist methods is the way in which probability is interpreted [43]. A frequentist relates the probability of an event to the relative *frequency* of the event's occurrence in the limit of a large number of repeated experiments under the same conditions. A Bayesian uses the concept of probability as a reflection of their uncertainty of an event as it changes with new information. This definition provides a basis for us to appeal to Bayesian inference as a means to addressing uncertainty in deep NNs.

For this section, let $X = \{\underline{x}_1, \underline{x}_2, ..., \underline{x}_n\}$ be a matrix of input variables with a vector of response variables $\underline{y} = \{y_1, y_2, ..., y_n\}$. We assume that the data was generated from some function $\underline{y} = f(X; \underline{\theta})$ with a collection of unknown parameters $\underline{\theta}$.

3.1 Likelihood

In a statistical context, likelihood refers to the plausibility of parameter values for a sample of observed data. This measure of plausibility is the conditional probability of the data given those specific parameter

values, i.e.

$$\mathcal{L}(\underline{\theta}|X, \underline{y}) = P(X, \underline{y}|\underline{\theta}).$$

Many frequentist estimation techniques are based on maximum likelihood estimation (MLE), where the optimal parameter values are those that maximise the likelihood function.

In a Bayesian framework, the likelihood function is used in conjunction with the prior distribution to obtain a posterior distribution. The mode of the posterior distribution of the parameters produces the maximum a posteriori (MAP) estimates of the parameters, i.e. the parameter values that satisfy the Eq.

$$\hat{\underline{\theta}}_{MAP} = argmax_{\underline{\theta}} P(\underline{\theta}|X, y).$$

When using Bayesian inference specifically to estimate parameters from a set of data, the MAP values are a popular choice. However, in the limit of infinite data, the MAP estimate can be shown to converge to the MLE since the evidence of the data contained in the likelihood overwhelms the prior distribution [43]. The specific structure of the likelihood function also has an important role to play in Bayesian inference, as will be shown in Section 3.2.2 where we discuss conjugacy.

3.2 Prior

The prior distribution represents our prior knowledge of an unknown quantity in Bayesian inference before we take the evidence from a set of data into account. Depending on the problem, the unknown quantity of interest may be a latent variable that can only be inferred instead of observed, or a parameter used in a statistical model. The choice of prior is of critical importance in Bayesian inference, especially in cases where we have limited data. An inappropriate choice of prior can lead to misleading results when performing inference. In addition to the specific probability distribution chosen, different types of priors exist which allows Bayesian inference to be performed on a wide variety of problems, depending on the prior information the analyst has at hand.

3.2.1 Informative vs uninformative priors

Informative priors are prior distributions which incorporate specific information gathered on the data before calculating the posterior. Examples of such information could be expert opinions or literature on the data being analysed, or the results of previous surveys. Although it sounds useful to apply such information when analysing data, it could lead to problematic results if not applied carefully. The prior opinions of experts may be biased and not adequately represent the population. Conversely, the appropriate use of an informative prior could lead to sensible restrictions placed on parameter values and improve estimation

especially in cases where data is limited. In cases of a conjugate prior (discussed in the next section), the posterior distribution obtained from a sample could be used as an informative prior for inference on future data sampled from a similar population.

Uninformative priors express a general shape of the unknown quantity of interest without incorporating as much bias into the posterior as an informative prior would. These prior distributions often allow for a wider range of possible values with close to equal probabilities. The estimations resulting from a posterior distribution using an uninformative prior are usually close to the estimations resulting from frequentist techniques since the likelihood of the data provides more information than the uninformed prior. This is especially true when using large samples of data.

For example, consider a coin-tossing experiment, where we make the assumption that the coin is balanced. The outcome of each coin-flip experiment has a $Bernoulli(p)$ distribution where p denotes the probability of observing heads (H) and X is the number of heads observed in a certain number of tosses. We can assume an uninformative prior for the proportion p by using a Uniform(0,1) prior distribution. This prior can be considered an uninformative prior since we only express a general shape of the distribution of p, i.e. $p \in [0.1]$ where each value of p in that interval has equal probability. As we observe more coin-tosses, the likelihood of the observed data would overwhelm the prior, causing the MAP estimate from the posterior distribution to converge to the MLE estimate.

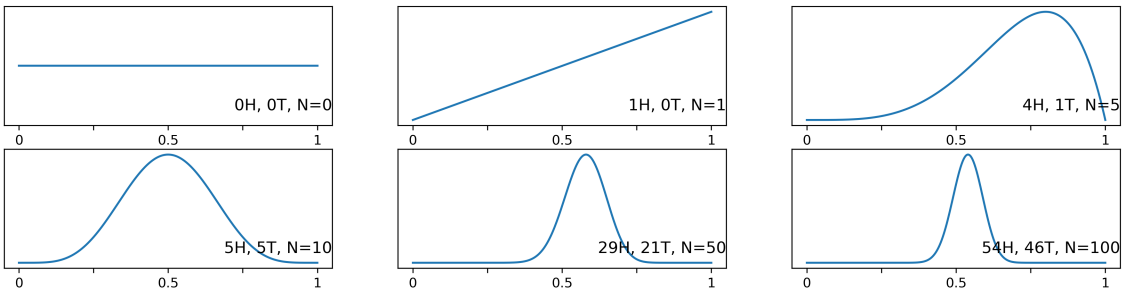


Figure 3.1: Posterior distributions of p from coin-tossing experiments. Each subplot indicates the observed heads (H) or tails (T) from a certain number of coin tosses (N).

As shown in Figure 3.1, the uninformative uniform posterior in the top left is shown before any coin flips have been observed. As we observe results, the shape of the posterior becomes more centered around the true value of p.

3.2.2 Conjugacy

We define a prior distribution as a conjugate prior when the resulting posterior distribution possesses a similar structure as the prior. For example, suppose that we have a sample of i.i.d data $\underline{x} = \{x_1, x_2, ..., x_n\}$ which follows a Bernoulli distribution with parameter $\theta \in [0, 1]$. If we are interested in deriving the posterior distribution for θ, we would use the likelihood function of the observed data

$$p(\underline{x}|\theta) = \prod_{i=1}^{n} \theta^{x_i}(1-\theta)^{1-x_i}, \tag{3.1}$$

together with some prior distribution $P(\theta)$. By inspecting Eq. 3.1, it follows that the Beta distribution has a similar structure up to a constant with respect to the parameter θ. Therefore, we choose the $Beta(\alpha, \beta)$ distribution as a prior with hyper-parameters (parameters of the prior distribution) α and β:

$$\begin{aligned} p(\theta) &= \frac{1}{B(\alpha,\beta)}\theta^{\alpha-1}(1-\theta)^{\beta-1} \\ &\propto \theta^{\alpha-1}(1-\theta)^{\beta-1} \end{aligned}$$

where $B(\alpha, \beta)$ is a ratio of gamma functions $\frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}$ which stays constant with respect to θ.

Next, using Bayes' theorem we multiply the likelihood function and prior to derive an expression of the posterior distribution:

$$\begin{aligned} p(\theta|\underline{x}) &= \frac{p(\underline{x}|\theta)p(\theta)}{p(\underline{x})} \qquad (3.2) \\ &\propto p(\underline{x}|\theta)p(\theta) \\ &\propto \Big(\prod_{i=1}^{n} \theta^{x_i}(1-\theta)^{1-x_i}\Big)\Big(\theta^{\alpha-1}(1-\theta)^{\beta-1}\Big) \\ &= \theta^{\sum_{i=1}^{n} x_i+\alpha-1}(1-\theta^{n-\sum_{i=1}^{n} x_i+\beta-1}) \\ &\sim Beta(\sum_{i=1}^{n} x_i + \alpha, n - \sum_{i=1}^{n} x_i + \beta). \qquad (3.3) \end{aligned}$$

It follows from Eq. 3.3 above that the posterior $p(\theta|\underline{x})$ also follows a Beta distribution, meaning that the Beta distribution is a conjugate prior for the Bernoulli likelihood function.

The term $p(\underline{x})$ in Eq. 3.2 is known as evidence of the data (also called the normalisation factor) and is constant with respect to θ. In this example, it follows that

$$p(\underline{x}) = B(\alpha, \beta).$$

Using a conjugate prior is a mathematical convenience because it allows us to use the posterior as a prior when we receive new data and wish to update our belief of the value of θ. It also provides us with an analytical expression of the evidence $p(\underline{x})$, which otherwise can become very challenging to compute for

complex models.

Consider again the coin-tossing experiment from the previous section. By using a Beta(6,6) prior distribution (an informative prior), we restrict the value of p to the interval [0.1] and place higher probability on the value $p = 0.5$. Put differently, we use our prior knowledge of coins in general and assume that the coin is balanced.

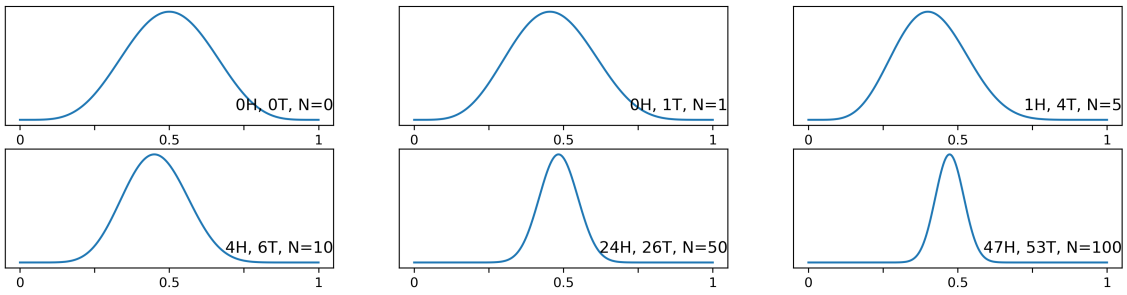


Figure 3.2: Posterior distributions of p from coin-tossing experiments. Each subplot indicates the observed heads (H) or tails (T) from a certain number of coin tosses (N).

From Figure 3.2 we see that the prior distribution (top left) has the same general shape as the posterior. By observing data and incorporating the likelihood we only change the width and central location of the distribution. These posterior distributions can then conveniently be used as a prior distribution for future coin-tossing experiments under the same circumstances (using the same coin, for example).

3.3 Posterior distribution

If the data was generated by some function $\underline{y} = f(X;\theta)$ with a collection of unknown parameters θ, the Bayesian approach would be to express our knowledge of θ by deriving a posterior distribution for θ. Before observing any data, our initial belief about the possible values of θ is captured in a prior distribution that we specify. Accordingly, we use Bayes' theorem to combine the prior with the likelihood function of observed data:

$$P(\theta|X,\underline{y}) = \frac{P(\underline{y},X|\theta)P(\theta)}{\int P(\underline{y},X|\theta)P(\theta)d\theta}. \tag{3.4}$$

Eq. 3.4 produces an updated probability distribution over all the possible values of θ, conditional on the observed data. It can be acquired by combining the likelihood function $P(\underline{y},X|\theta)$, the prior distribution and normalisation constant $\int P(\underline{y},X|\theta)P(\theta)d\theta$ also referred to as the "evidence" of the data. The purpose of the normalizer is to allow the posterior distribution to integrate to 1, which is a requirement for a valid

probability distribution. When using a conjugate prior, the entire posterior distribution becomes available without having to calculate the evidence separately. However, as convenient as conjugate priors are, they are rarely available or suitable to the problem. For higher dimensional problems where a conjugate prior is not available, the calculation of the evidence may become problematic. This is because the expression $\int P(\underline{y}, X|\theta)P(\theta)d\theta$ requires us to integrate over all possible parameter settings in the prior and likelihood. In the case of deep learning, for example, extremely complex models are often used containing hundreds of thousands of parameters. Calculating the evidence therefore becomes computationally intractable and requires us to approximate the posterior instead. Approximation methods for the posterior will be discussed in Chapter 5.

3.4 Posterior predictive distribution

A posterior predictive distribution can be derived using the posterior to produce a probability distribution over new/unseen data. For a new observation x', we derive the posterior predictive distribution of its corresponding response variable y' given the observed data X and x' by integrating the likelihood of the new data with respect to the posterior distribution:

$$P(y'|x', X) = \int P(y'|x', X, \theta)P(\theta|X)d\theta. \tag{3.5}$$

The posterior predictive distribution is not only useful for making single point predictions. The posterior predictive variance of each new data point can be used as a measure of uncertainty in new predictions as illustrated in Figure 3.3 using a simple example of Bayesian linear regression. We can also generate new data from the posterior predictive and see whether the generated data displays the same behaviour as the observed data (known as a Posterior Predictive Check), which indicates to us how well we have chosen our prior.

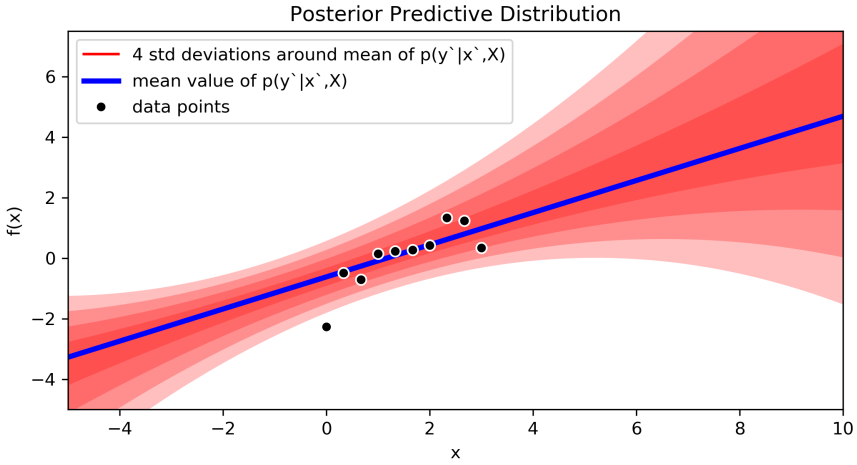


Figure 3.3: Example of posterior predictive standard deviation used as uncertainty measure.

Note, however, that the calculation of Eq. 3.5 looks similar to the evidence of the data. The only difference is instead of using the prior in the integral, we use the posterior. This once again could lead to computational difficulties and may require us to approximate the posterior predictive distribution as well. Although approximation may seem like a compromise, it has become common practice among Bayesian analysts to employ powerful algorithms such as Markov Chain Monte Carlo (MCMC) or variational inference capable of producing accurate approximations of these posterior distributions.

3.5 Summary

Bayesian inference is a time-tested method which allows us to extract additional information from the data's distribution, provide robust regularisation from the prior (usually through a non-informative prior) which prevents over-fitting to data and allow us to measure the uncertainty around a prediction. It also allows us to integrate any available expert knowledge we have on the unknown parameters through the use of an informative prior. For model evaluation, the posterior distribution allows us to perform a posterior predictive check (ppc) where we generate additional data from the posterior and evaluate its behaviour in comparison to the training data. To illustrate these advantages, the next chapter introduces classical linear methods and we discuss how to approach the model from a Bayesian perspective.

Chapter 4

Linear classifier

Before introducing Neural Networks (NNs) it would be useful to first review some of the linear model structures present in NN architecture. This dissertation focuses on models used for classification tasks and therefore logistic regression would be an appropriate starting point.

4.1 Logistic regression

Unlike linear regression which is used to model a continuous output variable Y for a given continuous input X, we use logistic regression to learn the probabilities of a categorical output Y for a given continuous input X. Although linear regression can be used in an environment where the response variable is defined in the unit interval $[0, 1]$ for binary classification, it is not advised since the linear model will produce values outside of this interval when extrapolating.

To account for this, logistic regression applies a sigmoid function $\Phi : \mathbb{R} \rightarrow [0, 1]$ shown in Eq. 4.2 on the dot product between the model parameters and input data (also known as the link function). Logistic regression is considered a general linear model (GLM) since the output of the model is not directly related to the link function. Instead, the link function is first passed through a non-linear function, similar to an NN with one hidden layer containing a single node. However, despite the non-linearity present in the design of a logistic regression, the model only works well on linearly-separable data since the model produces a linear decision boundary from the estimated probabilities. The importance of this assumption will be highlighted in the example to follow. For ease of illustration we will continue the chapter in a binary-classification environment.

Consider an input matrix $X = \{\underline{x_1}, \underline{x_2}, ..., \underline{x_N}\}$, where each $\underline{x_i} \in \mathbb{R}^p$ and vector of response variables

$\underline{y} \in \{0, 1\}$, indicating which class each observation belongs to. We create a linear model of the log-odds of observing the event $(Y = 1)$ for any input from X as follows:

$$log\left(\frac{P(Y = 1|X = \underline{x})}{1 - P(Y = 1|X = \underline{x})}\right) = \underline{w}^\intercal \underline{x}, \tag{4.1}$$

where $\underline{w}$ is a vector of weight parameters (also called regression coefficients) with dimension p.

The probability $P(Y = 1|X = \underline{x})$ can easily be derived algebraically:

$$\frac{P(Y = 1|X = \underline{x})}{1 - P(Y = 1|X = \underline{x})} = e^{\underline{w}^\intercal \underline{x}}$$

by adding 1 to each side of the equation we get

$$\begin{aligned}
\frac{P(Y = 1|X = \underline{x})}{1 - P(Y = 1|X = \underline{x})} + 1 &= e^{\underline{w}^\intercal \underline{x}} + 1 \\
\frac{1}{1 - P(Y = 1|X = \underline{x})} &= e^{\underline{w}^\intercal \underline{x}} + 1 \\
P(Y = 1|X = \underline{x}) &= 1 - \frac{1}{e^{\underline{w}^\intercal \underline{x}} + 1} \\
&= \frac{e^{\underline{w}^\intercal \underline{x}}}{e^{\underline{w}^\intercal \underline{x}} + 1}
\end{aligned}$$

by multiplying both sides of the equation with $\frac{e^{-\underline{w}^\intercal \underline{x}}}{e^{-\underline{w}^\intercal \underline{x}}}$, we are left with

$$P(Y = 1|X = \underline{x}) = \frac{1}{1 + e^{-\underline{w}^\intercal \underline{x}}}. \tag{4.2}$$

Eq. 4.2 shows that the class probabilities are expressed in terms of a linear function of the input data X, passed through a sigmoid function.

Fitting a logistic regression model to data is usually achieved through maximum likelihood estimation. Since we are in a binary classification environment, the distribution of response variable Y can be assumed to be Bernoulli with parameter $p = P(Y = 1|X = \underline{x})$. For notational simplicity and to emphasise dependence on the parameters $\underline{w}$, let:

$$\begin{aligned}
P(Y = 1|X = \underline{x}) &= p(\underline{x}; \underline{w}), \text{ and} \\
P(Y = 0|X = \underline{x}) &= 1 - p(\underline{x}; \underline{w}).
\end{aligned} \tag{4.3}$$

Using the Bernoulli probability function $f(x) = p^x(1 - p)^{1-x}$, the log-likelihood for weight parameters $\underline{w}$

can be expressed as follows:

$$\begin{aligned}\ell(\underline{w}) &= log\Bigg[\prod_{i=1}^{N} p(\underline{x}_i;\underline{w})^{y_i}(1-p(\underline{x}_i;\underline{w}))^{1-y_i}\Bigg] \\ &= \sum_{i=1}^{N}\Big[y_i log(p(\underline{x}_i;\underline{w})) + (1-y_i)log(1-p(\underline{x}_i;\underline{w}))\Big] \\ &= \sum_{i=1}^{N}\Big[y_i\underline{w}^\intercal\underline{x}_i - log(e^{\underline{w}^\intercal\underline{x}_i}+1)\Big] \end{aligned} \tag{4.4}$$

We maximize the log-likelihood function by setting its first derivative equal to zero:

$$\frac{\partial\ell(\underline{w})}{\partial\underline{w}} = \sum_{i=1}^{N}\underline{x}_i\big(y_i - p(\underline{x}_i;\underline{w})\big) = 0 \tag{4.5}$$

Since a closed form expression for $\underline{w}$ which maximizes Eq. 4.5 cannot be found analytically, we need to use numerical approximation. In the case of logistic regression, the log-likelihood function is known to be concave. Therefore, the optimal point can be approximated through an iterative method called the Newton-Raphson algorithm [44]. To that end, we need to calculate the second-derivative (known as the Hessian matrix):

$$\frac{\partial^2\ell(\underline{w})}{\partial\underline{w}\partial\underline{w}^\intercal} = -\sum_{i=1}^{N}\underline{x}_i\underline{x}_i^\intercal p(\underline{x}_i;\underline{w})(1-p(\underline{x}_i;\underline{w})).$$

Algorithm 3 Newton-Raphson algorithm

1. Choose initial values for weight parameters $\underline{w}$ (zero is usually a good choice [44])
2. Using current weight parameters $\underline{w}$, calculate $p(\underline{x}_i;\underline{w})$ for each $\underline{x}_i \in X$
3. Update current weights $\underline{w}$ to $\underline{w}^{new} = \underline{w} - \left(\frac{\partial\ell(\underline{w})}{\partial\underline{w}\partial\underline{w}^\intercal}\right)^{-1}\frac{\partial\ell(\underline{w})}{\partial\underline{w}}$
4. Steps 2 and 3 are repeated until convergence. Convergence is met when the difference between $\underline{w}$ and $\underline{w}^{new}$ becomes smaller than a user-specified threshold.

Algorithm 3 summarises the steps taken when using the Newton-Raphson method. For concave log-likelihood functions such as in logistic regression, convergence is almost always guaranteed. In cases where the optimal point estimates are erroneously passed over, a suitable fix would be to decrease the update increment in Step 3 [44].

4.1.1 Example

As a simple classification example, let $\mathcal{D} = \{\underline{x},\underline{y}\} = \{(x_1,y_1),(x_2,y_2),...,(x_{200},y_{200})\}$ be a collection of observed pairs where each $x_i \in \mathbb{R}$ is an explanatory variable (i.i.d.) with corresponding binary response

variable $y_i \sim Bernoulli(p_i)$. Using a set of parameters $\underline{w} = [w_0, w_1] = [1, 2]$, the corresponding Bernoulli parameter p_i was generated using the sigmoid function $p_i = \frac{1}{1+exp\{w_0+w_1 x_i\}}$. A random sample of 150 observations were used as a training set, leaving the remaining 50 observations as a test set.

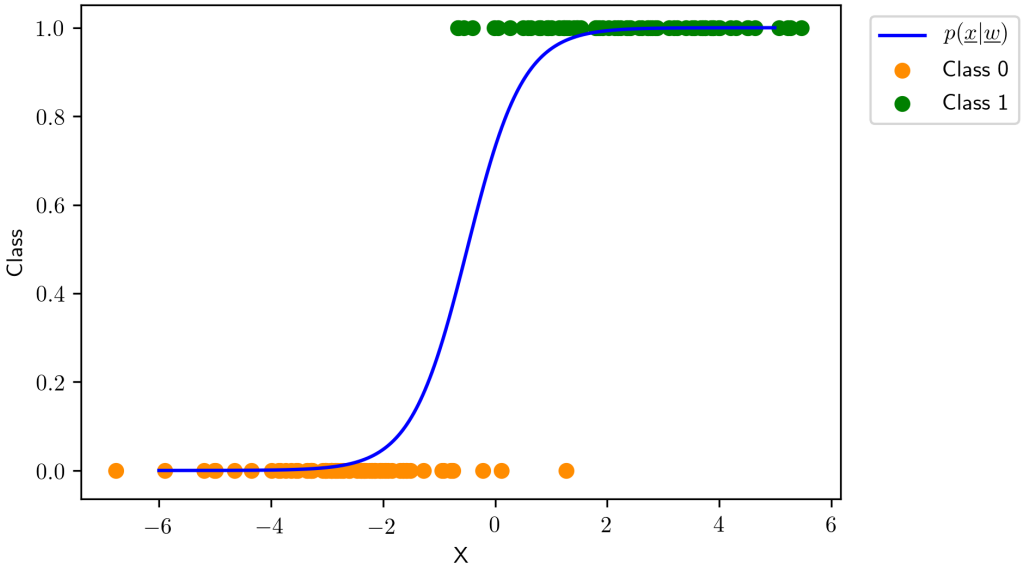


Figure 4.1: Training data with theoretical probabilities using the true parameter values.

The model was trained in Python using the Newton Raphson algorithm shown in Algorithm 3. With initial values set to zero, the log-likelihood shown in Eq. 4.4 converged after 500 iterations using Algorithm 3 with a convergence criteria of 5×10^{-4} and produced estimates $\underline{\hat{w}} = [0.84, 2.47]$.

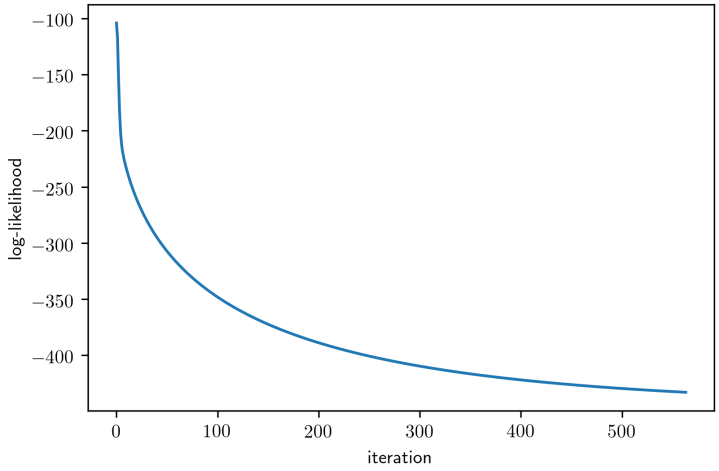


Figure 4.2: Convergence of log-likelihood.

For each observation in the test set, the model yielded a predicted class probability $\hat{p_i} = \frac{1}{1+exp\{\hat{w_0}+\hat{w_1}x_i\}}$. To produce a categorical output for classification, a decision boundary at 0.5 was chosen, i.e.

$$\hat{y}_i = \begin{cases} 1 & \text{if } \hat{p} \geq 0.5 \\ 0 & \text{if } \hat{p} < 0.5 \end{cases}$$

The model scored a classification accuracy of 94% on the small test set containing 50 observations. Despite the high accuracy score, Figure 4.3 shows that the model did not generalise well to the outliers from both classes. For a logistic regression, any point that violates the assumption of linear separability may cause the model to over-fit to data or perform poorly on future data.

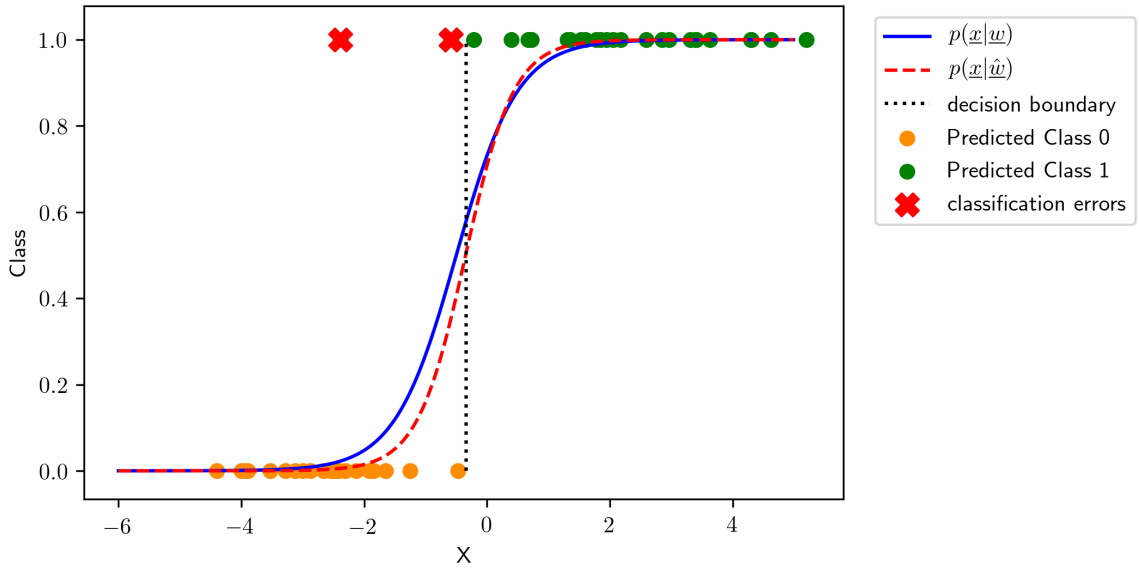


Figure 4.3: Predicted class probabilities and class assignments on test data.

If we inspect the predicted probabilities in Figure 4.3, it becomes clear that observations close to the decision boundary, i.e. where $p(x_i|\underline{\hat{w}})$ is close to 0.5 are assigned to a class with high uncertainty. In many practical scenarios such as medical diagnosis [1] or autonomous driving [45], important decisions rely on output from more complex regression or classification models. In such cases, understanding the uncertainty around the output of a model could prove useful. Moreover, instead of a single point-estimate for parameter or output values, addressing an entire probability distribution of such values could provide important insight. In the next section we consider the logistic regression model approached from a probabilistic perspective in order to capture uncertainty in a simple classification problem.

4.2 Bayesian logistic regression

The Bayesian approach to logistic regression allows us to compute the full posterior distribution over the model parameters and gain insight into how uncertain we are about making predictions given the posterior predictive distribution.

In Bayesian logistic regression, we derive the posterior distribution of the weight parameters $\underline{w}$. Following the example from the previous section, we choose our prior distribution over the weight parameters as i.i.d $P(\underline{w}) = \mathcal{N}(\underline{0}, \Sigma_0)$ for some covariance matrix Σ_0. For an uninformative prior, the covariance matrix can be chosen such that $P(\underline{w})$ allows for a wide range of values. The assumption of independence between weight parameters simplifies the calculation of the posterior distribution at the cost of capturing covariance between weights within the model. By selecting a Gaussian prior with zero mean, we may be more immune to over-fitting since we enforce our initial belief that the weight parameters are centred around zero before updating the posterior distribution with the likelihood. The same result is achieved through L1 or L2 regularisation in a non-Bayesian context. However, the disadvantage of choosing a Gaussian prior is that we force the distribution of the weight parameters to a symmetric bell-shape in the absence of more adequate prior knowledge.

The likelihood function $P(\underline{y}|X, \underline{w})$ constitutes the probability of observing the outcome of the data given the current parameter setting. Since the response variable Y only takes two possible values in this context, we assume that Y follows a Bernoulli distribution with parameter $P(Y = 1|X = \underline{x})$ as defined in Eq. 4.3.

Using the the result from Eq. 4.2 and the notation introduced by Eq. 4.3, the likelihood function takes the following form:

$$\begin{aligned} P(\underline{y}|X, \underline{w}) &= \prod_{i=1}^{N} p(\underline{x_i}) \\ &= \prod_{i=1}^{N} p(\underline{x_i})^{y_i} (1 - p(\underline{x_i}))^{1-y_i} \\ &= \prod_{i=1}^{N} \left[\frac{1}{1 + e^{-\underline{w}^\intercal \underline{x_i}}}\right]^{y_i} \left[\frac{e^{-\underline{w}^\intercal \underline{x_i}}}{1 + e^{-\underline{w}^\intercal \underline{x_i}}}\right]^{1-y_i}. \end{aligned}$$

The posterior distribution over the weights can therefore be expressed as:

$$\begin{aligned} P(\underline{w}|X, \underline{y}) &= \frac{P(\underline{y}|X, \underline{w})P(\underline{w})}{\int ... \int P(\underline{y}|X, \underline{w})P(\underline{w})d\underline{w}} \\ &\propto \prod_{i=1}^{N} \left[\frac{1}{1 + e^{-\underline{w}^\intercal \underline{x_i}}}\right]^{y_i} \left[\frac{e^{-\underline{w}^\intercal \underline{x_i}}}{1 + e^{-\underline{w}^\intercal \underline{x_i}}}\right]^{1-y_i} \times \prod_{j=1}^{p} \frac{1}{\sqrt{2\pi}\sigma_j} exp\Big\{ -\frac{1}{2}\Big(\frac{w_j}{\sigma_j}\Big)^2 \Big\}. \end{aligned} \tag{4.6}$$

To calculate the complete posterior distribution we require the normalisation term $\int ... \int P(\underline{y}|X, \underline{w})P(\underline{w})d\underline{w}$. For models where the weight vector is defined in higher dimensions, the integral becomes too complex to calculate analytically since we marginalize over all the possible parameter settings. This can be solved by using approximation methods such as Markov Chain Monte Carlo discussed in Section 5.1. However, since we use a simple model with only 2 parameters (w_0 and w_1) in this example, we are able to express the normalisation term with a double integral:

$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\left(\prod_{i=1}^{N}\left[\frac{1}{1+e^{-\underline{w}^{\intercal}\underline{x}_i}}\right]^{y_i}\left[\frac{e^{-\underline{w}^{\intercal}\underline{x}_i}}{1+e^{-\underline{w}^{\intercal}\underline{x}_i}}\right]^{1-y_i}\times\prod_{j=1}^{p}\frac{1}{\sqrt{2\pi\sigma_j}}exp\left\{-\frac{1}{2}\left(\frac{w_j}{\sigma_j}\right)^2\right\}\right)d\underline{w}, \tag{4.7}$$

where p is defined as the number of weight parameters used by the model. The double integral in Eq. 4.7 can be approximated using the SciPy package in Python [46] and using this result we are able to derive the full posterior of the weight parameters $\underline{w} = \{w_0, w_1\}$ used in the logistic regression example. The joint posterior distribution of $\underline{w}$ is shown in Figure 4.4.

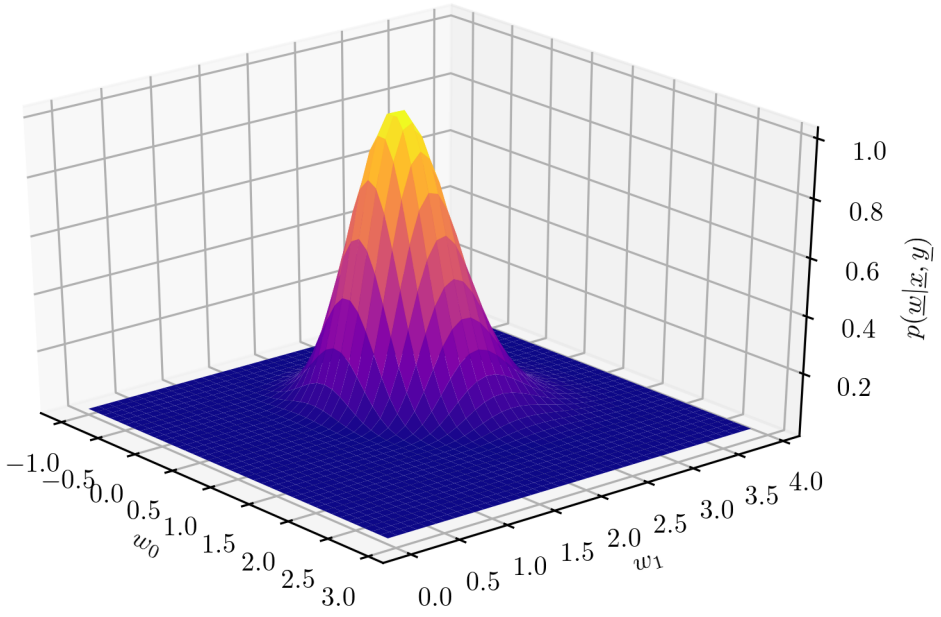
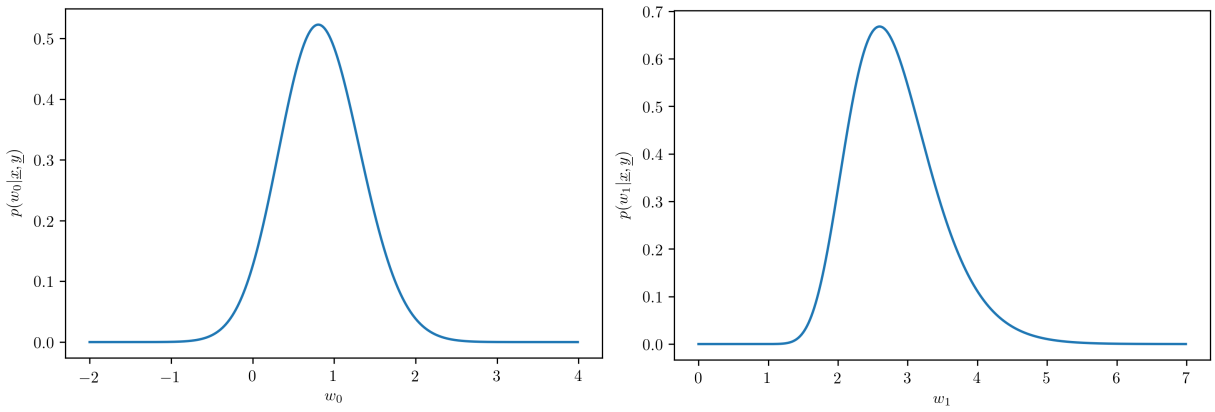


Figure 4.4: Joint posterior distribution of weight parameters for logistic regression.

Using the joint posterior we derive the marginal posterior distribution of each parameter as illustrated in Figure 4.5.



(a) w_0 (b) w_1

Figure 4.5: Marginal posterior distributions of weight parameters.

4.2.1 Monte Carlo Integration

The posterior predictive distribution allows us not only to make single predictions for new data using the mean or median of the posterior, but to create confidence intervals and express our uncertainty for predicted response values. For a new input $\underline{x}'$, the posterior predictive distribution can be expressed as follows:

$$p(y'|\underline{x}', X, \underline{y}) = \int p(y'|\underline{x}', \underline{w}) p(\underline{w}|X, \underline{y}) d\underline{w} \tag{4.8}$$

Similar to the normalisation term, we marginalize over all the weight parameters when calculating the posterior predictive distribution. However, since we already have the posterior distribution available we are able to sample parameter values directly from the posterior in order to approximate the posterior predictive and generate predictions for new data using Monte Carlo integration.

To prove that Monte Carlo integration can approximate the integral in Eq. 4.8, define the weight parameters as our random variable of interest W governed by the probability distribution $p(W) = p(\underline{w}|X, \underline{y})$, i.e. the posterior distribution. Our function of interest is the product inside the integral

$$f(W) = p(y'|\underline{x}', \underline{w}) p(\underline{w}|X, \underline{y}).$$

We define the Monte Carlo estimator as

$$\begin{aligned} F_N &= \frac{1}{N}\sum_{i=1}^{N}\frac{f(W_i)}{p(W_i)} \\ &= \frac{1}{N}\sum_{i=1}^{N}\frac{p(y'|\underline{x}', \underline{w}_i)p(\underline{w}_i|X, \underline{y})}{p(\underline{w}_i|X, \underline{y})} \\ &= \frac{1}{N}\sum_{i=1}^{N} p(y'|\underline{x}', \underline{w}_i) \end{aligned}$$

Monte Carlo integration states that by the law of large numbers, the expected value of the estimator F_N approaches the integral under consideration for a sample size N as follows:

$$\begin{aligned} \mathbb{E}[F_N] &= \mathbb{E}\left[\frac{1}{N}\sum_{i=1}^{N}\frac{f(W_i)}{p(W_i)}\right] \\ &= \frac{1}{N}\,\mathbb{E}\left[\sum_{i=1}^{N}\frac{p(y'|\underline{x}', \underline{w}_i)p(\underline{w}_i|X, \underline{y})}{p(\underline{w}_i|X, \underline{y})}\right] \\ &= \frac{1}{N}\sum_{i=1}^{N}\mathbb{E}\left[p(y'|\underline{x}', \underline{w}_i)\right] \\ &= \frac{1}{N}\sum_{i=1}^{N}\int p(y'|\underline{x}', \underline{w})p(\underline{w}|X, \underline{y})d\underline{w} \\ &= \frac{1}{N}\sum_{i=1}^{N} p(y'|\underline{x}', X, \underline{y}) \qquad \text{(using Eq. 4.8)} \\ &= p(y'|\underline{x}', X, \underline{y}) \qquad (4.9) \end{aligned}$$

Given a new observation $\underline{x}'$, the following steps are taken to produce an estimate of $p(y'|\underline{x}', X, \underline{y})$:

1. Generate a large sample $W^s = \{\underline{w}_1^s, \underline{w}_2^s, ..., \underline{w}_{10000}^s\}$ from $p(\underline{w}|\underline{x}, \underline{y})$
2. Calculate $p(y'|\underline{x}', \underline{w}_j^s) = \frac{1}{1+e^{-\underline{x}\underline{w}_j^s}}$ for every $\underline{w}_j^s \in W^s$
3. Calculate $\frac{1}{N}\sum_{j=1}^{N} p(y'|\underline{x}', \underline{w}_j^s)$

To illustrate the algorithm, Figure 4.6 shows a distribution of predicted class probabilities for a single input value $x' = -0.5$. This value was chosen to be close to the linear decision boundary such that its class prediction contains more uncertainty, as observed in spread of the histogram.

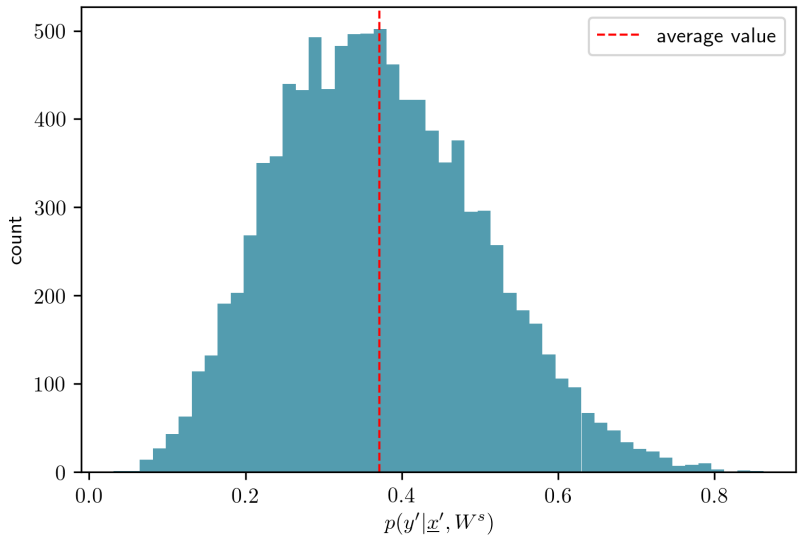


Figure 4.6: Histogram of posterior predictive probabilities for a single input value

This produces approximate samples from the posterior predictive distribution and allows us to capture important characteristics of model predictions such as the mean and standard deviation.

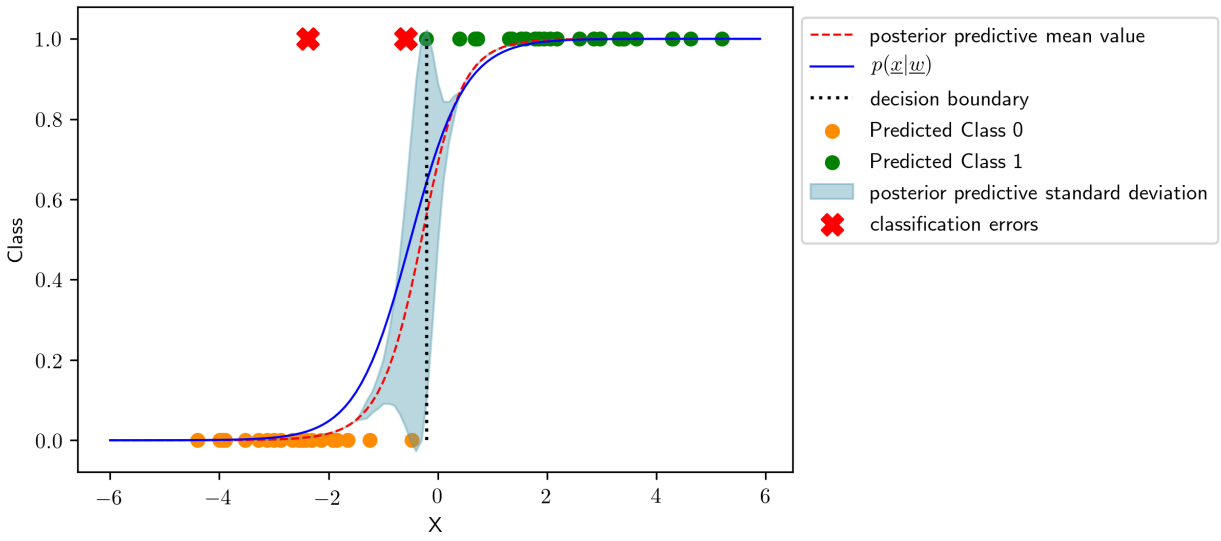


Figure 4.7: Model predictions obtained from posterior predictive samples.

As discussed in the previous section, we are interested in the measure of uncertainty when making class predictions from the model output. This is achievable by using the standard deviation of each prediction obtained from the posterior predictive distribution as shown in Figure 4.7. As we observe data closer to

the decision boundary, the standard deviation increases rapidly since the linear boundary isn't capable of separating the data perfectly.

Compared to the frequentist approach in the previous section, the model shows similar performance. However, by using a Bayesian approach we have gained additional information about the problem such as the distribution of the model parameters and the uncertainty of prediction. For a simple example such as this, we were able to easily calculate the posterior distribution and use Monte Carlo integration to predict class labels of new data. For more complex models, more advanced methods are required to approximate the posterior distribution. In the next section we will expand the concept of Monte Carlo sampling to Markov Chain Monte Carlo (MCMC) and introduce an alternative method called Variational Inference (VI).

Chapter 5

Approximation Methods

Given the complex structure of deep learning models, performing Bayesian inference on such models would require us to use powerful algorithms to approximate the posterior distribution of the unknown parameter values. As shown in the previous chapters, the posterior distribution contains an analytically intractable integral for most complex models where conjugate priors are few and far between. The goal of this chapter is to introduce two different families of approximation algorithms. Their approximation of the posterior as well as scalability to large sets of data or model complexity will be discussed with a simple Bayesian logistic regression example at the end of each section. These methods will also be used for more complex problems in the next chapter when we formally introduce Neural Networks.

5.1 Markov Chain Monte Carlo

Monte Carlo simulation is the method of using random sampling in order to infer characteristics about a population or probability distribution. Based on Monte Carlo simulation as shown in Section 4.2.1, Markov Chain Monte Carlo (MCMC) is a large class of sampling algorithms which can be used to solve integration and optimization problems defined in spaces of high dimensions. It has been shown that MCMC methods produce exact samples from the target density/posterior distribution [36] after enough samples have been drawn from the Markov Chain to reach its stationary distribution. This result has made MCMC a popular tool and area of research among statisticians when using methods such as Bayesian inference for complex problems. One caveat of MCMC is that computational performance may suffer when using massive sets of data or extremely complex models. A more scalable solution called variational inference may be preferable in such circumstances and will be discussed in Section 5.2. In this section we investigate a widely used MCMC algorithm called Metropolis-Hastings, show how it specialises to

Gibbs-sampling under certain conditions and illustrate the sampling method with an example.

5.1.1 The Metropolis-Hastings algorithm

The Metropolis-Hastings algorithm, which was named one of the top 10 most important algorithms of the 20th century by the journal Computing in Science and Engineering [47] belongs to the family of MCMC algorithms. Suppose that our goal is to sample from some probability distribution p^* called the target distribution, but doing so is not computationally or analytically feasible. For example, when calculating the posterior distribution over parameters in a Bayesian setting, the normalisation term requires us to integrate over all the possible parameter values. Such an integral is analytically intractable in high dimensions and needs to be approximated.

The Metropolis-Hastings algorithm allows us to approximate p^* through some form of a random walk, where each point is chosen in proportion to the probability associated with it.

Let $\{X(t) : t \geq 0\}$ be a Markov chain defined on a state space S with the following properties:

- For every state there is a positive probability of moving to any other state defined in S, i.e. the chain is irreducible.
- The Markov Chain is aperiodic, meaning that the chain does not become trapped in cycles.

Given the above properties, the Markov Chain is called ergodic. Another important property of the above mentioned Markov Chain is that it is stationary with respect to some distribution π. This means that for large t, if $X(t) \sim \pi(x)$ then $X(t+1) \sim \pi(x)$ and so the chain converges in distribution to the distribution π.

The Metropolis-Hastings algorithm shown in Algorithm 4 generates the states of this Markov chain and upon convergence to its stationary distribution allows us to sample from the target distribution p^*. The number of iterations needed for the Markov Chain to reach this stationary distribution is called the burn-in and varies depending on the initial values chosen for the algorithm.

For each state $X(t)$, the algorithm uses a conditional distribution $q(X(t+1)|X(t))$ called a proposal density to generate a new candidate state $X(t+1)$. For this we use a Gaussian distribution, i.e. $q(X(t+1)|X(t)) \sim \mathcal{N}(X(t+1)|X(t) = x, \Sigma)$ although alternative probability distributions can be selected to suit the problem at hand.

Algorithm 4 Metropolis - Hastings

1. Pick an initial "guess" value, x_0.
2. For i = 0,1,2,...
   - Sample $x' \sim q(x'|x_i)$
   - Compute $\mathcal{A}(x_i, x') = \frac{p^*(x')q(x_i|x')}{p^*(x_i)q(x'|x_i)}$
   - Let $r = min(1, \mathcal{A}(x_i, x'))$
   - Sample $u \sim \mathcal{U}(0, 1)$
   - Set new sample to

$$x_{i+1} = \begin{cases} x' & \text{if } u < r \\ x_i & \text{if } u \geq r \end{cases}$$

Since we are using the algorithm in this dissertation for Bayesian inference, the target density p^* refers to the posterior distribution over the parameters. However, the normalisation term used for the posterior distribution is not needed for this approximation since the fraction found in $\mathcal{A}(x_i, x')$ cancels it out. Therefore, all we need in order to compute p^* is the product of the likelihood and the prior. For example, in a Bayesian logistic regression problem such as in the previous section we only need the expression

$$p^* = \prod_{i=1}^{N} \left[\frac{1}{1+e^{-\underline{w}^\intercal \underline{x_i}}}\right]^{y_i} \left[\frac{e^{-\underline{w}^\intercal \underline{x_i}}}{1+e^{-\underline{w}^\intercal \underline{x_i}}}\right]^{1-y_i} \times \prod_{j=1}^{K} \frac{1}{\sqrt{2\pi\sigma_j}} exp\Big\{ -\frac{1}{2}\Big(\frac{w_j}{\sigma_j}\Big)^2 \Big\},$$

where K is defined as the number of unknown parameters used in the model.

5.1.2 Example

To illustrate the Metropolis-Hastings algorithm, we consider again Bayesian logistic regression. In this example, however, we include an additional parameter in the model such that the evidence in the posterior, i.e.

$$\int ... \int P(\underline{y}|X, \underline{w})P(\underline{w})d\underline{w}$$

becomes impractical to compute directly. Using notation introduced by Eq. 4.3, the model is defined as follows:

$$p(\underline{x}|\underline{w}) = \frac{1}{1+e^{-\underline{w}^\intercal \underline{x}}},$$

where $\underline{w} = \{w_0, w_1, w_2\}$, w_0 serving as an intercept paramer. We generate a set of classification data $\mathcal{D} = \{X, \underline{y}\}$ such that $X = \{\underline{x_1}, \underline{x_2}\}$ from the Scikit-Learn Python package [17] with a binary response variable $\underline{y}$ separating 2 clusters within the data as shown in Figure 5.1.

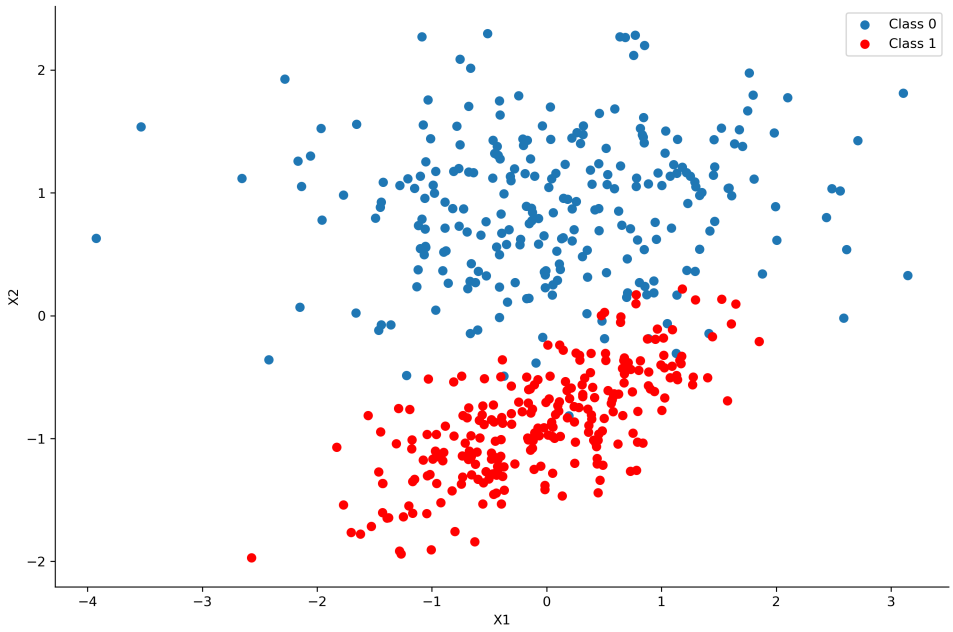


Figure 5.1: Binary classification training data.

To approximate the posterior distribution, we use the Metropolis-Hastings algorithm with a proposal density $q(X(t+1)|X(t)) \sim \mathcal{N}(X(t+1)|X(t) = x, \Sigma)$ and all initial values set to zero. The first 500 iterations of the algorithm were discarded as burn-in and the Markov Chain was set to generate 10000 steps following the initial discarded values. A trace-plot of the Markov Chain after burn-in and corresponding distribution of the weight parameters is shown in Figure 5.3

By visual inspection, the trace plot in Figure 5.3 indicates that the Markov Chain has approached a stationary distribution which will be used as an approximate posterior distribution of $\underline{w}$ to sample from. Using the estimated posterior, we generate samples from the posterior predictive distribution for a test dataset. The predicted class labels and a heat map of the posterior predictive mean value is shown in Figure 5.2

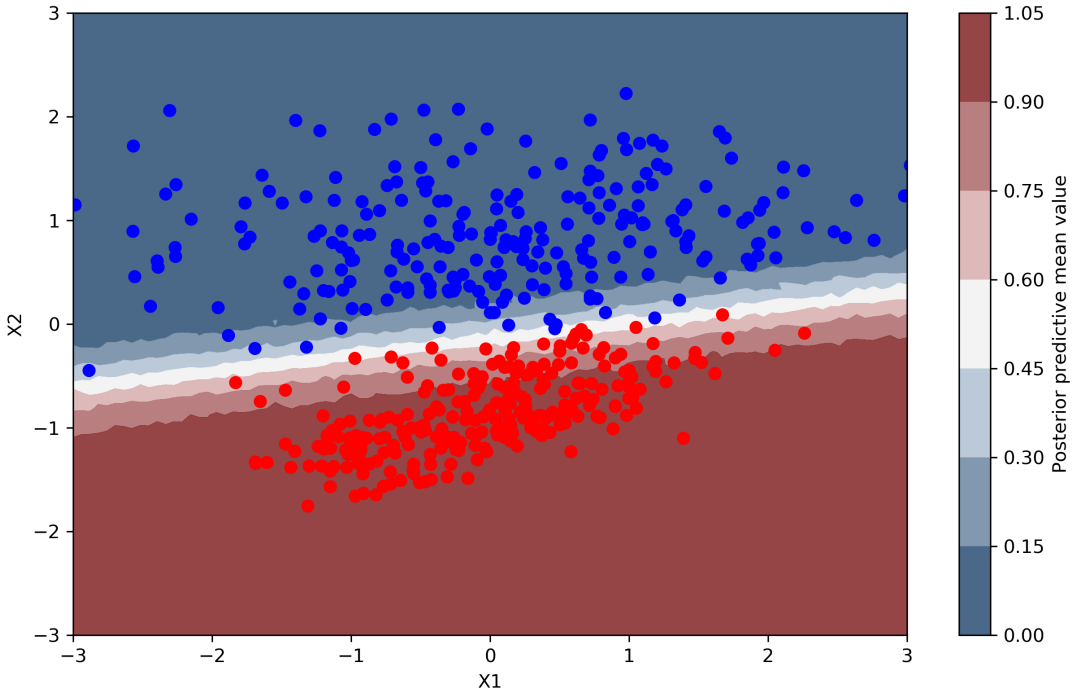


Figure 5.2: Class label prediction and posterior predictive distribution.

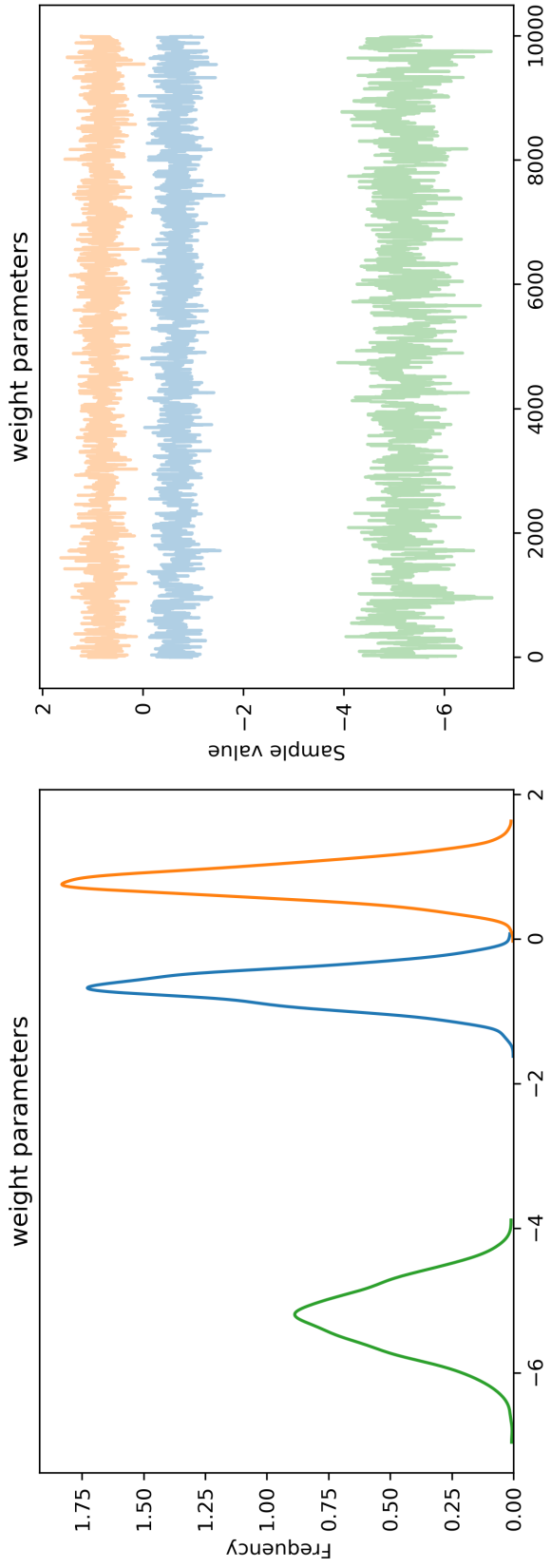


Figure 5.3: Posterior distributions of weight parameters estimated by Markov Chains.

The above predictions and parameter estimates are easily obtainable by logistic regression using maximum likelihood estimation. Since we are interested in obtaining a measure of uncertainty in prediction, we use the standard deviation in the posterior predictive distribution to give us additional insight into the model. Figure 5.4 illustrates the uncertainty in prediction of classes given newly observed data. As expected, in addition to low class probabilities of observations close to the linear coundary, the uncertainty of these probabilities are considerably higher than observations further away from the line.

The output probability such as shown in Figure 5.2 is often erroneously interpreted as model confidence. For a data point with predicted class probability $P(Y = 1|X = \underline{x}) = 0.49$, a user would assume that the predicted class label should be 0. However, for slightly different parameter values (caused by variation in the data) the predicted class probability of the observations close to the linear boundary may change and produce a different predicted class label. Therefore, in cases where we have limited training data, the logistic model may not generalise well to irregular patterns in the data and incorrectly predict the classes of future data points in areas of high uncertainty.

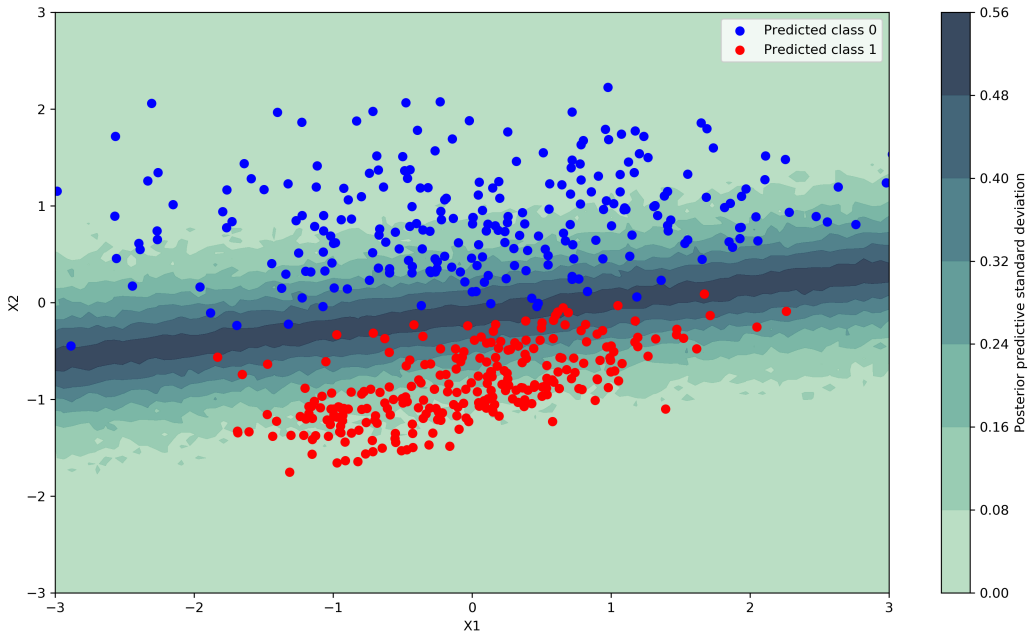


Figure 5.4: Posterior predictive standard deviation as uncertainty measure.

5.1.3 Gibbs sampling

The Gibbs sampler is a special case of the Metropolis-Hastings algorithm in that the values sampled from the proposal density are always accepted, i.e. $r = 1$ using the notation from the previous section. Furthermore, the random variables are sampled from univariate conditional distributions, conditional upon all but one of the variables at a time. Therefore, the proposal distribution introduced in the previous section is the posterior probability of the variable.

Suppose that we have a vector of model parameters $\underline{w} = \{w_1, w_2, ..., w_p\}$ for which we would like to find the joint posterior distribution $p(\underline{w}|\mathcal{D})$ where $\mathcal{D}$ is our observed data. If we are able to draw samples from each parameter's conditional distribution

$$p(w_1|w_2, w_3, ..., w_p, \mathcal{D})$$
$$p(w_2|w_1, w_3, ..., w_p, \mathcal{D})$$
$$\dots$$
$$p(w_p|w_1, w_3, ..., w_{p-1}, \mathcal{D}),$$

then Gibbs sampling lets us construct a Markov Chain from the above distributions with a stationary distribution equal to the posterior distribution, similar to Metropolis-Hastings. Once each variable has been sampled from its conditional distribution, we have completed one cycle of the Gibbs sampler.

The advantage of Gibbs sampling is that we are able to simplify a higher-dimensional problem into a univariate one by drawing a single variable at a time. This is advantageous since sampling from high dimensional joint distribution functions is often harder than univariate conditional distributions.

The disadvantage of Gibbs sampling is that expressions for the above conditional distributions are not always available. In addition, if the expressions are available, Gibbs sampling is only useful if it is computationally efficient to draw samples from these distributions. It is also possible for consecutive samples to be strongly correlated, causing convergence to the stationary distribution to take a large number of steps. The Gibbs sampler is therefore preferable over the Metropolis-Hastings algorithm in cases where it is necessary (and also relatively easy) to reduce the dimensionality of the target distribution into univariate conditional probabilities.

5.2 Variational Inference

Variational Inference (VI) and MCMC are two very different techniques that are widely used to solve the same problem. In this case, we are interested in approximating a posterior distribution over some unknown parameters in the data when the evidence of the data, $p(\mathcal{D})$, is computationally intractable. Unlike MCMC, VI approximates the target distribution through means of optimization. In addition to VI's scalability to massive sets of data and complex models, another advantage is that convergence is easily measured, as opposed to MCMC where the required number of samples before convergence is often not known beforehand. This makes VI a better choice in some circumstances. However, despite its usefulness, VI is not as widely used and understood by statisticians as its counterpart MCMC. This section will serve as a short introduction to the theory behind VI and provide a practical illustration in a Bayesian setting.

5.2.1 Shannon Entropy

In variational inference we specify a family of probability distributions $\mathcal{Q}$ over a random variable X, where our aim is to approximate the "true" distribution $p(X)$ also called the target distribution. Each distribution $q(X) \in \mathcal{Q}$ is a candidate approximation to $p(X)$ (usually with a simpler structure), where we select the candidate that minimizes a distance measure called the Kullback-Leibler (KL) divergence [48], also known as the Relative Entropy. The KL divergence uses entropy to measure how "close" our candidate distribution is to the target probability distribution, based on the distribution we choose for q as well as its parameters settings. Before further investigating KL divergence, we briefly review Information Theory and the role of entropy in VI.

Suppose that the probability distribution $p(X)$ over a set of events $\mathcal{D} = \{X_1, X_2, ..., X_n\}$ is known. In information theory, we can measure the amount of information of observing an event $X_i \in \mathcal{D}$ as

$$I(X_i) = -log(p(x_i)).$$

From the above expression it is clear that observing events with high probability holds less information (or is less "surprising") than observing an event with low probability. Using the above information measure, the average information found in $\mathcal{D}$ is simply the expected value of $I(X_i)$ with respect to the probability distribution $p(X)$:

$$\begin{aligned} H_{p(X)} &= \mathop{\mathbb{E}}_{p(X)}[I(X_i)] \\ &= -\mathop{\mathbb{E}}_{p(X)}[log(p(x_i))] \end{aligned} \tag{5.1}$$

The average information expressed in Eq. 5.1 is also known as Shannon Entropy (also called Entropy) and is used as a measure of uncertainty found in a probability distribution.

5.2.2 Kullback-Leibler Divergence

We now return to the problem of approximating the posterior distribution over latent variables $\underline{w}$, i.e. $p(\underline{w}|\mathcal{D})$. If we wanted to measure the distance or similarity between a candidate distribution $q(\underline{w})$ and the posterior $p(\underline{w}|\mathcal{D})$, we could use the concept of entropy to acquire a measure called the Relative Entropy, or the KL divergence. The notation used for the the KL divergence with respect to q against p is shown below:

$$\begin{aligned} KL(q(\underline{w})||p(\underline{w}|\mathcal{D})) &= \mathop{\mathbb{E}}_{q(\underline{w})}[log(q(\underline{w}))] - \mathop{\mathbb{E}}_{q(\underline{w})}[log(p(\underline{w}|\mathcal{D}))] \\ &= \mathop{\mathbb{E}}_{q(\underline{w})}[log(q(\underline{w}))] - \mathop{\mathbb{E}}_{q(\underline{w})}[log(p(\underline{w},\mathcal{D}))] + log(p(\mathcal{D})). \end{aligned} \tag{5.2}$$

Note that KL divergence is not symmetric, that is

$$KL(q||p) \neq KL(p||q).$$

Eq. 5.2 shows dependence on the evidence $p(\mathcal{D})$, causing the KL divergence to become computationally intractable, which contradicts the reason we appeal to VI in the first place. Instead, we re-arrange Eq. 5.2 to obtain an alternative objective function called the Evidence Lower Bound (ELBO):

$$\begin{aligned} \mathop{\mathbb{E}}_{q(\underline{w})}[log(p(\underline{w},\mathcal{D}))] - \mathop{\mathbb{E}}_{q(\underline{w})}[log(q(\underline{w}))] &= -KL(q(\underline{w})||p(\underline{w}|\mathcal{D})) + log(p(\mathcal{D})) \\ \mathop{\mathbb{E}}_{q(\underline{w})}[log(p(\underline{w},\mathcal{D}))] - \mathop{\mathbb{E}}_{q(\underline{w})}[log(q(\underline{w}))] &= ELBO(q) \end{aligned}$$

The ELBO can be interpreted as the sum of the negative KL divergence and the log of the evidence, i.e. $log(p(\mathcal{D}))$, which remains constant when optimizing with respect to $q(\underline{w})$. This means that instead of minimizing the KL divergence, we can maximize the ELBO instead to achieve the same result without having to compute the evidence.

To prove that the ELBO is the lower bound of the evidence $p(\mathcal{D})$, recall from Section 3.3 that the evidence can be expressed as

$$p(\mathcal{D}) = \int p(\underline{w},\mathcal{D}) d\underline{w}.$$

It follows that

$$
\begin{aligned}
log(p(\mathcal{D})) &= log \int p(\underline{w}, \mathcal{D}) d\underline{w} \\
&= log \int p(\underline{w}, \mathcal{D}) \frac{q(\underline{w})}{q(\underline{w})} d\underline{w} \\
&= log\Big(\mathop{\mathbb{E}}_{q(\underline{w})} \Big[\frac{p(\underline{w}, \mathcal{D})}{q(\underline{w})} \Big] \Big) \\
&\geq \mathop{\mathbb{E}}_{q(\underline{w})} log\Big(\frac{p(\underline{w}, \mathcal{D})}{q(\underline{w})} \Big) \qquad \text{(through Jensen's inequality [39])} \\
&= \mathop{\mathbb{E}}_{q(\underline{w})} [log(p(\underline{w}, \mathcal{D}))] - \mathop{\mathbb{E}}_{q(\underline{w})} [log(q(\underline{w}))] \\
&= ELBO(q)
\end{aligned}
$$

In the case where $q(\underline{w})$ matches $p(\underline{w}, \mathcal{D})$ exactly, the KL divergence between q and p becomes zero and so $ELBO(q) = p(\mathcal{D})$. The objective is therefore to find a setting of parameters for the variational distribution $q(\underline{w})$ that gives as tight a bound as possible on the evidence.

To summarize, VI approximates the target distribution by selecting a variational distribution which minimises the KL divergence. This optimisation problem is achieved by minimising the ELBO instead of the KL divergence, so that we needn't calculate the intractable normalisation term. Optimisation of the ELBO is satisfied by finding the parameters of the variational distribution $q(\underline{w})$, denoted θ, such that

$$\underline{\theta}^* = \mathop{argmax}_{\underline{\theta}} ELBO(q(\underline{w}; \underline{\theta}))$$

The next section explores a practical method used to find the variational distribution through simplifying assumptions.

5.2.3 Mean Field Variational Inference

Selecting a variational distribution to approximate the posterior plays a critical role in VI. The complexity of the variational distribution determines the complexity of the optimization. Therefore, the goal is to restrict the family $\mathcal{Q}$ of candidate distributions such that each distribution q is computationally tractable but flexible enough to provide a good approximation to the true posterior p.

In mean field variational inference we assume that the latent variables $\underline{w}$ are mutually independent. This allows us to express the variational distribution as seperate factors, i.e.

$$q(\underline{w}) = \prod_{k=1}^{K} q_k(w_k). \tag{5.3}$$

This assumption results in the true posterior p not being in the family of variational distributions since the latent variables in p are dependent on each other. Therefore, mean field variational inference produces a more simple distribution which may be close in KL divergence to the true posterior.

The distribution of each variational factor $q_k(w_k)$ is chosen to suit the single latent variable that it governs and the parameter settings of every factor is chosen to maximize the ELBO. For example, a Gaussian distribution can be chosen for a continuous latent variable where a multinomial distribution would suit a categorical latent variable. More formally, after choosing the distribution for each factor $q_k(w_k)$, we iteratively optimize the ELBO with respect to each factor in turn. Optimization can be performed via a gradient-based method called Coordinate Ascent Variational Inference which we will introduce in the next section.

5.2.4 Coordinate Ascent Variational Inference

A commonly used algorithm for maximizing the ELBO is called Coordinate Ascent Variational Inference (CAVI). With this, we iteratively optimize the ELBO for each variational factor $q_k(w_k)$ while keeping the rest of the factors fixed. To achieve this, we need to deconstruct the ELBO so that we can express it in terms of any single latent variable, while treating the remaining latent variables as constants.

First, by using the chain rule of probability we decompose the joint distribution:

$$p(\underline{w}, \mathcal{D}) = p(\mathcal{D}) \prod_{k=1}^{K} p(w_k | w_{1:k-1}, \mathcal{D}). \tag{5.4}$$

Note that since the latent variables are assumed independent, the ordering of $w_{1:k-1}$ is not of importance. Next, we decompose the entropy of the variational distribution as follows:

$$\mathop{\mathbb{E}}_{q(\underline{w})}[log(q(\underline{w}))] = \sum_{k=1}^{K} \mathop{\mathbb{E}}_{q_k(w_k)}[log(q_k(w_k))]. \tag{5.5}$$

By using Eq. 5.4 and 5.5, we can now express the ELBO as a function of the k^{th} variational factor $q_k(w_k)$ and compress the terms excluding it into a constant:

$$\begin{aligned} ELBO(q) &= \mathop{\mathbb{E}}_{q(\underline{w})}[log(p(\underline{w}, \mathcal{D}))] - \mathop{\mathbb{E}}_{q(\underline{w})}[log(q(\underline{w}))] \\ &= \mathop{\mathbb{E}}_{q(\underline{w})}[log(p(\mathcal{D}) \prod_{k=1}^{K} p(w_k | w_{1:k-1}, \mathcal{D}))] - \sum_{k=1}^{K} \mathop{\mathbb{E}}_{q_k(w_k)}[log(q_k(w_k))] \\ &= \sum_{k=1}^{K} \mathop{\mathbb{E}}_{q(\underline{w})}[log(p(w_k | w_{1:k-1}, \mathcal{D}))] - \mathop{\mathbb{E}}_{q_k(w_k)}[log(q_k(w_k))] + C \end{aligned} \tag{5.6}$$

Since the order of the latent variables is arbitrary, we can treat any specific latent variable w_k as the last variable in the list. As a result we can simplify eq 5.6 even further:

$$ELBO(q_k) = \mathop{\mathbb{E}}_{q(\underline{w})}[log(p(w_k | w_{-k}, \mathcal{D}))] - \mathop{\mathbb{E}}_{q_k(w_k)}[log(q_k(w_k))] + C,$$

where $w_{-k} = \{w_1, w_2, w_3, ..., w_{k-1}, w_{k+1}, ..., w_K\}$.

Finally, we are able to find a value for $q_k(w_k)$ which optimizes the $ELBO(q_k)$ through the following derivation:

$$\begin{aligned} ELBO(q_k) &= \mathop{\mathbb{E}}_{k}[\mathop{\mathbb{E}}_{-k}[log(p(w_k|w_{-k}, \mathcal{D}))]] - \mathop{\mathbb{E}}_{q_k(w_k)}[log(q_k(w_k))] + C \\ &= \int q_k(w_k) \mathop{\mathbb{E}}_{-k}[log(p(w_k|w_{-k}, \mathcal{D}))]dw_k - \int q_k(w_k) log(q_k(w_k))dw_k + C \end{aligned} \tag{5.7}$$

Taking the derivative of Eq. 5.7 with respect to $q_k(w_k)$ and setting equal to zero results in

$$\frac{d}{dq_k(w_k)} ELBO(q_k) = \mathop{\mathbb{E}}_{-k}[log(p(w_k|w_{-k}, \mathcal{D}))] - log(q_k(w_k)) + 1 = 0.$$

Therefore, the optimal value of $q_k(w_k)$ is obtained as follows:

$$\begin{aligned} log(q_k^*(w_k)) &\propto \mathop{\mathbb{E}}_{-k}[log(p(w_k|w_{-k}, \mathcal{D}))] \\ q_k^*(w_k) &\propto exp\{\mathop{\mathbb{E}}_{-k}[log(p(w_k|w_{-k}, \mathcal{D}))]\} \\ &\propto exp\{\mathop{\mathbb{E}}_{-k}[log(p(w_k, w_{-k}, \mathcal{D}))]\}. \end{aligned} \tag{5.8}$$

Expression 5.8 is a result of the evidence $p(\mathcal{D})$ being independent of w_k, causing the posterior $p(w_k|w_{-k}, \mathcal{D})$ to absorb the constant $p(\mathcal{D})$ with respect to w_k and be proportional to the joint distribution $p(w_k, w_{-k}, \mathcal{D})$.

Using result 5.8, we are now able to compute the CAVI algorithm shown in Algorithm 5.

Algorithm 5 Coordinate Ascent Variational Inference

1. Select a variational distribution $q(\underline{w}) = \prod_{k=1}^{K} q_k(w_k)$
2. Choose initial parameter values for each variational factor $q_k(w_k)$
3. Repeat until ELBO converges:
   - For $k = 1$ to K:
     - Calculate $q_k^*(w_k) \propto exp\{\mathbb{E}_{-k}[log(p(w_k, w_{-k}, \mathcal{D}))]\}$
   - Calculate $ELBO(q) = \mathbb{E}_{q(\underline{w})}[log(p(\underline{w}, \mathcal{D}))] - \mathbb{E}_{q(\underline{w})}[log(q(\underline{w}))]$
4. Return $q(w)$

5.2.5 Automatic Differentiation Variational Inference

The CAVI algorithm as derived in the previous section is a useful tool and easy to implement, but only in the case where we have a familiar form of the joint distribution available. This is a statistical convenience

not always encountered in practice. Therefore, we require a more robust method to implement VI to a wider family of posterior distributions.

Automatic Differentiation Variational Inference (ADVI) is an algorithm developed by Kucukelbir et al. [49] which uses a software-driven technique called Automatic Differentiation [50] to optimize the ELBO and approximate the posterior of differentiable probability models. In short, the algorithm applies two transformations to the unknown parameters and approximates their distribution with a variation mean-field Gaussian distribution, which allows the algorithm to iteratively maximise the ELBO using stochastic gradient ascent. The gradients of these models are valid within the support of the prior distribution

$$supp(p(\underline{w})) = \{\underline{w} | \underline{w} \in \mathbb{R}^K \text{ and } p(\underline{w}) > 0\},$$

where K is the dimension of $\underline{w}$ and $p(\underline{w})$ is the prior distribution of $\underline{w}$. Again, we are interested in finding a variational distribution $q(\underline{w}; \underline{\theta})$ where $\underline{\theta}$ constitutes the parameters used by q, such that we maximize the ELBO:

$$ELBO(q) = \mathop{\mathbb{E}}_{q(\underline{w})}[log(p(\underline{w}, \mathcal{D}))] - \mathop{\mathbb{E}}_{q(\underline{w})}[log(q(\underline{w}))] \tag{5.9}$$

for a certain setting of $q(\underline{w}; \underline{\theta})$.

Let T be a differentiable function such that

$$T : supp(p(\underline{w})) \mapsto \mathbb{R}^K.$$

We now define the transformed parameters $\underline{\delta} = T(\underline{w})$. With the newly transformed variables, the support of $\underline{\delta}$ lies in $\mathbb{R}^K$ and the joint posterior density is expressed as

$$p(\underline{\delta}, \mathcal{D}) = p(T^{-1}(\underline{\delta}), \mathcal{D}) |J_{T^{-1}(\underline{\delta})}|,$$

where $|J_{T^{-1}}(\underline{\delta})|$ is the determinant of the Jacobian of the inverse of T, i.e.

$$|J_{T^{-1}}(\underline{\delta})| = \det \begin{vmatrix} \frac{\partial T_1^{-1}}{\partial \delta_1} & \cdots & \frac{\partial T_1^{-1}}{\partial \delta_K} \\ \vdots & \ddots & \vdots \\ \frac{\partial T_K^{-1}}{\partial \delta_1} & \cdots & \frac{\partial T_K^{-1}}{\partial \delta_K} \end{vmatrix}.$$

For a variational posterior approximation of the transformed parameters $\underline{\delta}$ we consider a mean-field Gaussian , $q(\underline{\delta}; \underline{\theta}) \sim \mathcal{N}(\underline{\delta} | \underline{\mu}, diag(\underline{\sigma}^2))$ such that

$$q(\underline{\delta}; \underline{\theta}) \prod_{k=1}^{K} q(\delta_k; \theta_k)$$

where $\underline{\theta}$ encapsulates the Gaussian parameters $\underline{\mu}$ and $\underline{\sigma}^2$. Note that since $\underline{\sigma}^2$ is used as the variance parameter, its support is constrained to $\mathbb{R}^K_{>0}$. To remove the constraint, we apply a logarithm element-wise to the standard deviations $\underline{\sigma}$ and define $\underline{\gamma} = log(\underline{\sigma})$. The variational distribution simply becomes

$$q(\underline{\delta}; \underline{\theta}) = \mathcal{N}(\underline{\delta} | \underline{\mu}, diag(e^{\underline{\gamma}})),$$

where we define $\underline{\theta}$ as $\{\underline{\mu}, \underline{\gamma}\}$ (concatenating $\underline{\mu}$ and $\underline{\gamma}$) such that the support of $\underline{\theta}$ lies in $\mathbb{R}^{2K}$.

Now that we have a choice of variational distribution with parameters $\underline{\theta}$ defined on unconstrained support, we can return to the problem of finding the optimal value of $\underline{\theta}$ for expression 5.9 using gradient ascent. We re-write the expression as follows:

$$ELBO(q) = \mathop{\mathbb{E}}_{q(\underline{\delta};\underline{\theta})}[\log p(T^{-1}(\underline{\delta}), \mathcal{D}) + \log |J_{T^{-1}}(\underline{\delta})|] - \mathop{\mathbb{E}}_{q(\underline{\delta};\underline{\theta})}[\log q(\underline{\delta};\underline{\theta})]. \tag{5.10}$$

Since the calculation of the $ELBO$ includes an intractable integral over $q(\underline{\delta};\underline{\theta})$, we cannot use automatic differentiation directly. Instead, we aim to calculate derivatives of the terms inside the expectation, where the terms are suitable for automatic differentiation [50]. Once we have calculated the derivatives inside the expectation, we are able to approximate the integral using Monte Carlo integration as discussed in Section 4.2.1.

In order to achieve this, we need to move the derivative operation inside the expectation. This can be achieved by "absorbing" the variational parameters $\underline{\theta}$. In the context of a Gaussian variational distribution, this means transforming the variational parameters such that $q(\underline{\delta};\underline{\theta})$ becomes a standard Gaussian.

To that end, we define the standardizing transformation for the mean-field Gaussian as

$$S_{\underline{\theta}}(\underline{\delta}) = diag(e^{\underline{\gamma}})^{-1}(\underline{\delta} - \underline{\mu}).$$

This transformation is known as elliptical standardization. For notational simplicity, define the transformed variational parameters $S_{\underline{\theta}}(\underline{\delta}) = \underline{\nu}$. Our variational distribution can now be expressed as

$$\begin{aligned} q(\underline{\nu}) &= \mathcal{N}(\underline{\nu}|\underline{0}, I) \\ &= \prod_{k=1}^{K} \mathcal{N}(\nu_k|0, 1), \end{aligned}$$

and allows the expectation in the $ELBO$ to absorb the variational parameters by defining it with respect to $\underline{\nu}$:

$$\mathcal{L} = ELBO(q) = \mathop{\mathbb{E}}_{\mathcal{N}(\underline{\nu}|\underline{0},I)}[\log p(T^{-1}(S_{\underline{\theta}}^{-1}(\underline{\nu})), \mathcal{D}) + \log |J_{T^{-1}}(S_{\underline{\theta}}^{-1}(\underline{\nu}))|] - \mathop{\mathbb{E}}_{q(\underline{\delta};\underline{\theta})}[\log q(\underline{\delta};\underline{\theta})]. \tag{5.11}$$

Note that the entropy term in expression 5.11, $\mathbb{E}_{q(\underline{\delta};\underline{\theta})}[\log q(\underline{\delta};\underline{\theta})]$, does not change with the later transformations. This is because an analytic form is available for the entropy of a Gaussian distribution, which makes it easy for us to calculate its value and gradient. Using the definition of the $ELBO$ in Eq. 5.11, we are now able to move the derivative operator inside the expectation (since the expectation no longer depends on $\underline{\theta}$). Taking the derivatives with respect to $\underline{\mu}$ and $\underline{\gamma}$ and applying the chain rule yields the

following formulae:

$$\frac{\partial \mathcal{L}}{\partial \underline{\mu}} = \mathop{\mathbb{E}}_{\mathcal{N}(\underline{\nu}|\underline{0},I)} \left[\frac{\partial}{\partial \underline{w}} \log p(\underline{w}, \mathcal{D}) \frac{\partial}{\partial \underline{\delta}} T^{-1}(\underline{\delta}) + \frac{\partial}{\partial \underline{\delta}} \log |J_{T^{-1}}(\underline{\delta})| \right] \tag{5.12}$$

$$\frac{\partial \mathcal{L}}{\partial \underline{\gamma}} = \mathop{\mathbb{E}}_{\mathcal{N}(\underline{\nu}|\underline{0},I)} \left[\Big(\frac{\partial}{\partial \underline{w}} \log p(\underline{w}, \mathcal{D}) \frac{\partial}{\partial \underline{\delta}} T^{-1}(\underline{\delta}) + \frac{\partial}{\partial \underline{\delta}} \log |J_{T^{-1}}(\underline{\delta})| \Big) \underline{\nu}^{\intercal} diag(e^{\underline{\gamma}}) \right] + 1 \tag{5.13}$$

Each of the above derivatives are obtainable through automatic differentiation and the intractable integral from the expectation is easy to approximate through Monte Carlo integration. Before we formally define the ADVI algorithm, we need a step-size to update the variational parameters for each iteration. The step-size used by Kucukelbir et al. is called an adaptive step-size $\underline{\rho}^{(i)}$ for iteration i, where each element k in $\underline{\rho}^{(i)}$ is defined as:

$$\rho_k^{(i)} = \eta \times i^{-\frac{1}{2}+\epsilon} \times \left(\tau + \sqrt{s_k^{(i)}} \right)^{-1}, \text{ where} \tag{5.14}$$

$$s_k^{(i)} = \alpha g_k^{2(i)} + (1-\alpha) s_k^{(i-1)}.$$

The value of η as used above is chosen through a grid-search over the interval $\{0.01, 0.1, 1, 10, 100\}$ [51]. The scalar α is chosen in the interval $(0, 1)$ and determines the importance of information from the previous iterations. The value of ϵ is selected as $\epsilon = 10^{-16}$ and τ chosen as 1 by Kucukelbir et al. The vector $\underline{g}^{(i)} = \{g_1^{2(i)}, ..., g_k^{2(i)}\}$ is defined as a collection of the gradients calculated at iteration i.

We can now define an algorithm that iteratively optimizes the *ELBO*:

Algorithm 6 Automatic Differentiation Variational Inference (ADVI)

- Initialise all variational parameters in $\underline{\theta}^{(0)}$ as $\underline{0}$
- Initialise iteration counter $i = 0$
- Repeat until change in ELBO becomes smaller than user-defined threshold:
  - Increase i by 1
  - Draw a sample of size s for $\underline{\nu}^s \sim \mathcal{N}(\underline{0}, I)$
  - Calculate $\frac{\partial \mathcal{L}}{\partial \underline{\mu}}$ and $\frac{\partial \mathcal{L}}{\partial \underline{\gamma}}$ using automatic differentiation, Monte Carlo integration and the sample $\underline{\nu}^s$.
  - Calculate the step size $\underline{\rho}^i$
  - Update $\underline{\mu}^{(i+1)} = \mu^{(i)} + diag(\underline{\rho}^{(i)}) \frac{\partial \mathcal{L}}{\partial \underline{\mu}}$
  - Update $\underline{\gamma}^{(i+1)} = \gamma^{(i)} + diag(\underline{\rho}^{(i)}) \frac{\partial \mathcal{L}}{\partial \underline{\gamma}}$
- Return updated $\underline{\mu}^{(i)} = \underline{\mu}^*$ and $\underline{\gamma}^{(i)} = \underline{\gamma}^*$
- Let $\underline{\theta}^* = \{\underline{\mu}^*, \underline{\gamma}^*\}$
- We end up with $\underline{\theta}^* = \underset{\underline{\theta}}{argmax}\, ELBO(q(\underline{w}; \underline{\theta}))$

Note that the $ELBO$ used in Algorithm 6 implicitly maps all transformed parameters back to its original support that we started with, as defined in expression 5.11. Therefore, the output from the algorithm above can be used as an approximation to the posterior distribution of the unknown weight parameters, $p(\underline{w}|\mathcal{D})$. We can also introduce scalability to large sets of data for ADVI by using stochastic optimization. This is achieved by sampling a smaller subset of size $M < N$ of the data with each iteration and scaling the likelihood of the model by $\frac{N}{M}$ when calculating the gradients [52]. This scalability to large sets of data proves to be an advantage over MCMC methods in cases where we have highly complex models which require massive sets of data for training.

5.2.6 Example

To illustrate VI, we use the same set of data from Section 5.1.2 and compare the results to that obtained by MCMC. To recap, we are interested in approximating the posterior distribution over the weight parameters $\underline{w} = \{w_0, w_1, w_2\}$ with the form

$$P(\underline{w}|X,\underline{y}) \propto \prod_{i=1}^{N}\left[\frac{1}{1+e^{-\underline{w}^{\intercal}\underline{x_i}}}\right]^{y_i}\left[\frac{e^{-\underline{w}^{\intercal}\underline{x_i}}}{1+e^{-\underline{w}^{\intercal}\underline{x_i}}}\right]^{1-y_i} \times \prod_{j=1}^{3}\frac{1}{\sqrt{2\pi\sigma_j}}exp\Big\{-\frac{1}{2}\Big(\frac{w_j}{\sigma_j}\Big)^2\Big\} \tag{5.15}$$

for Bayesian logistic regression. As proven in Section 5.2.2, we don't need an exact expression of the evidence

$$\int ... \int P(\underline{y}|X,\underline{w})P(\underline{w})d\underline{w}$$

to approximate the posterior using VI. Instead, we maximize the lower bound of the evidence to achieve the same result. To that end, we use expression 5.15 as the target density and let ADVI automatically transform the weight parameters $\underline{w}$ so that we may use a mean-field Gaussian as the variational distribution.

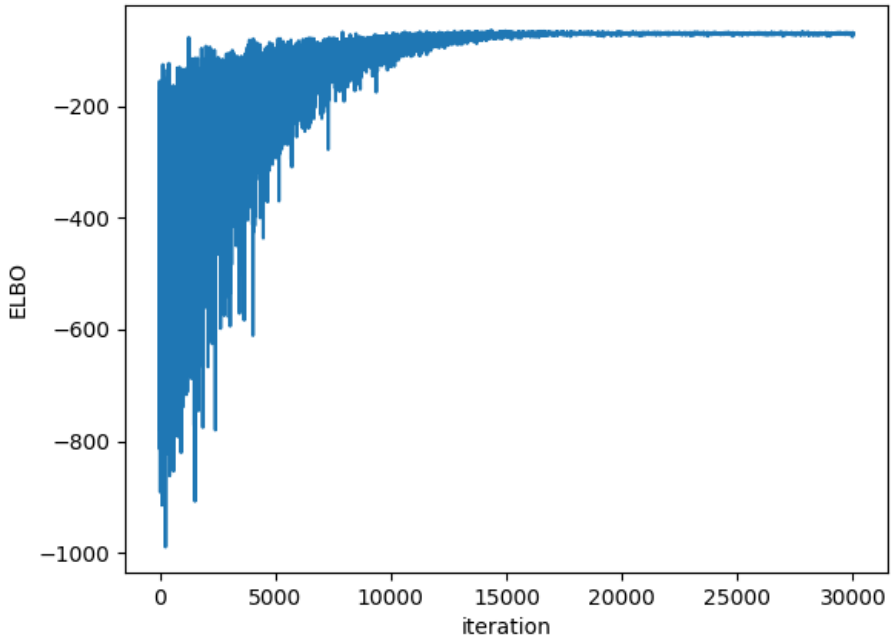


Figure 5.5: ADVI algorithm convergence.

Instead of sampling a predetermined number of samples from a Markov Chain as done in MCMC, we only need to define a threshold for the value of the ELBO which indicates that we have found a local minimum for the KL-divergence. This is fundamentally how VI differs from MCMC since we are optimizing an objective function instead of converging to a stationary distribution. Figure 5.5 illustrates how the ADVI algorithm converges shortly after reaching 15000 iterations, where the fluctuations shown in the value of the ELBO is due to sampling variation in the algorithm as it approaches the optimum. Having obtained the local maximum of the ELBO for a certain setting of the variational parameters $\underline{\theta}$, our posterior distribution

can be approximated through a variational distribution that uses the above mentioned parameters, $q(\underline{w};\underline{\theta})$.

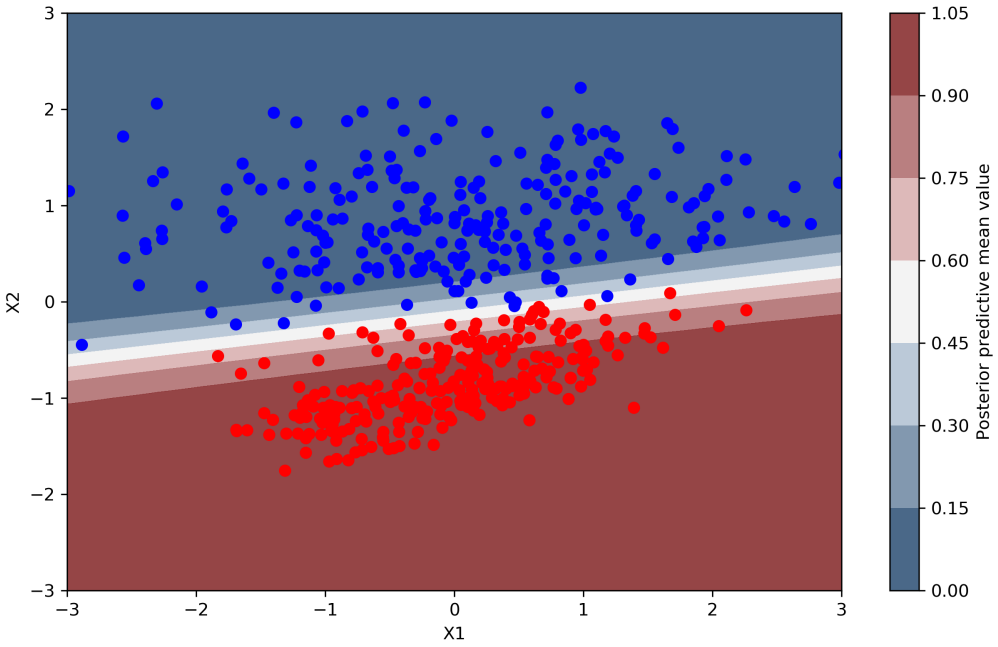


Figure 5.6: Class label prediction and posterior predictive distribution.

The predicted class labels and posterior predictive distribution is shown in Figure 5.6. Similar to the results obtained from MCMC in Section 5.1.2, the data points closest to the linear decision boundary show the highest uncertainty as seen in Figure 5.7.

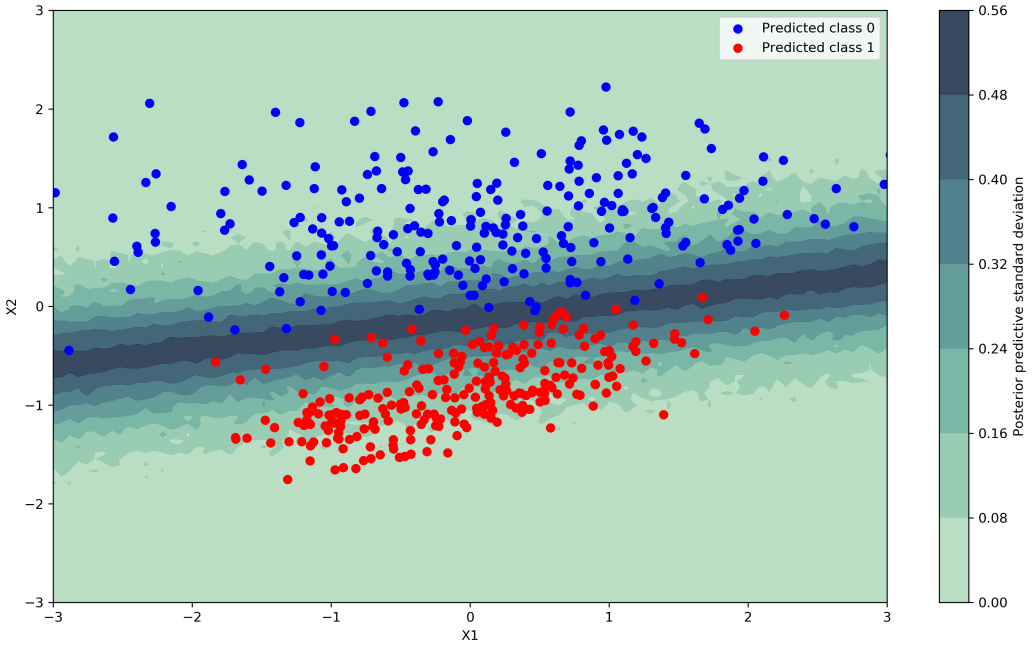


Figure 5.7: Posterior predictive standard deviation as uncertainty measure.

Given the simplicity of the model, the results obtained from VI and MCMC do not differ much other than calculation time, where VI performed slightly faster. In the next section we derive both frequentist and Bayesian approaches to Neural Networks and apply these models to more complex problems.

Chapter 6

Neural Networks

A Neural Network (NN) is a supervised learning algorithm which has recently become a popular solution to complex problems in areas such as Natural Language Processing (NLP) [12], healthcare [19] and image processing [3]. As suggested by its name, an NN is inspired by the way in which the human brain processes information through a large number of connected neurons, providing the ability to identify complex relationships in data.

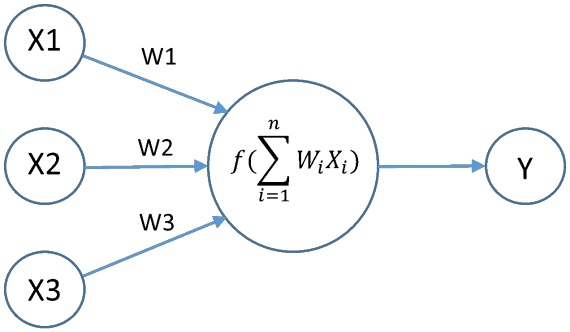


Figure 6.1: Structure of a simple Neural Network

The structure of the model consists of an input layer, output layer and a pre-defined number of hidden layers each containing a certain number of neurons (also called nodes). Each node consists of a weighted sum of values passed through an activation function on the previous layer and the weights are assumed to be unknown parameter values which are estimated during the learning phase. A simple NN diagram is shown in Figure 6.1 where the model's architecture consists of an input layer containing 3 nodes, a single hidden layer containing a single node and an output node. If we chose the activation function of this NN to be a sigmoid function $f(x) = \frac{1}{1+e^{-x}}$, this particular design would be identical to a logistic regression.

For a unit step function (as described in Section 2.1.1), the NN would become a perceptron as introduced in Chapter ??. This makes the NN a flexible model, allowing the user to design an architecture with a complexity that suits the problem and available resources.

In this chapter we start by deriving a learning method of a classical NN called backpropagation. We then introduce stochastic gradient descent, a learning algorithm which aids in scalability to larger sets of data and model complexity. To capture uncertainty in model predictions, we derive a probabilistic approach to NNs called Bayesian Neural Networks (BNNs) and show how we estimate the posterior distribution with techniques derived in Chapter 5.

6.1 Backpropagation in Neural Networks

The learning process of a classical NN discussed in this dissertation is called Backpropagation and constitutes two stages, viz. the Forward Pass and Backward Pass. With each iteration, the gradient descent algorithm calculates the gradients of the appointed cost function with respect to the weight parameters embedded in each layer (using backpropagation) and incrementally descends the cost function in the opposite direction of the gradients.

There are various alternative optimization techniques used with gradient descent such as Adam [53] which uses adaptive estimates of lower-order moments of the cost function, or Adagrad [54] which adapts the learning rate of the learning algorithm to the values of the model parameters. In addition to the choice of cost function, an activation function (in most cases a non-linear transformation) for each layer should also be decided on beforehand. More information regarding the activation function will be discussed in conjunction with the forward pass.

6.1.1 Forward Pass

Each iteration of the training process starts with a forward pass, where data from the input layer passes through each hidden layer and activation function and produces an output with a corresponding cost, as determined by the cost function. For ease of illustration, we mathematically derive the backpropagation algorithm for an NN with one hidden layer containing 3 components as shown in Figure 6.2, where we wish to output a single value between 0 and 1 for each row in our data matrix. For deeper NN architectures, the algebra remains similar in structure and can extend to an infinite number of hidden layers in theory. Similar to logistic regression, this model will be used for binary classification. However, since we have introduced additional nodes in our hidden layer, the model is capable of learning more interesting relationships in the data.

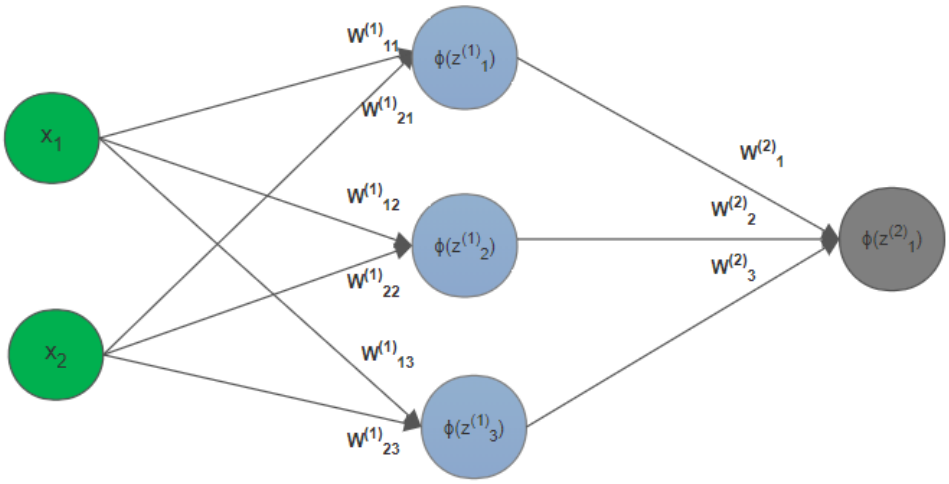


Figure 6.2: Simple Neural Network design

Let X be an $N \times 2$ data matrix with corresponding binary response variable $\underline{y}$ such that each element of $\underline{y}$ belongs to the set $\{0, 1\}$. At each layer an extra component equal to a constant called a bias (usually chosen as 1) is added. There are two weight matrices used in this model, $W^{(1)}$ and $W^{(2)}$ with dimensions 2×3 and 3×1 respectively. Intuitively, the dimension of each weight matrix corresponds to the number of nodes contained in the layers on either side of it. The initial values for the weights are randomized (preferably with a standard Gaussian) and each bias component is initialised as the value 1.

Initially, each row in input layer X is multiplied by the first weight matrix to produce the first hidden layer's components:

$$Z^{(1)} = XW^{(1)} + b^{(1)}, \text{ with dimension } 1 \times 3.$$

The hidden layer components are then passed through the activation function $\phi(Z^{(1)})$ and multiplied by the second weight matrix which results in

$$Z^{(2)} = \phi(Z^{(1)})W^{(2)} + b^{(2)}, \text{ a scalar value.}$$

As previously stated, we desire an output value between 0 and 1 for binary classification. To that end, our choice of activation function is the sigmoid function:

$$\phi(z_i^{\ell}) = \frac{1}{1 + e^{-z_i^{(\ell)}}}.$$

It should be noted that passing a matrix through this activation function means performing the transformation element-wise on the matrix. Finally, for an input value $\underline{x}_i$, the output value $\hat{y}_i$ is obtained by

passing $Z^{(2)}$ through the last activation function ϕ:

$$\hat{y} = \phi(Z^{(2)}).$$

6.1.2 Backward Pass

The goal during the learning phase is to minimize some cost function decided upon beforehand. We would now like to calculate the value of the cost function from the output $\hat{y}$ and attempt to minimize it by changing the values of the weights in each of the layers. This is achieved through backpropagation.

Before we perform backpropagation, we need a cost function and optimization technique. Continuing the example from the previous section, we use a squared error cost function and performm gradient descent to find optimal parameter values.

Gradient-descent is a simple method which allows us to incrementally lower the value of the cost function by changing the weight matrices of each layer. The changes in the weight matrices are determined by taking the partial derivative of the cost function with respect to each weight matrix. Backpropagation allows us to use the hierarchical structure of the NN to calculate each weight parameter's derivative.

We define the cost function as

$$S = \frac{1}{2}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2,$$

where

$$\hat{y} = \phi(Z^{(2)}) = \phi(\phi(Z^{(1)})W^{(2)} + b^{(2)}). \tag{6.1}$$

Starting with the weight matrix closest to the output, $W^{(2)}$, we want to determine the partial derivative

$$\frac{\partial S}{\partial W^{(2)}} = \frac{\partial}{\partial W^{(2)}}\frac{1}{2}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 = -\sum_{i=1}^{N}(y_i - \hat{y}_i)\frac{\partial \hat{y}}{\partial W^{(2)}}. \tag{6.2}$$

We focus on the term

$$\frac{\partial \hat{y}}{\partial W^{(2)}} = \frac{\partial \phi(Z^{(2)})}{\partial W^{(2)}}, \tag{6.3}$$

and by using the chain rule we can express Eq. 6.3 as

$$\frac{\partial \phi(Z^{(2)})}{\partial W^{(2)}} = \frac{\partial \phi(Z^{(2)})}{\partial Z^{(2)}} \frac{\partial Z^{(2)}}{\partial W^{(2)}}. \tag{6.4}$$

As defined above, $Z^{(2)} = \phi(Z^{(1)})W^{(2)} + b^{(2)}$ and therefore

$$\frac{\partial Z^{(2)}}{\partial W^{(2)}} = \frac{\partial}{\partial W^{(2)}}(\phi(Z^{(1)})W^{(2)} + b^{(2)}) = \phi(Z^{(1)}). \tag{6.5}$$

Since we have decided that our activation function is the sigmoid function

$$\phi(z) = \frac{1}{1 - e^{-z}},$$

the derivative of the sigmoid $\phi(Z^{(2)})$ with respect to $Z^{(2)}$ is expressed as

$$\frac{\partial \phi(Z^{(2)})}{\partial Z^{(2)}} = \frac{e^{-Z^{(2)}}}{(1 + e^{-Z^{(2)}})^2}. \tag{6.6}$$

By using Eq 6.5 and 6.6, we can now express Eq. 6.2 as a product of matrices:

$$\frac{\partial S}{\partial W^{(2)}} = -\sum_{i=1}^{N}(y_i - \hat{y}_i)\frac{e^{-Z^{(2)}}}{(1 + e^{-Z^{(2)}})^2}\phi(Z^{(1)}) = -\phi^{\intercal}(Z^{(1)})\delta^{(2)}, \tag{6.7}$$

where $\phi^{\intercal}(Z^{(1)})$ is the transpose of the $N \times 3$ activation matrix on the first layer's components, and

$$\delta^{(2)} = (\underline{y} - \underline{\hat{y}})\frac{e^{-Z^{(2)}}}{(1 + e^{-Z^{(2)}})^2}$$

is an element-wise multiplication between the vectors $(\underline{y} - \underline{\hat{y}})$ and $\frac{e^{-Z^{(2)}}}{(1+e^{-Z^{(2)}})^2}$.

We now determine the partial derivative of S with respect to the first weight-matrix, i.e.

$$\frac{\partial S}{\partial W^{(1)}} = -\sum_{i=1}^{N}(y_i - \hat{y}_i)\frac{\partial \hat{y}}{\partial W^{(1)}} = -\sum_{i=1}^{N}(y_i - \hat{y}_i)\frac{\partial \phi(Z^{(2)})}{\partial Z^{(2)}}\frac{\partial Z^{(2)}}{\partial W^{(1)}}. \tag{6.8}$$

Since we already have the result of $\frac{\partial \phi(Z^{(2)})}{\partial Z^{(2)}}$ shown in eq 6.6, we only need to focus on $\frac{\partial Z^{(2)}}{\partial W^{(1)}}$.

To show the dependence of $Z^{(2)}$ on $W^{(1)}$, we expand eq 6.1 as follows:

$$\hat{y}_i = \phi(Z^{(2)}) = \phi(\phi(Z^{(1)})W^{(2)} + b^{(2)}) = \phi(\phi(XW^{(1)} + b^{(1)})W^{(2)} + b^{(2)}). \tag{6.9}$$

We can now express $\frac{\partial Z^{(2)}}{\partial W^{(1)}}$ as

$$\frac{\partial Z^{(2)}}{\partial W^{(1)}} = \frac{\partial Z^{(2)}}{\partial \phi(Z^{(1)})} \frac{\partial \phi(Z^{(1)})}{\partial W^{(1)}}, \tag{6.10}$$

and by the definition of $Z^{(2)}$, we have that $\frac{\partial Z^{(2)}}{\partial \phi(Z^{(1)})} = W^{(2)}$.

Furthermore, by using a similar result as seen in eq 6.6 we can express $\frac{\partial \phi(Z^{(1)})}{\partial W^{(1)}}$ as

$$\frac{\partial \phi(Z^{(1)})}{\partial W^{(1)}} = \frac{\partial \phi(Z^{(1)})}{\partial Z^{(1)}} \frac{\partial Z^{(1)}}{\partial W^{(1)}} = \frac{e^{-Z^{(1)}}}{(1+e^{-Z^{(1)}})^2} X. \tag{6.11}$$

By using Eq. 6.11, 6.10 and 6.7 we can express Eq. 6.8 with matrix notation as

$$\frac{\partial S}{\partial W^{(1)}} = -X^\intercal \left[\delta^{(2)} W^{\intercal(2)} \frac{e^{-Z^{(1)}}}{(1+e^{-Z^{(1)}})^2} \right], \tag{6.12}$$

$$= -X^\intercal \delta^{(1)} \tag{6.13}$$

where $\delta^{(1)} = \delta^{(2)} W^{\intercal(2)} \frac{e^{-Z^{(1)}}}{(1+e^{-Z^{(1)}})^2}$ and the terms $\delta^{(2)} W^{\intercal(2)}$ and $\frac{e^{-Z^{(1)}}}{(1+e^{-Z^{(1)}})^2}$ are multiplied element-wise.

We can now update the weight and bias matrices by using Eq. 6.7 and 6.13 and a learning rate $\delta \in [0, 1]$:

$$W^{(1)}_{new} = W^{(1)}_{old} - \lambda \frac{\partial S}{\partial W^{(1)}}$$

$$= W^{(1)}_{old} + \lambda X^\intercal \delta^{(1)} \tag{6.14}$$

$$W^{(2)}_{new} = W^{(2)}_{old} - \lambda \frac{\partial S}{\partial W^{(2)}}$$

$$= W^{(2)}_{old} + \lambda \phi^\intercal(Z^{(1)}) \delta^{(2)} \tag{6.15}$$

$$b^{(1)}_{new} = b^{(1)}_{old} + \lambda \delta^{(1)} \tag{6.16}$$

$$b^{(2)}_{new} = b^{(2)}_{old} + \lambda \delta^{(2)} \tag{6.17}$$

Similarly, for a Neural Network with any number of hidden layers, the weight and bias updates can be expressed as

$$W^{(\ell)}_{new} = W^{(\ell)}_{old} - \lambda \frac{\partial S}{\partial W^{(\ell)}}$$

$$= W^{(\ell)}_{old} + \lambda \phi^\intercal(Z^{(\ell-1)}) \delta^{(\ell)} \tag{6.18}$$

$$b^{(\ell)}_{new} = b^{(\ell)}_{old} + \lambda \delta^{(\ell)} \tag{6.19}$$

where the calculation of $\delta^{(\ell)}$ depends on the cost and activation functions decided upon beforehand.

By using the derivations in expressions 6.18 and 6.19, we can now formally define the backpropagation algorithm.

Algorithm 7 Backpropagation algorithm

- Generate random initial values for weight parameters in all layers
- Set initial values of bias parameters to 1
- Repeat while change in cost function exceeds a user defined threshold:
  - Using input matrix X, perform a forward pass and calculate the cost
  - Update the weights and biases with gradients derived in expressions 6.18 and 6.19
- For each layer ℓ, return weight matrix $W_{new}^{(\ell)}$ and $b_{new}^{(\ell)}$

6.1.3 Stochastic Gradient Descent

Gradient descent was a powerful and widely used tool when training earlier versions of the multilayer perceptron [55]. However, since the entire set of data is required to compute the gradients in each iteration, such a traditional method has started to suffer in modern applications. This is due to the lack of computational resources for complex models (such as deep NNs) sometimes requiring massive sets of data, or in on-line learning environments where observations are acquired sequentially instead of a single batch. In the latter case, computing the gradient of the cost function would be impossible without the entire set of data available.

Stochastic Gradient Descent (SGD) is a method used to allow gradient descent algorithms to scale to large sets of data without overwhelming the memory of a computer. Instead of using the entire set of data to calculate the gradient of the cost function for each iteration, we compute the gradient of the cost function for only one randomly selected observation (or a small random sample) at a time. It can be shown that the gradient calculated on a single observation is an unbiased estimator of the gradient over the mean squared error cost function using the full dataset.

$$\begin{aligned}
\mathop{\mathbb{E}}_{\underline{x}_m}\left[\frac{\partial S_m}{\partial W}\right] &= \frac{\partial}{\partial W}\mathop{\mathbb{E}}_{\underline{x}_m}[S_m] \\
&= \frac{\partial}{\partial W}\sum_{i=1}^{N} P(\underline{X}_m = \underline{x}_i)S_i \\
&= \frac{1}{N}\sum_{i=1}^{N} S_i \\
&= \frac{1}{N}S
\end{aligned}$$

Algorithm 7 would therefore be changed as follows:

Algorithm 8 SGD Backpropagation algorithm

- Generate random initial values for weight parameters in all layers
- Set initial values of bias parameters to 1
- Repeat while change in cost function exceeds a user defined threshold:
  - Select a uniformly random sample $X^{(m)} \subseteq X$ of size m, with replacement.
  - Using the sampled $X^{(m)}$, perform a forward pass and calculate the cost
  - Update the weights and biases with gradients derived in expressions 6.18 and 6.19
- For each layer ℓ, return weight matrix $W^{(\ell)}_{new}$ and $b^{(\ell)}_{new}$

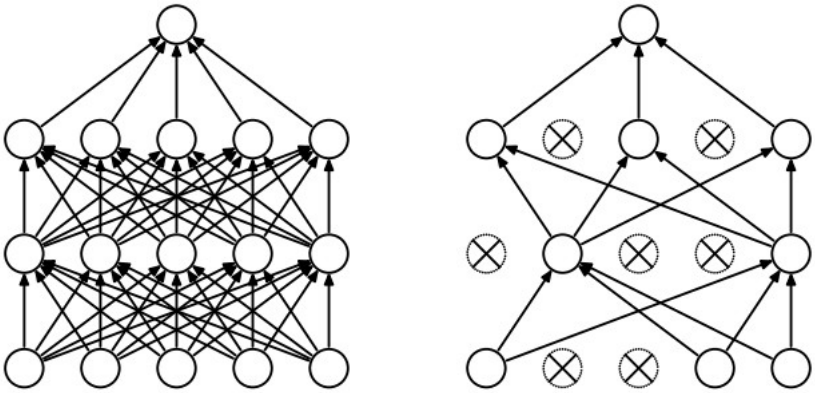
Typically we would choose a smaller step size for the update in each iteration since the random sampling within the algorithm introduces higher variance in our estimate of the weight parameters. Compared to traditional gradient descent, SGD would follow a more noisy path toward the optimum. Despite this additional variance, gradients obtained by SGD are estimators of the full gradient and would converge to the same optimum given the appropriate step size.

6.1.4 Dropout

Deep neural networks often contain thousands of weight parameters, which in turn may require large sets of data to train on. These deep architectures are highly capable of learning complex patterns in a set of data, however, some obstacles persist when using these models in practice. Aside from the computational challenges caused by model complexity, another common challenge when training NNs is the possibility of learning patterns caused by variation inherent to only the specific training set, i.e. over-fitting the data. Therefore, ensuring that an NN generalises well to unseen data is an important step and is achieved

through parameter regularisation methods. A classic regularisation approach used in NN training is a technique called dropout [24].

Intuitively, dropout is applied to a training algorithm by "ignoring" a random selection of hidden and visible (input) nodes with each training observation or epoch (a single pass through the entire training set). This causes the nodes within the model to be less co-dependent and in turn learn a simpler pattern over the training data, at the cost of training error. Even though we are interested in minimizing the cost function over the training set, by sacrificing predictive performance on the training set through regularisation we improve the chance of providing a better fit over future, unseen data.



(a) Standard Neural Net (b) After applying dropout.

Figure 6.3: Illustration of dropout in a neural network

For each layer ℓ, the corresponding weight matrix $W^{(\ell)}$ is multiplied by a random vector $\underline{\beta}^{(\ell)}$ where each element of $\underline{\beta}^{(\ell)}$ is randomly generated from a Bernoulli distribution with parameter p (called the keep-probability). Therefore, when we perform a forward pass for training example $\underline{x_i}$, every node in a hidden layer multiplied by the random vector has a probability p of remaining in the NN. When applying the NN to a test set, we use the entire NN without dropping any nodes. To ensure that the expected outcome of the full NN on the test set remains the same as with the training set where dropout was used, we use a technique called Inverted Dropout where the resulting activation function from each node that has not been dropped is scaled by $\frac{1}{1-p}$ during training. Dropout NNs can be trained using the backpropagation algorithm and optimized using gradient descent or SGD. Following a forward pass with dropout, only the weight gradients for nodes that weren't dropped are updated. Implementation of the above steps in the backpropagation algorithm (Algorithm 8) will regularize the weights of the NN, providing additional robustness to the over-fitting problem inherent to NNs.

6.1.5 Example

The NN's ability to fit a model over non-linear data will be illustrated in this section. We design the NN to consist of two hidden layers, each containing 5 nodes. A ReLU activation function is used on each hidden layer. The ReLU function is widely used and has been shown to improve convergence rates on SGD algorithms due to its computational simplicity and non-saturation of its gradient [3]. We define the ReLU activation function as $\phi(z) = max(0, z)$.

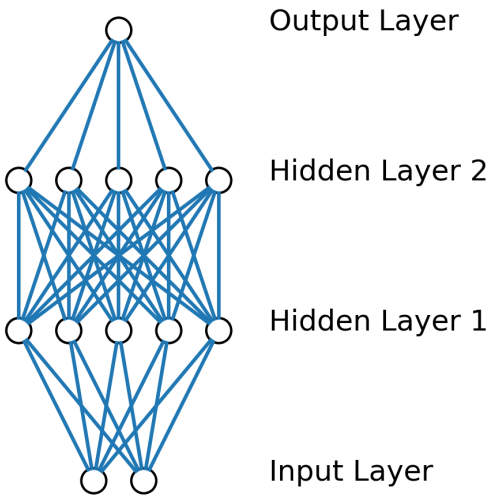


Figure 6.4: Neural network with two hidden layers, each containing 5 nodes.

The generated data $\mathcal{D} = \{X, \underline{y}\}$ contains pairs of observations where each observation $\underline{x}_i \in X$ is defined as $\underline{x}_i = \{x_{i1}, x_{i2}\}$ for $i = 1, ..., n$ and every $y_i \in \underline{y}$ is a binary response variable indicating the class of every observation. Since the observations contained in X are 2-dimensional, our input layer contains 2 nodes. The output layer contains only 1 node with a sigmoid activation function $\sigma(\sum_{i=1}^{5} \phi(z_i^{(2)}))$ which produces a value in the range $[0, 1]$ (for notational clarity, we denote each node in hidden layer 2 as $z_i^{(2)}$ for $i = 1, ..., 5$). Classification is done by passing the value produced by the sigmoid activation function σ through a final step-function defined as:

$$\hat{y} = f(\sigma),$$
$$\text{where } f = \begin{cases} 1 & \text{if } \sigma > \alpha \\ 0 & \text{otherwise} \end{cases},$$

where α is a pre-defined threshold, usually chosen as 0.5.

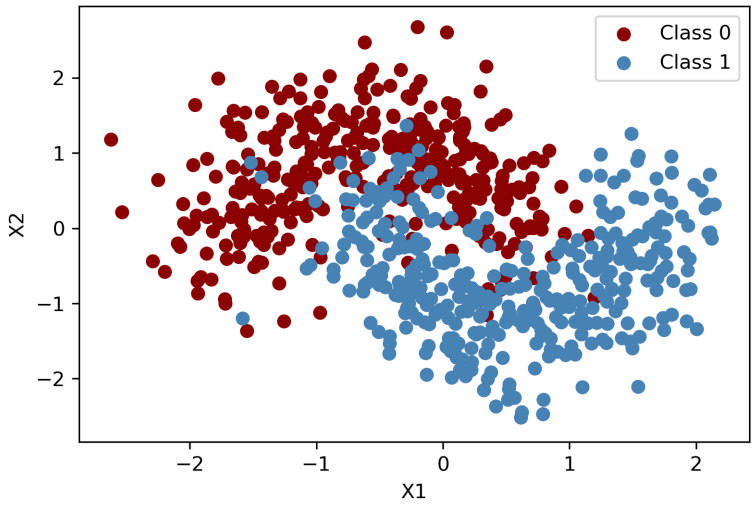


Figure 6.5: Training data

From Figure 6.5 we see that a linear classification model such as logistic regression would not perform well on the available data, since the classes represented by the colours are not linearly separable. We train the NN on the above-mentioned data using backpropagation and optimized with SGD. As the algorithm iterates, the predictive accuracy of the model is evaluated on a training and a test set. The comparison between the training and test accuracy as the model learns is shown in Figure 6.6.

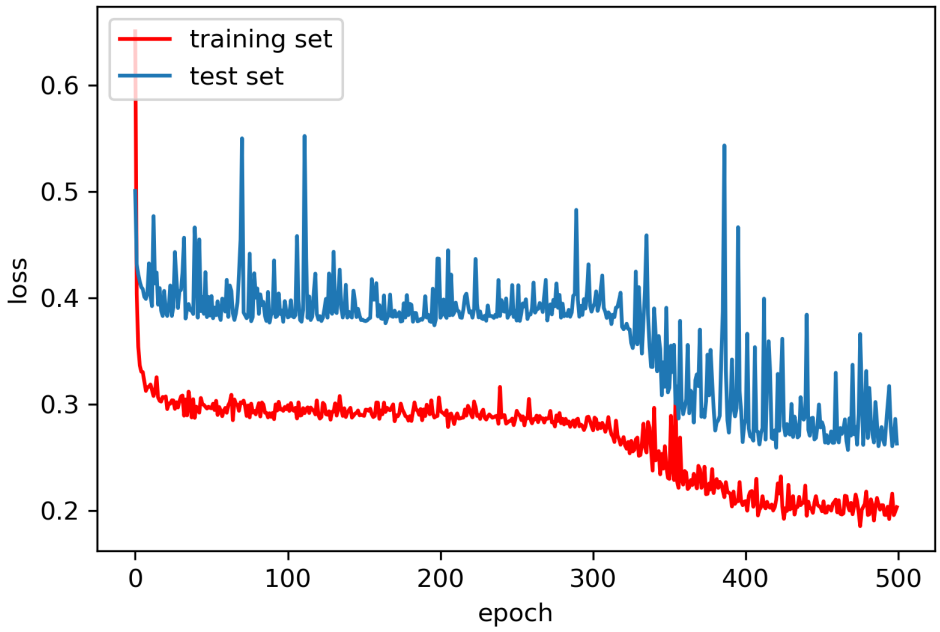
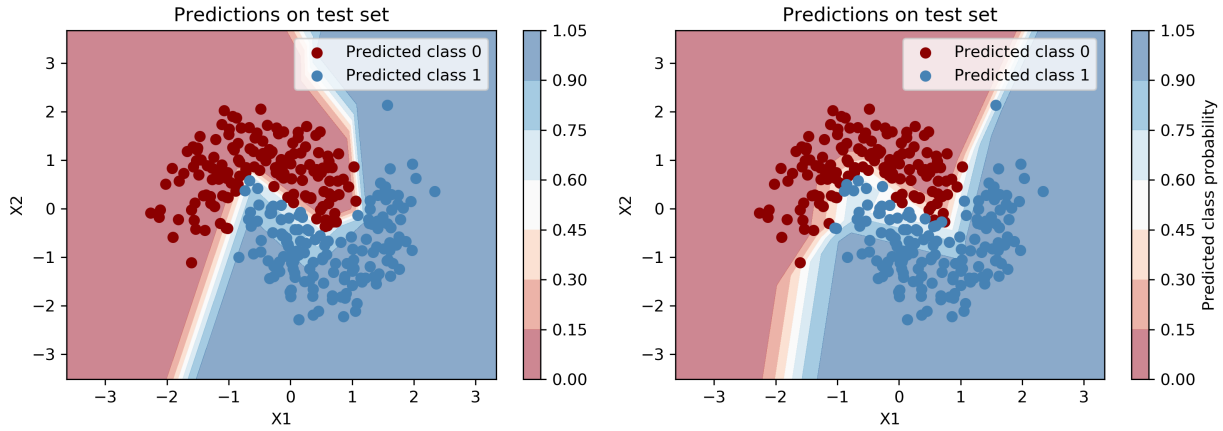


Figure 6.6: Training and test loss during backpropagation with SGD

The difference between training and test loss shown in Figure 6.6 hints at the possibility that our model has been over-fitted to the training data. To prevent this, we train an additional NN by using the dropout regularisation method. Figure 6.7 illustrates how using dropout affects the shape and width of the decision curve learned from the data. Although both models achieved similar prediction accuracy on a simulated evaluation set, the decision curve produced by the dropout NN in Figure 6.15a shows a wider curve and a more general shape to account for variation in the training set, which is preferable when predicting classes on future, unseen data.



(a) Without dropout

(b) With dropout

Figure 6.7: Model predictions and decision curve produced by NN.

The difference between the training and test loss while training the dropout NN is shown in Figure 6.8. It is clear that the model is less prone to over-fitting to the training data since the difference in loss is smaller than the case shown in Figure 6.6.

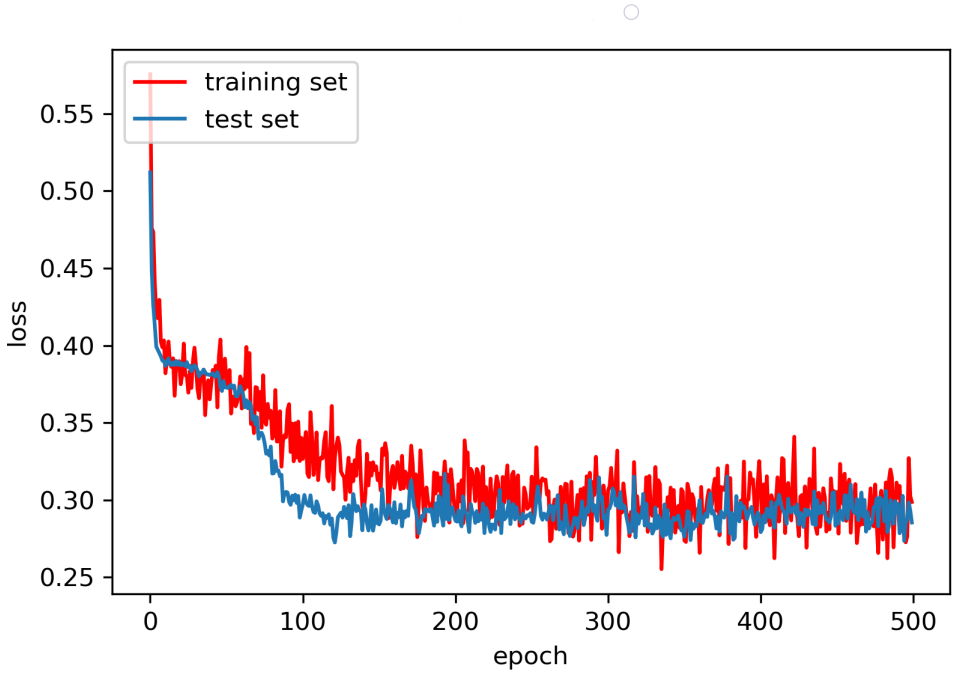


Figure 6.8: Training and test loss during backpropagation with SGD using dropout regularisation

It would be tempting to interpret the width of the decision boundary in Figure 6.7 as a measure of

uncertainty of class predictions that lie close to it. However, the class prediction probability for each observation remains a single point estimate from the NN. In fact, we have no measure of how certain we are about the probabilities assigned to observations that lie close to the decision boundary before applying the step-function and assigning a predicted class.

In Section 6.3.1 we introduce a probabilistic approach to NNs in order to derive a posterior probability distribution over all weight parameters and model predictions. Certain characteristics from these posterior distributions would provide insight into how certain we are when predicting classes on unseen data.

6.2 Convolutional Neural Networks

A Convolutional Neural Network (CNN) is a class of NNs with a deep architecture designed for image processing tasks [56]. The superior performance shown by Krizhevsky et al. in the 2012 ImageNet competition that sparked widespread interest in deep learning was achieved with a CNN. Similar to traditional feed-forward NNs (as described earlier in this chapter), the CNN contains an input layer, output layer and associated differentiable cost function with which we update unknown weight parameters during training. What makes a CNN unique is the shape of the input data as well as the hidden layers, called convolutions. Instead of each new training/test observation being a vector of inputs, the CNN receives an image with three dimensions, i.e. the image's width and height (represented by the number of pixels) and the depth (represented by the colour value of each pixel such as RGB values). Since the input data for a CNN is assumed to be image data, the nodes between hidden layers are not always fully-connected as in traditional NNs. Instead, we apply filters (also called feature maps) on layers that shifts over different sections of the layer. In NN literature, this operation is known as a convolution. The design is inspired by the way the visual cortex creates a small receptive field in which we process visual information [57] [56]. The advantage to using this design is in its scalability to large images containing more pixels as well as its invariance to the specific location of objects in an image.

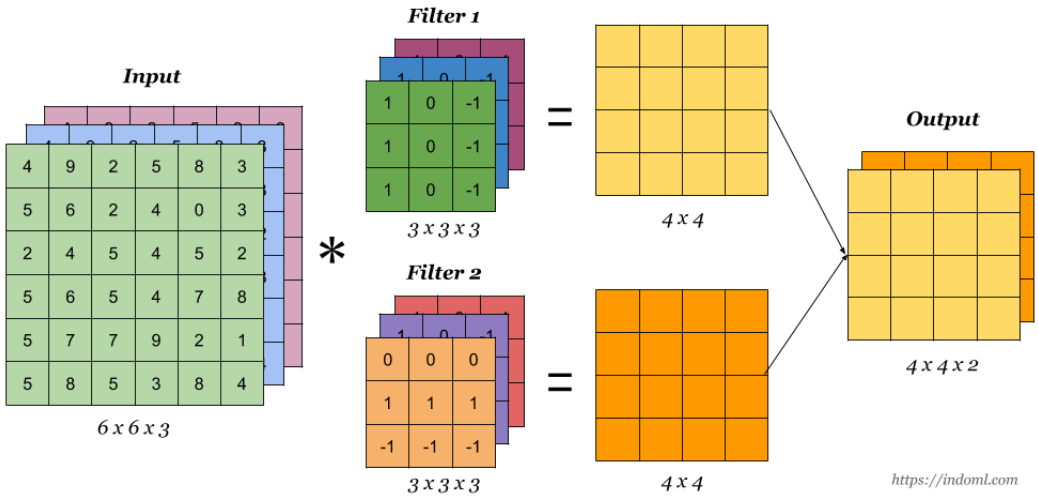


Figure 6.9: Illustration of CNN filters stacked in a layer
(Image source: www.indoml.com)

In a CNN, each layer has 3 dimensions as opposed to 2 layers in a traditional NN. For example, if an input color image (RBG) contained 32×32 pixels, the input layer would have dimension $32 \times 32 \times 3$ to account for the RGB (red, green blue) values represented by each pixel. In subsequent hidden layers, multiple filters could be applied over the previous layer. The resulting convolutional layers from each filter are stacked and create one three-dimensional layer on its own. The third dimension (called depth) of a layer indicates the number of filters applied to the previous layer. Intuitively, filters search for specific observed patterns or features in pixel values inside the image, hence the alternative name "feature maps". The elements in each filters act as network weights where each weight passes through a nonlinear activation function and is estimated during the learning phase.

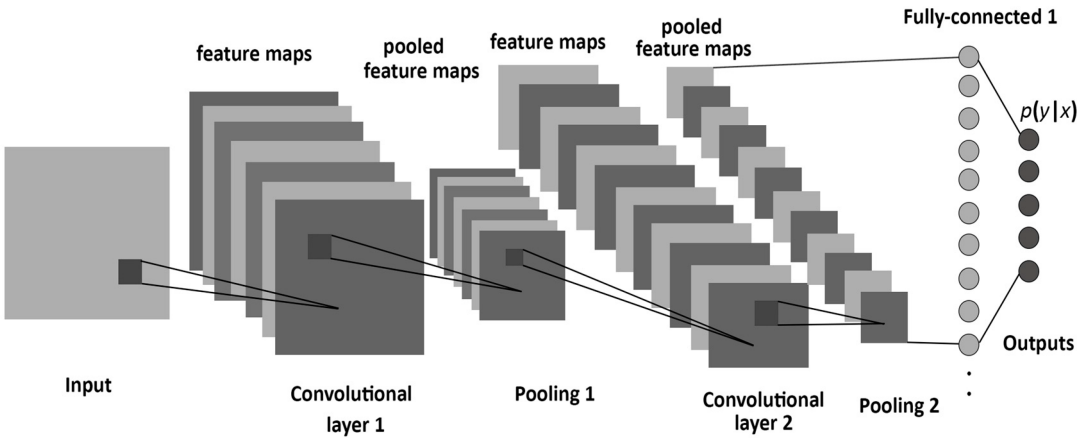


Figure 6.10: Illustration of CNN architecture

As a means to prevent over-fitting as well as reduce the complexity of the model, many CNN architectures include a tool called pooling layers. Though many pooling techniques exist, the most commonly used pooling layer is called max pooling. Max pooling is achieved by moving a window over a layer and selecting only the maximum value in the window with each move, where the stride of the pooling layer is the size of each jump when the window moves. Generally, the resulting layer has a smaller width×height dimension and depends on the stride and dimensions of the pooling layer.

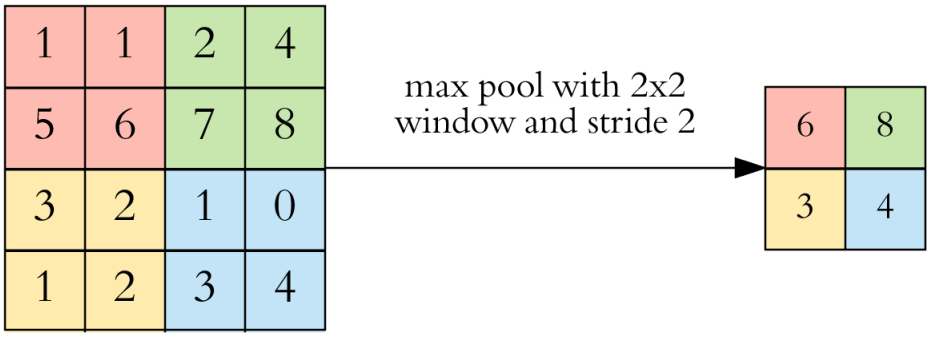


Figure 6.11: Illustration of max pooling technique

In a classification context, we need to output a final softmax layer indicating each observation's class probability prediction. To achieve this, we "flatten" the final convolutional layer by stacking each value from every 2-dimensional feature map into a 1-dimensional vector as shown in the final layer of Figure 6.10. We treat this vector as the input to a standard NN where the nodes in each layer are fully connected. Standard CNNs can be trained using the backpropagation algorithm and optimized with gradient-based methods such as SGD.

6.3 Probabilistic Framework

The objective of gaining additional insight to the output from an NN can be achieved through the use of a probabilistic framework. Instead of producing a single point estimate for a given observation, we are interested in inferring an entire probability distribution over possible model predictions for a given input. This is made possible through the use of Bayesian inference where we define a prior distribution over all the weight parameters and derive posterior distributions for all variables of interest defined in the model. In this section we describe the process of defining and training a Bayesian Neural Network (BNN).

6.3.1 Bayesian Neural Networks

Suppose we are interested in finding a posterior predictive distribution over the output of some NN with $\mathcal{L}$ hidden layers. At the ℓ^{th} layer, we define the matrix of weight parameters as W^ℓ and its corresponding bias term as b^ℓ. Since our approach is from a Bayesian perspective, we first need to define an appropriate prior distribution over every weight parameter $w_{ij}^{(\ell)} \in W^\ell$ and bias term b^ℓ, for every layer $\ell = 1, ..., \mathcal{L}$. A widely used approach is to start with a standard Gaussian distribution and tune the hyper-parameters using either a grid-search or a hierarchical model. Therefore, we select $P(W)$ as

$$P(W) = \prod_{j=1}^{J} p(w_j) = \prod_{j=1}^{J} \frac{1}{\sqrt{2\pi}} exp\{-\frac{1}{2} w_j^2\},$$

where each $w_j \in W$ is a single weight parameter out of a total of J. By using independent standard Gaussian priors for each weight parameter in every layer, we naturally impose regularisation to the trained model parameters. The zero-mean of the standard Gaussian shrinks the values of the weights and can be shown to be equivalent to L2 regularisation [1].

The preferred likelihood function depends on the task being performed by the model. Since deep learning is a popular choice in multi-class classification such as object recognition in images, we will use an appropriate likelihood function for multi-class classification. In this case, the softmax likelihood (as described in Section 6.3.1) is a valid choice and is defined as follows:

$$P(y = k|\underline{x}, W) = \frac{exp(f_k^W(\underline{x}))}{\sum_{k=1}^{K} exp(f_k^W(\underline{x}))} = \sigma(\underline{x})$$

where $f_k^W(\underline{x})$ indicates the output for class k from the NN using the set of weights W for input $\underline{x}$.

Now that we have specified the prior distributions over the weight parameters as well as the likelihood function prodiced by the NN's output, we define the posterior distribution of the weights as

$$\begin{aligned} P(W|X, \underline{y}) &= \frac{P(\underline{y}|X, W)P(W)}{\int P(\underline{y}|X, W)P(W)dW} \\ &= \frac{1}{Z} \frac{exp(f_k^W(\underline{x}))}{\sum_{k=1}^{K} exp(f_k^W(\underline{x}))} \prod_{j=1}^{J} \frac{1}{\sqrt{2\pi}} exp\{-\frac{1}{2} w_j^2\}. \end{aligned} \tag{6.20}$$

The term Z in Eq. 6.20 is the normalizing constant which is computationally intractable in most cases due to the large number of weight parameters which we need to integrate over. Instead, we use the methods discussed in Chapter 5 to calculate an approximation to the posterior distribution over the NN weights.

[1] See: "A Probabilistic Interpretation of Regularisation" http://bjlkeng.github.io/posts/probabilistic-interpretation-of-regularisation/

6.3.2 Example

We continue the example from Section 6.1.5 where we fit an NN with 2 hidden layers using ReLU activation functions, each containing 5 nodes. The output layer uses a sigmoid activation function to output values between 0 and 1. This requires us to estimate a total of 40 weight parameters, which causes the normalisation factor in the posterior distribution as shown in Eq. 6.20 to be computationally intractable. We shall use the ADVI algorithm to approximate the posterior distribution.

6.3.2.1 BNN with ADVI

The ADVI algorithm as described in Section 5.2.5 uses mean-field Gaussians as a variational distribution with which we approximate the posterior. The KL-divergence between the posterior and variational distributions is minimized by maximizing the expected lower bound (ELBO). Figure 6.12 shows the convergence of the ELBO as the ADVI algorithm finds optimal variational parameters.

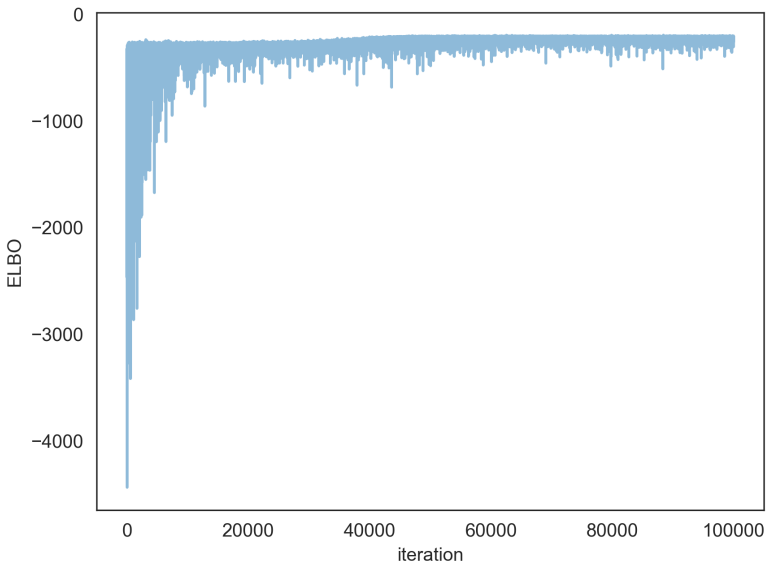


Figure 6.12: Convergence of ADVI algorithm

Having obtained a variational distribution which we use to approximate to the posterior distribution, we generate weight samples from it before deriving a posterior predictive distribution which we use to predict class probabilities on new data. Figure 6.13 shows the distribution of the sampled weights from each layer within the NN. The trace of the sampled values is illustrated on the right hand side which we obtained from the approximated posterior distribution. From this visualization we see that the standard Gaussian prior plays an important role in parameter regularisation since the estimated parameter values remain

small and centered around zero. This can prevent the model from over-fitting to the test data.

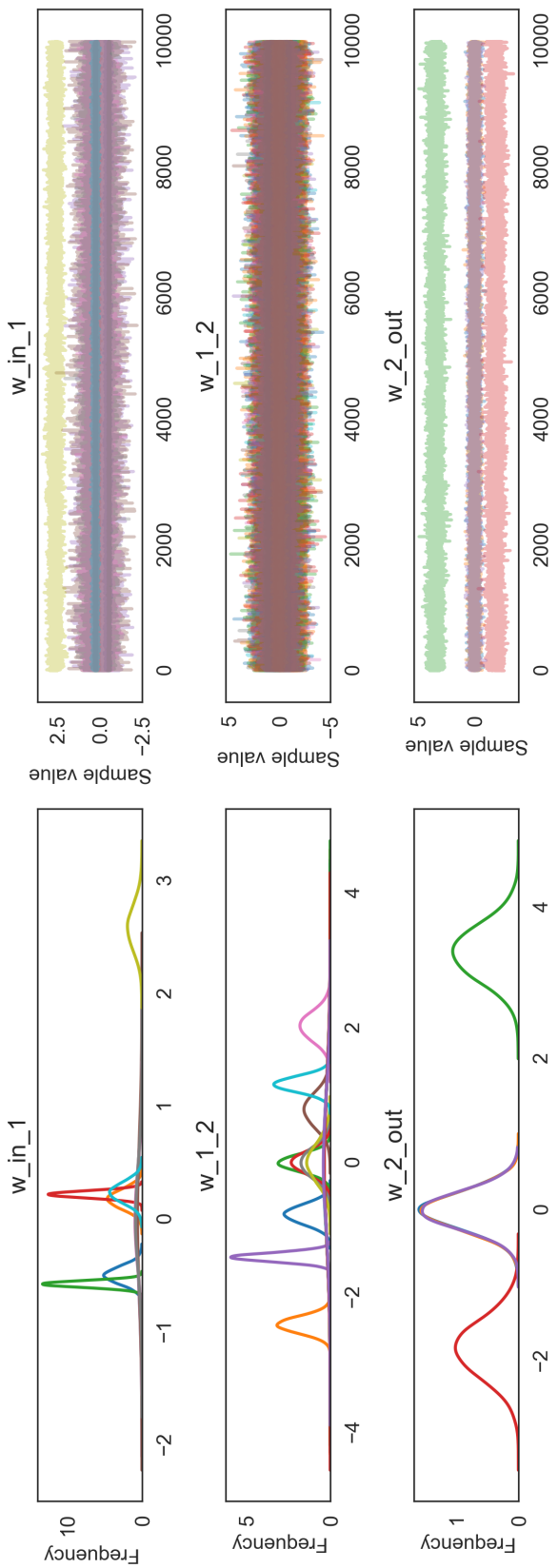


Figure 6.13: Weight parameters sampled from variational approximation to posterior

Using the sampled weight parameters, we derive a posterior predictive distribution over a set of test data and evaluate the performance of the BNN. The mean of each predictive distribution over a new observation can be used as a point estimate of its class probability and the standard deviation gives us a measure of uncertainty regarding the prediction.

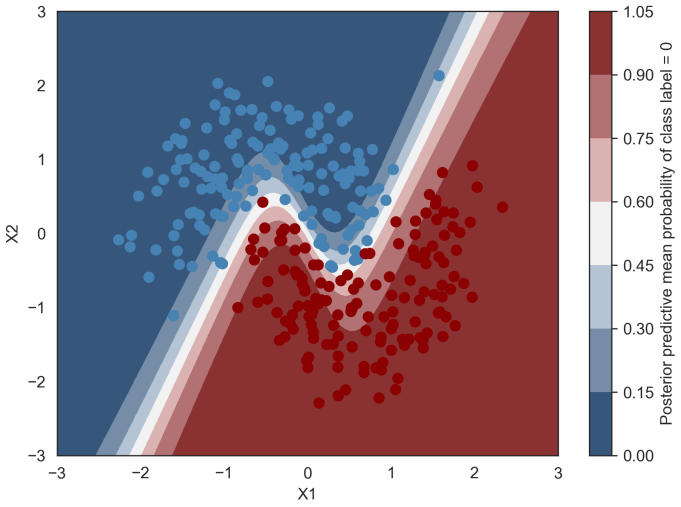
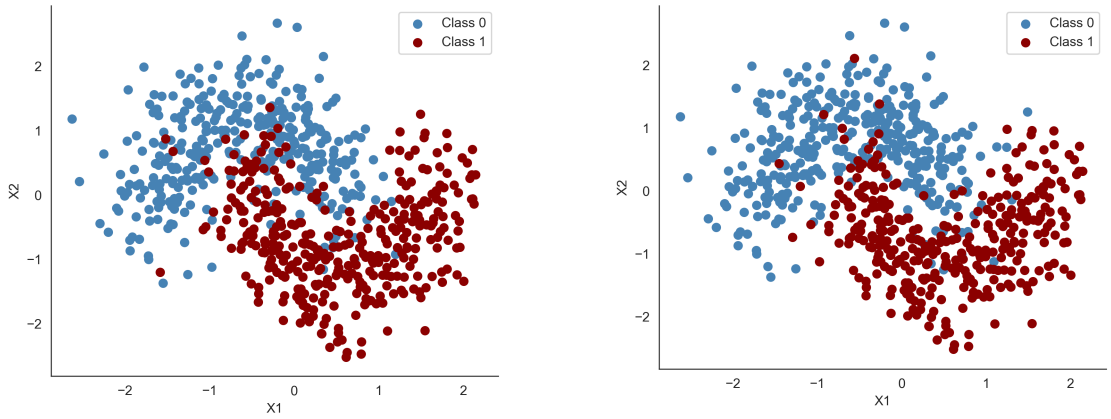


Figure 6.14: Posterior predictive mean values as class probabilities

As expected, the regularisation induced by the Gaussian priors caused the BNN to fit a more general curve over the data shown in Figure 6.14 than the curve shown in Figure 6.7. Although both the NN and BNN scored a classification accuracy of 92%, we have gained some advantages in approximating a posterior distribution over the parameters. We are able to perform a posterior predictive check (PPC), in which we generate data based on parameters sampled from the posterior.



(a) Training data

(b) Generated data from posterior

Figure 6.15: Comparison of training data and data generated from posterior predictive distribution.

By visually inspecting Figure 6.15, we see that the BNN has fit the data well since the classes generated from the posterior predictive distribution show similar behaviour to the training data. We are also able to measure our uncertainty around model prediction by using the posterior predictive distribution's standard deviation. Figure 6.16 shows a heat map of the uncertainty when predicting class labels for the test set. Observations that lie close to the decision boundary as well as the center of each cluster carry higher uncertainty and given the non-linear nature of the data, higher uncertainty is found at the decision boundary further away from the data.

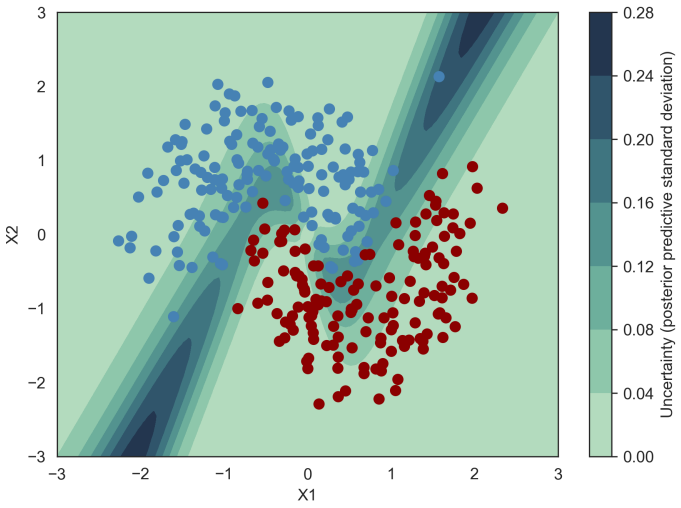


Figure 6.16: Posterior predictive standard deviation as uncertainty measure over predicted class labels for test set.

In conclusion, the BNN applied easier regularisation with standard Gaussian priors on the weights and fit a smooth curve over the data that generalises well to future observations. The posterior predictive check shows that the variation of the data has been captured well since we are able to generate class labels similar to those observed in the training set. Furthermore, we have gained an understanding of which observations we are more uncertain about when predicting class labels for new data.

Up until now we have only used simple, low dimensional examples for ease of illustration and to visually demonstrate the decision curves that each model produces after training. In the next chapter, we compare traditional NNs with BNNs on a more complex problem of image processing, where the dimensionality of the data and number of weight parameters are significantly higher.

Chapter 7

Application to image classification

Image classification is a pattern recognition task which belongs to the computer vision domain and is widely used in machine learning research as well as industry applications. Many machine learning models are evaluated on image classification problems such as the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) for object recognitinon [16] or the MNIST dataset for handwritten character recognition [14].

In this chapter we evaluate the Bayesian Neural Network (BNN) and Bayesian Convolutional Neural Network (BCNN) on the MNIST dataset. As shown in Figure 7.1, each observation contains a 28×28 greyscale image and a corresponding target variable indicating the observation's class label.

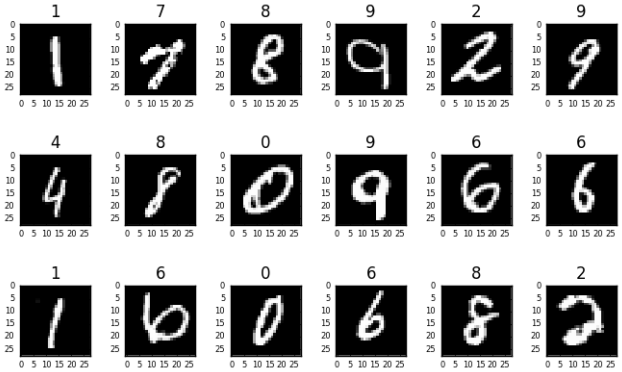


Figure 7.1: MNIST data examples.

7.1 Technical details

In this dissertation, all experiments and examples with a Bayesian approach were performed using the Python package PyMC3 [58]. The package has built-in functionality to perform approximation techniques such as MCMC or ADVI and utilizes the Theano [59] backend to declare and train NNs, allowing for fast computation using a Graphics Processing Unit (GPU).

For this experiment we used a training set of 50000 MNIST images and reserved a set of 10000 images to evaluate the model once training completed. For scalability, we trained the models using mini-batches of 1000 images each on the ADVI algorithm. The GPU used for experimentation was an Nvidia GTX 1080ti and we used Nvidia's CUDA v9 software [60] together with the Theano/PyMC3 backend to facilitate GPU training in Python.

7.2 BNN implementation

To illustrate the flexibility of the NN, we first approach the problem with a fully connected feedforward NN with 2 hidden layers, each containing 600 hidden nodes. A standard Gaussian prior distribution was declared over the weight parameters and a softmax likelihood is used from the model output to ensure class probability values that sum to 1. The posterior distribution was approximated by the ADVI algorithm and convergence was achieved within 1 hour and 15 minutes (illustrated in Figure 7.2). The BNN scored a classification accuracy of 95.14% on a test set containing 10000 handwritten images.

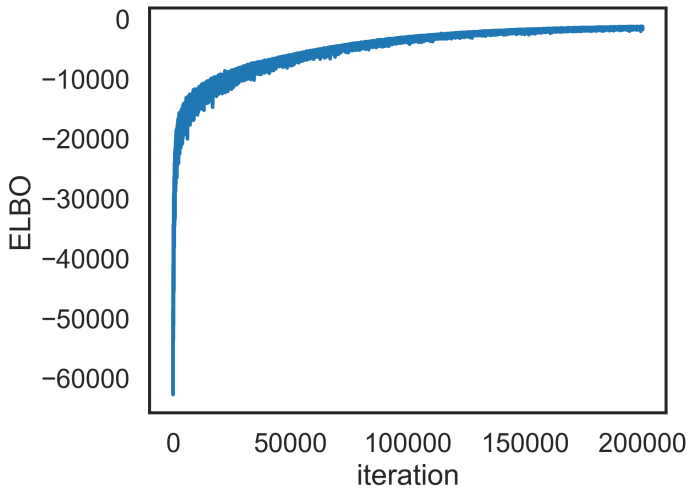


Figure 7.2: ADVI convergence for BNN.

7.2.1 Uncertainty

Using the posterior predictive distribution, we are able to extract information regarding the variability around predictions. Here we explore different methods of measuring prediction uncertainty.

These methods will be evaluated using a ratio which we will now define:

- Let $\mathcal{U}_{correct}$ be the collection of all uncertainty measurements from test observations which were correctly classified.
- Let $\mathcal{U}_{misclassified}$ be the collection of all uncertainty measurements from test observations which were misclassified.
- Let $\alpha \in (0, 1)$ be a scalar which we call the evaluation strictness.
- Let $\mathcal{K}$ be a value such that the proportion of values in $\mathcal{U}_{misclassified} < \mathcal{K}$ is α, which we call the cut-off value.
- Let γ be the proportion of elements in $\mathcal{U}_{correct}$ with values smaller than $\mathcal{K}$, which will be the metric we use to evaluate the performance of an uncertainty measurement tool.

Intuitively, γ indicates how well the uncertainty measure separates misclassified observations from correctly classified ones.

For example, if we chose the strictness $\alpha = 0.1$, then 90% of all misclassified observations had uncertainty larger than $\mathcal{K}$ and $\gamma * 100\%$ of correctly classified observations produced uncertainty smaller than $\mathcal{K}$.

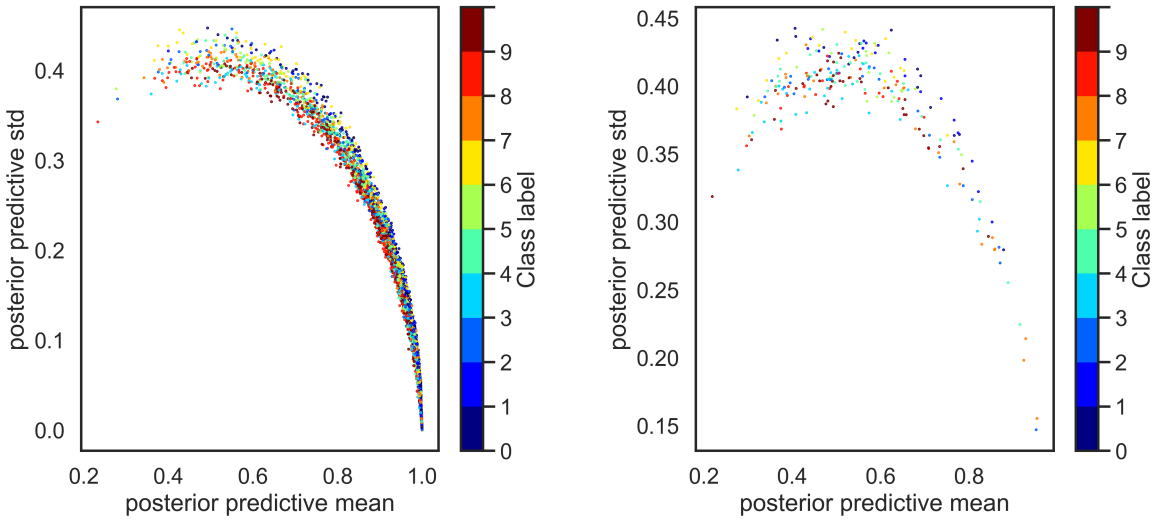
7.2.1.1 Posterior predictive standard deviation

Upon convergence of the ADVI algorithm, we are able to sample from the variational approximation of the posterior predictive distribution. For every new observation from the test set, we draw 1000 samples of class probabilities from the posterior predictive. This yields a $K \times 10$ matrix where K is the number of samples drawn from the posterior predictive and each column represents a predicted class probability for a single class. For a single observation, every class's uncertainty can be captured by calculating the standard deviation of the class prediction probabilities over the K samples.

For example, the posterior predictive standard deviation for observation i with respect to class j is calculated as follows

$$S_{p_{ij}} = \frac{1}{K-1} \sum_{k=1}^{K} p_k(y_i = j | X, \underline{w}), \tag{7.1}$$

where $p_k(y_i = j|X, \underline{w})$ indicates a sampled probability of observation i belonging to class j, out of a total of K samples.



(a) Uncertainty from correctly classified observations

(b) Uncertainty from misclassified observations

Figure 7.3: Comparison of uncertainty from correctly classified vs missclassified observations.

Figure 7.3 illustrates the relationship between the predicted label probabilities from observations and their corresponding standard deviations obtained from the posterior predictive distribution. From this uncertainty metric we see that class-probabilities in the interval $[0.4, 0.6]$ carry higher uncertainty and that the majority of misclassified observations produced an uncertainty measurement near the maximum of the curve.

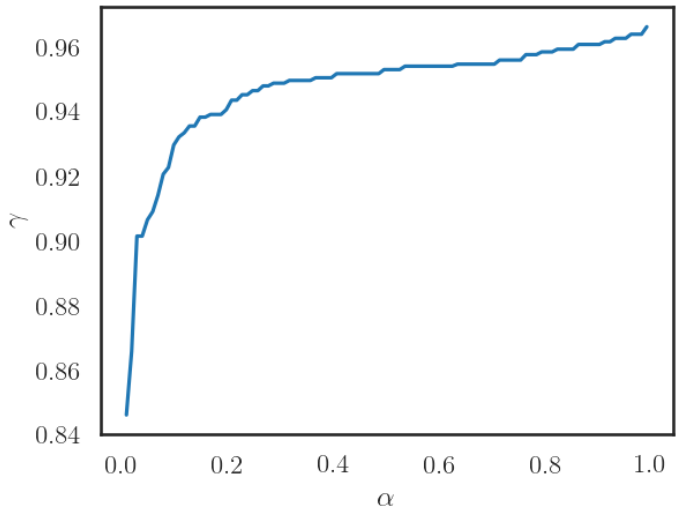


Figure 7.4: γ metric with varying evaluation strictness α

Figure 7.4 illustrates how the evaluation rating γ increases with α. For an uncertainty measure that doesn't separate correctly classified observations from misclassifications, we expect γ to increase linearly with α. The posterior predictive standard deviation uncertainty measure provides satisfying insight into observations that are more likely to be incorrectly classified. If we choose $\alpha = 0.1$, we get a γ-ratio of 0.866. This means that the uncertainty metric is able to create a boundary that separates $(1-\alpha)100\% = 90\%$ of the misclassifications from 86% of the correctly classified observations. As we increase the value of α, our γ-ratio would increase but would separate fewer misclassifications from correct predictions. Therefore, an α-value of 0.1 is a reasonable choice.

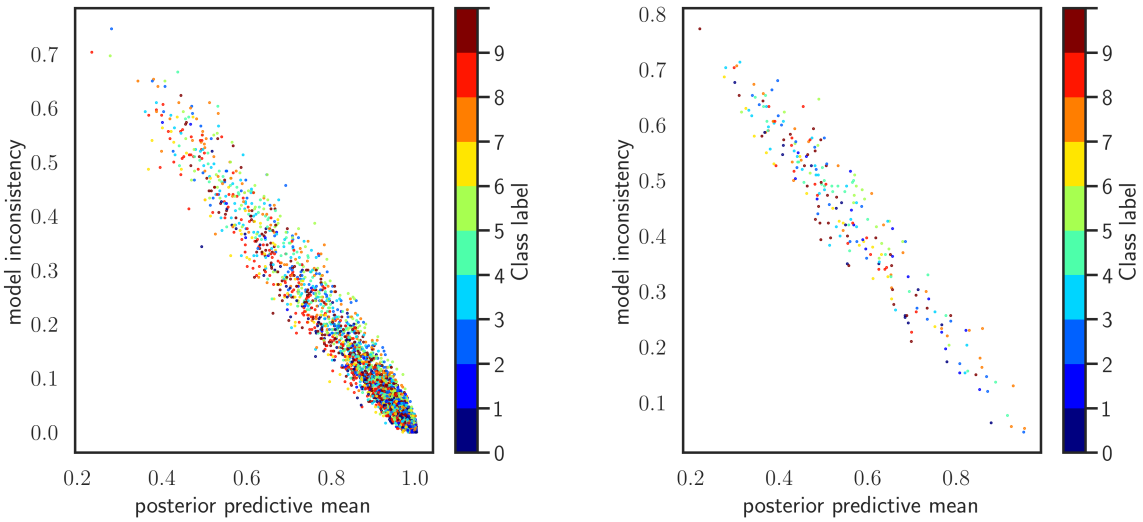
7.2.1.2 Model consistency using sampled class counts

A measure of model consistency can be obtained through a discrete count of sampled categorical outcomes. Instead of calculating the posterior predictive standard deviation for each class, the counts of sampled class *assignments* for a given observation can be used. This allows us to calculate a single measure of uncertainty for an observation as opposed to measuring each class's uncertainty.

The counts-based consistency uncertainty measure is calculated as follows:

- Draw K samples from the posterior predictive distribution, where each sample is a collection of 10 class-probabilities.
- For each sample, assign a predicted class (choose the class with the highest corresponding probability).
- We end up with a collection of K outcomes, each voting for a specific class-assignment for the given observation.

We then define $\mathcal{C}$ as the largest group of "voters" in the sample that produced identical class-predictions. Model consistency is then defined as the ratio $\frac{\mathcal{C}}{K}$.



(a) Consistency from correctly classified observations

(b) Consistency from misclassified observations

Figure 7.5: Comparison of uncertainty from correctly classified vs missclassified observations.

In order to express model consistency as a form of uncertainty, we instead calculated inconsistency as $1 - \frac{C}{K}$. As expected, the counts-based inconsistency shows a negative linear relationship between class-probability and model inconsistency. In other words, observations that produce a high probability for a certain class will show an equally low inconsistency measure since most sampled class-predictions will produce similar results.

The model inconsistency measure produced a γ-value close to the posterior predictive standard deviation. For $\alpha = 0.1$, the uncertainty measure scored 0.865.

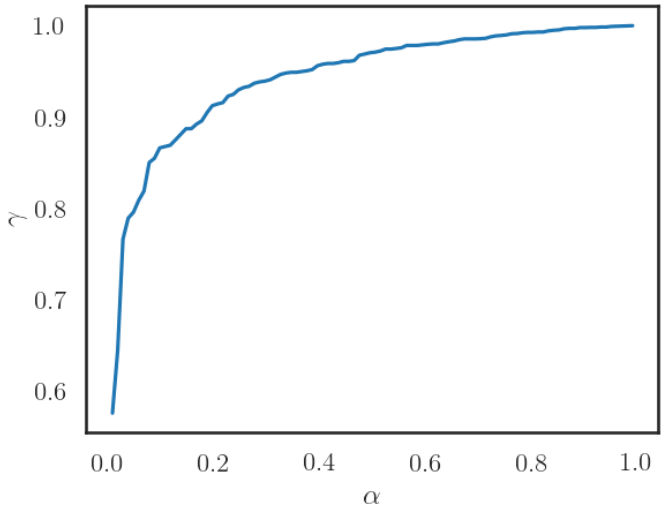


Figure 7.6: γ metric with varying evaluation strictness α

7.2.1.3 Gaussian Approximation (GA) uncertainty

In this section we use the counts-based random variable and model its behaviour by fitting a Binomial distribution to it. Using this distribution and large enough samples from the posterior predictive distribution, we derive a new uncertainty measure for categorical outcomes called Gaussian Approximation (GA) uncertainty.

Consider a random variable $\mathcal{G} = \mathcal{C} - 1$, where $\mathcal{C}$ is defined as in Section 7.2.1.2. We then have that

$$\mathcal{G} \in \{0, 1, 2, ..., K-1\}.$$

Since all our posterior samples are mutually independent, it follows that $\mathcal{G} \sim Bin(K-1, p)$ where p can be estimated as the proportion $\hat{p} = \frac{\mathcal{G}}{K-1} = \frac{\mathcal{C}-1}{K-1}$.

If we assume that $p \notin \{0, 1\}$, then for a large enough sample K, it follows that the distribution of $\mathcal{G}$ can be approximated as

$$\mathcal{G} \dot{\sim} \mathcal{N}((K-1)\hat{p}, (K-1)\hat{p}(1-\hat{p}))$$

Recall that, for small values of $\mathcal{C}$, the model is less consistent since it implies that many sampled predictions produce different results. We can therefore use the approximated Gaussian distribution of $\mathcal{G}$ to find a value in the lower-tail of $\mathcal{G}$'s distribution that separates 90% of misclassifications from correctly classified observations, as illustrated in Figure 7.7.

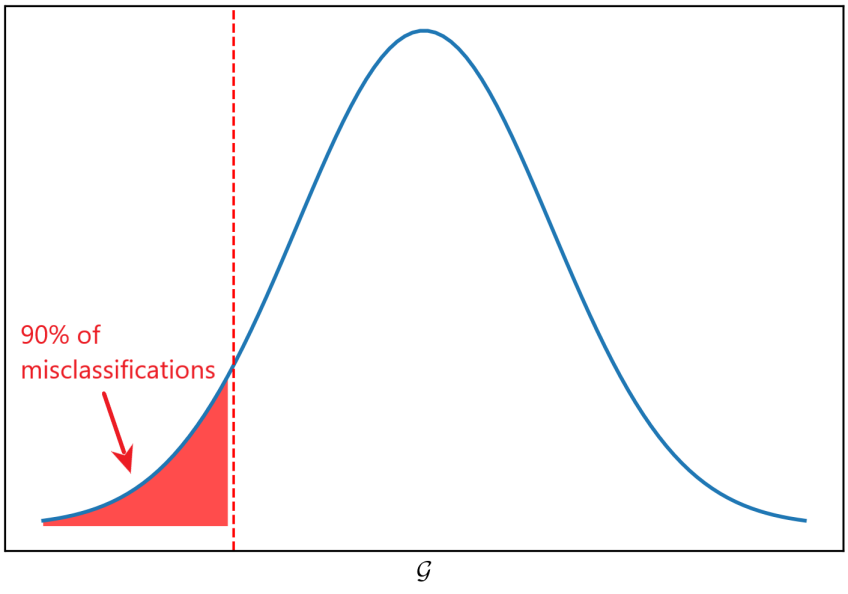


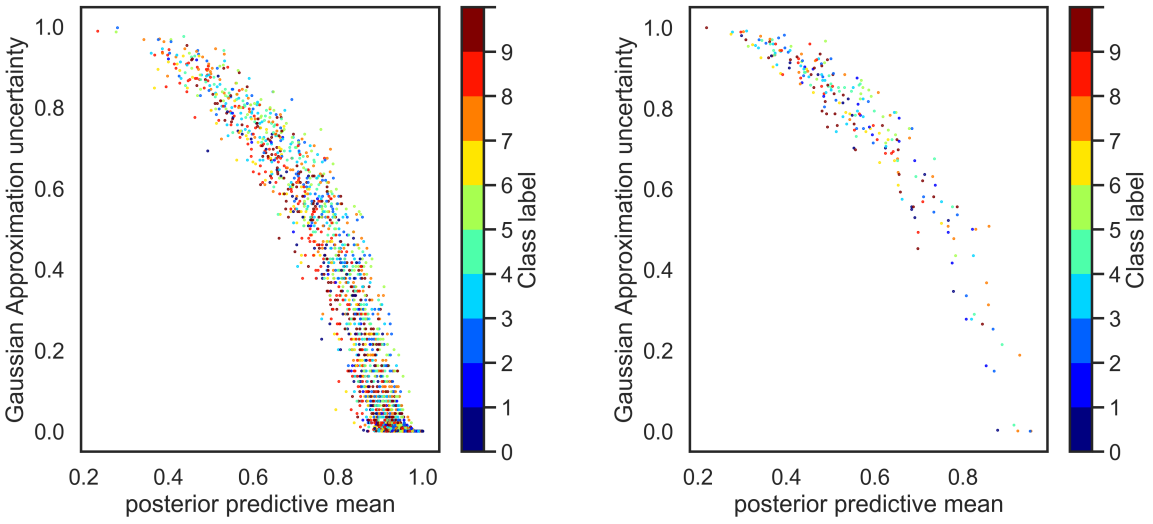
Figure 7.7: 90% of misclassifications cut-off point for $\mathcal{G}$

The uncertainty of an observation X (denoted as $\mathcal{U}_{GA}(X)$) can therefore be defined as the probability of

its associated $\mathcal{G}$-value falling in the above-mentioned lower tail area:

$$\mathcal{U}_{GA}(X) = P(\mathcal{G}_X < \epsilon),$$

where ϵ is the 90% cut-off value on the lower-tail of the distribution of $\mathcal{G}_X$.



(a) GA uncertainty from correctly classified observations (b) GA uncertainty from misclassified observations

Figure 7.8: Comparison of GA uncertainty from correctly classified vs missclassified observations.

The GA uncertainty measure scored a γ-value of 0.978, which (according to our γ metric) is the best performance between the three uncertainty measurements discussed in this section.

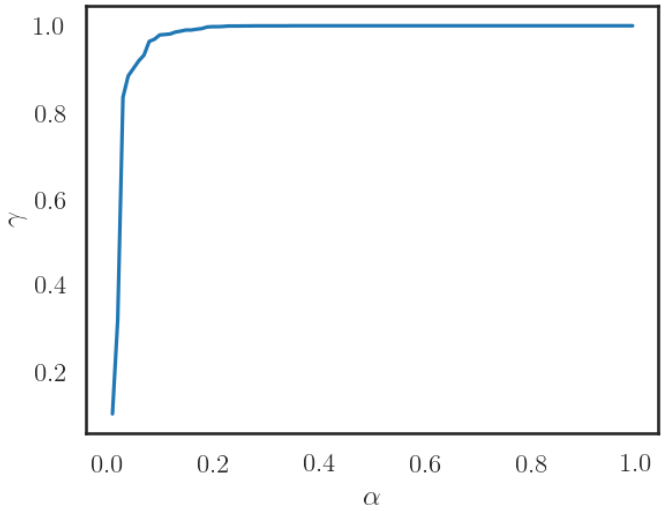


Figure 7.9: γ metric with varying evaluation strictness α

7.3 BCNN implementation

The CNN architecture used for this task is shown in Figure 7.10. The model includes two convolutional layers, each containing 32 filters with size 5×5 and stride 1 using ReLU activation functions. After each convolutional layer, a maxpooling layer is declared with a size and stride of 2. Following the second maxpool layer, the model is flattened into 288 nodes and passed to a fully connected (dense) hidden layer containing 256 nodes. A final softmax output layer is declared containing 10 nodes, each corresponding to the probability of a specific class label.

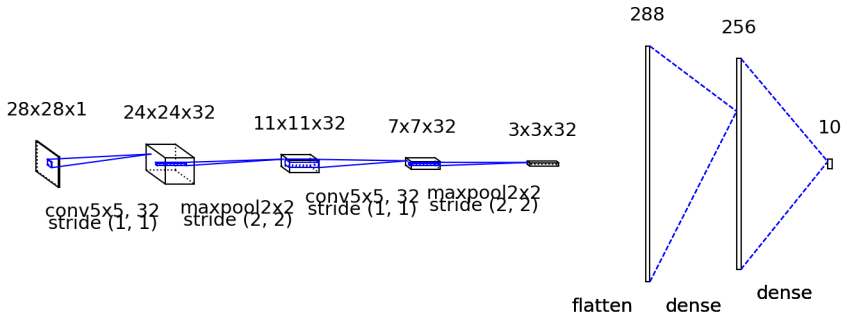


Figure 7.10: CNN architecture used for MNIST experiment.

Similar to the BNN, a standard Gaussian prior distribution was chosen over all the weight parameters and softmax likelihood from the output layer. Using the above design, the posterior distribution of the BCNN was approximated with ADVI algorithm using mean-field Gaussian approximation.

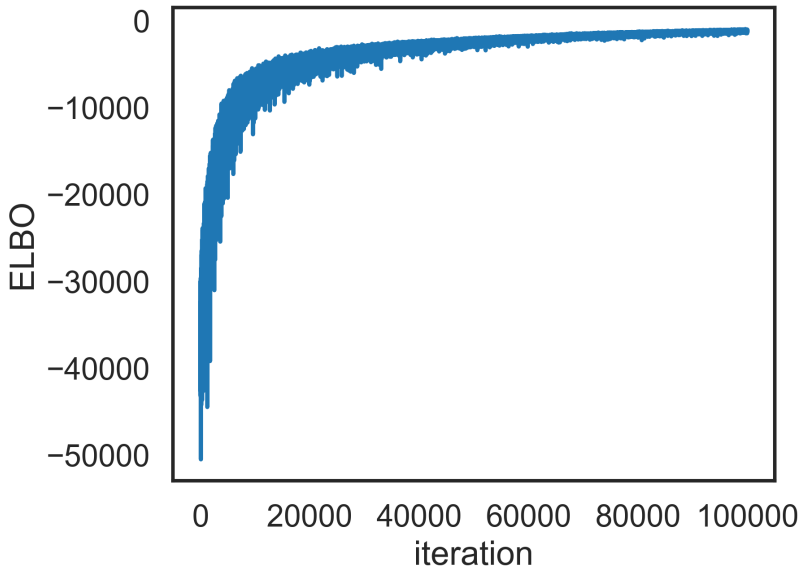


Figure 7.11: CNN architecture used for MNIST experiment.

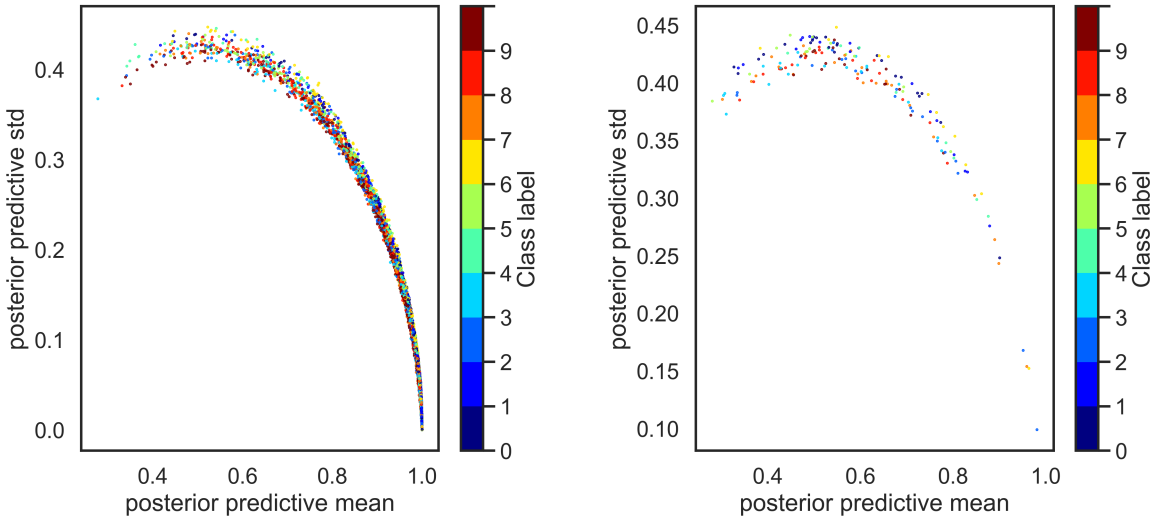
The ADVI algorithm converged at 100000 iterations for a duration of 1 hours and 34 minutes and achieved a classification accuracy of 98.18

7.3.1 Uncertainty

Similar to the BNN implementation, we explore the uncertainty measures from the output of the BCNN in this section.

7.3.1.1 Posterior predictive standard deviation

The posterior predictive standard deviation from the BCNN output in Figure 7.12 indicates that the class-probabilities in the interval $[0.4, 0.6]$ carry higher uncertainty.



(a) Uncertainty from correctly classified observations

(b) Uncertainty from misclassified observations

Figure 7.12: Comparison of uncertainty from correctly classified vs missclassified observations.

The posterior predictive standard deviation scored a γ-ratio value of 0.934, higher than the ratio found in the BNN model. This might indicate that higher model complexity may provide additional insight which we extract from the posterior distribution.

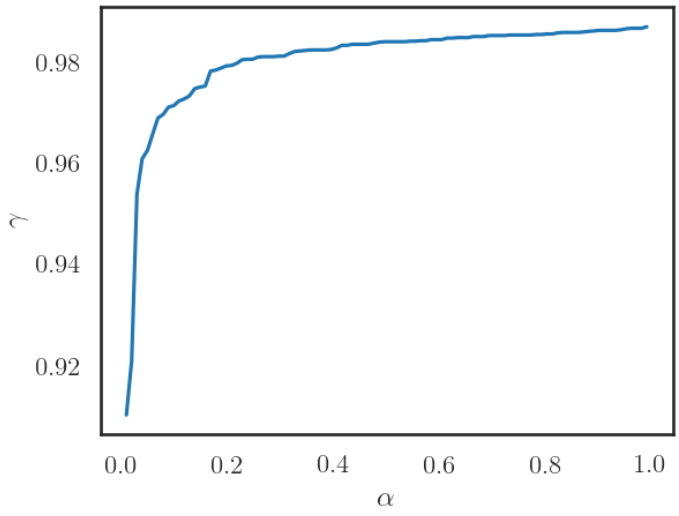
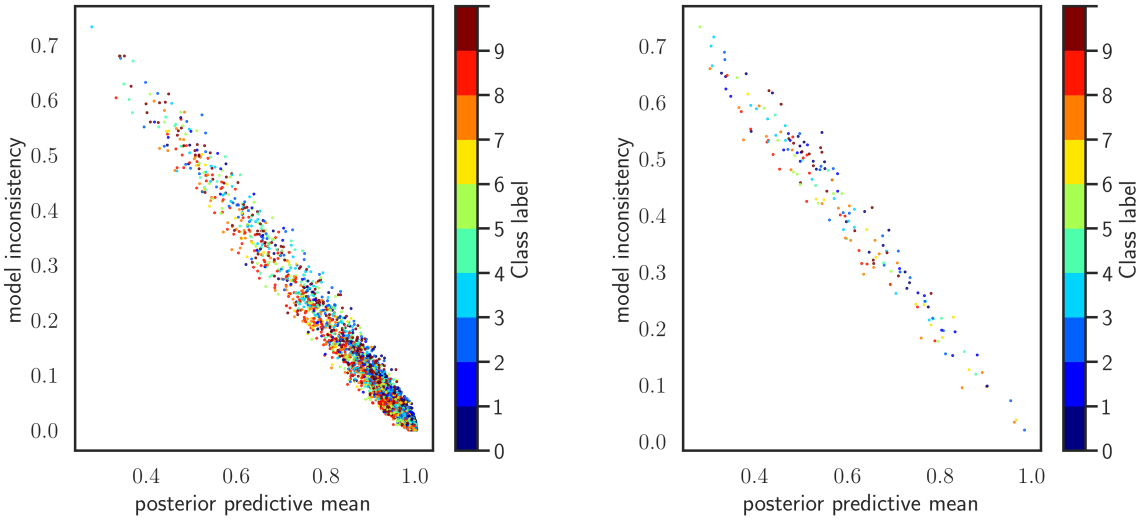


Figure 7.13: γ metric with varying evaluation strictness α

7.3.1.2 Model consistency using sampled class counts



(a) Consistency from correctly classified observations

(b) Consistency from misclassified observations

Figure 7.14: Comparison of uncertainty from correctly classified vs missclassified observations.

The model inconsistency from the BCNN measure produced a γ-value close to the posterior predictive standard deviation, similar to the result found for the BNN. For $\alpha = 0.1$, the uncertainty measure scored 0.931.

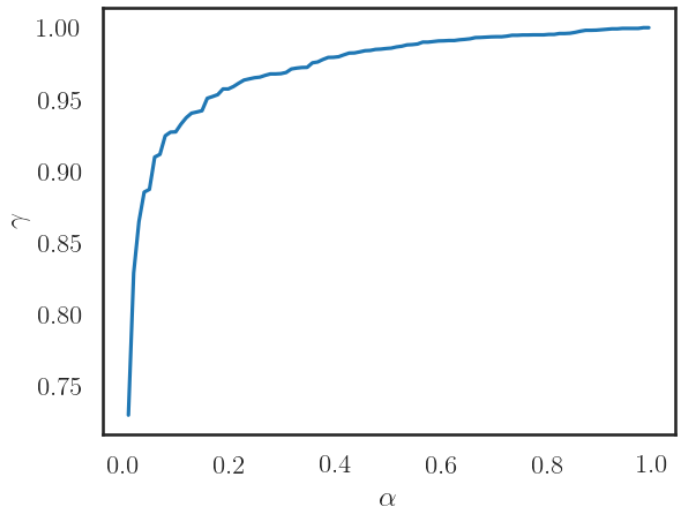
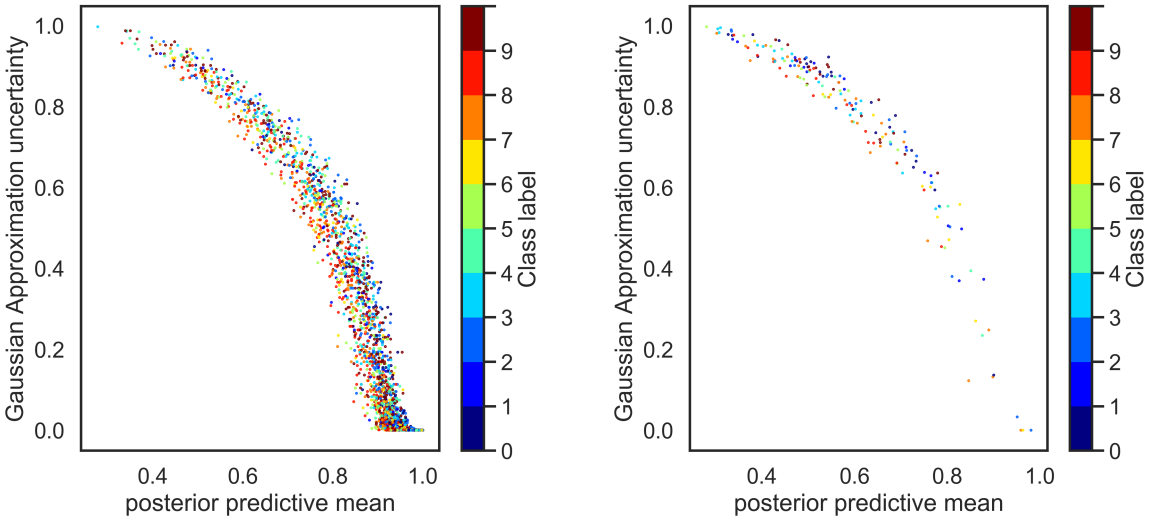


Figure 7.15: γ metric with varying evaluation strictness α

7.3.1.3 Gaussian Approximation (GA) uncertainty



(a) GA uncertainty from correctly classified observations (b) GA uncertainty from misclassified observations

Figure 7.16: Comparison of GA uncertainty from correctly classified vs missclassified observations.

The GA uncertainty measure scored a γ-value of 0.99, which is again the best performance between the three uncertainty measurements for the BCNN.

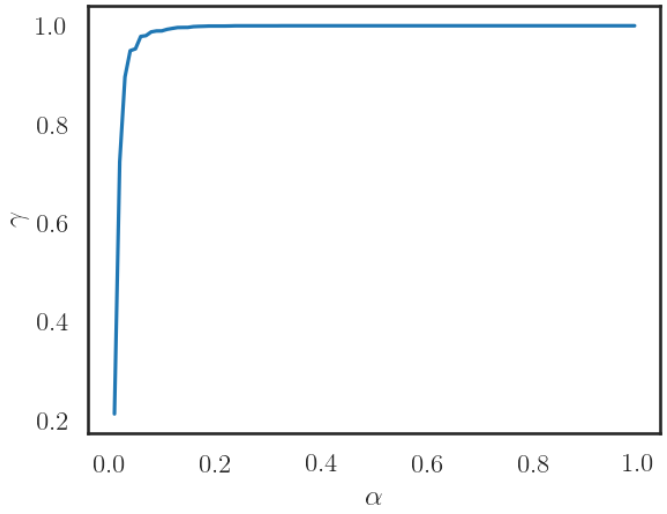
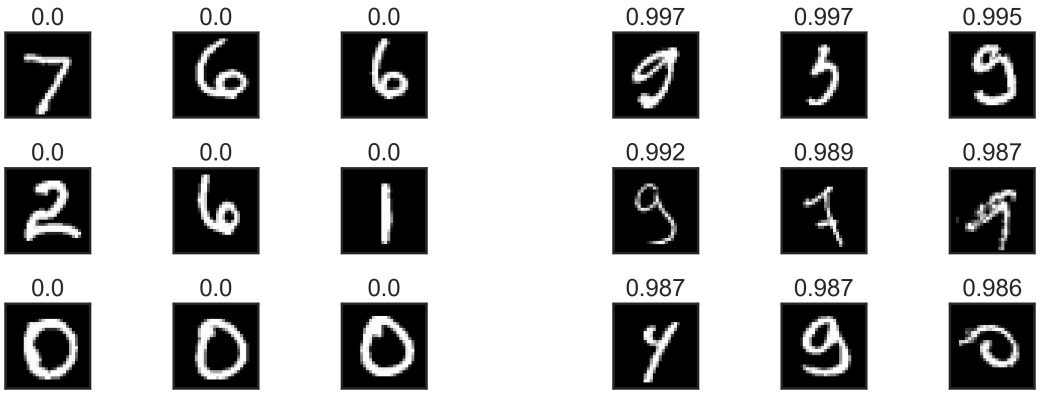


Figure 7.17: γ metric with varying evaluation strictness α

The GA uncertainty measure proved to be the most capable of separating correctly classified observations from misclassified observations. This information could prove useful in many different ways. For example, we could extract from the test set the most uncertain images and compare them to the images we were

most certain of, as shown in Figure 7.18.



(a) Most certain images

(b) Most uncertain images

Figure 7.18: Comparison of most certain vs most uncertain images using the GA uncertainty measure as indicated above each image.

Badly drawn digits clearly show a higher GA uncertainty value as excepted. The uncertainty measure displays an ability to highlight observations which may lie further away from the training data or closer to a decision boundary (in high dimensions) and are more likely to be misclassified by the model.

Chapter 8

Conclusion

The ultimate goal of this dissertation was to change the approach to deep learning from discriminative to generative. Put differently, the main focus was shifted from achieving the highest possible prediction accuracy to extracting valuable information from model output via Bayesian inference.

Following the literature review in Chapter **??** in which we discussed the history of NN research, we established the fundamental theory necessary to perform Bayesian inference in Chapter 3. It highlighted that caution should be exercised when selecting a prior for a specific problem since the additional information gained from Bayesian inference could be subjective when an informative prior is used. In Bayesian modelling, the main objective is to derive a posterior distribution over the unknown variables of interest, such as parameter values in a model. Using the posterior distribution, a posterior predictive distribution can be derived to better understand the behaviour of model predictions on new observations.

In Section 4.2 we discussed how to apply the theory from Chapter 3 to a discriminative linear classifier such as logistic regression. Doing so, we specified a non-informative prior distribution over the unknown parameter values, i.e. a standard Gaussian distribution, and used the sigmoid function as the likelihood of the data. For only an intercept parameter and a single coefficient used in the model, we were able to compute the exact posterior by calculating the integral over the two parameters directly. In this chapter it became clear that for simple models, calculating the exact posterior distribution can easily be done using Bayes' rule. However, for more complex models using many parameters, the formula for calculating the posterior becomes computationally intractable because of the multiple integrals required in the denominator (normalisation term). In order to proceed, we needed to find a method to obtain the posterior distribution without having to directly calculate the integrals.

In response, Chapter 5 discussed approximation methods used to avoid the above-mentioned computational challenge and derive the posterior distribution. Markov Chain Monte Carlo (MCMC) was intro-

duced first as the more popular technique among statisticians because of its ability to find the exact posterior distribution in the stationary distribution of the markov chain. However, MCMC does not scale well to very complex models on large sets of data as we often see in deep learning environments, even when using simplified methods such as Gibbs sampling. Furthermore, it is difficult to measure convergence of MCMC methods since we are not optimizing an objective function - we are only following the trace of a Markov Chain until it reaches a stationary distribution. The alternative method introduced was Variational Inference (VI) which is designed as an optimisation problem rather than a sampling method. Although VI does not guarantee the exact posterior as MCMC does, it offers some advantages. VI is scalable to complex models when using mean-field VI because of it's independence assumptions and is able to optimise the objective function using mini-batch training to overcome large sets of data.

Neural Networks were formally introduced in Chapter 6 with a graphical representation of a small network and a mathematical derivation of the backpropagation algorithm used to train it with. NNs gain most of their scalability from the optimisation methods we use in order to perform backpropagation. The time-tested Gradient Descent algorithm faces computational problems in modern applications due to the large volume of data deep learning models are trained on. Stochastic Gradient Descent (SGD) was developed as a scalable solution to data-intensive training algorithms in which we train the model in smaller batches observations instead of all the data at once. To prevent over-fitting the data with a traditional feedforward NN, we introduced the Dropout technique and applied it to a simple non-linear classification example. The Convolutional Neural Network (CNN) was the final frequentist model we introduced in this dissertation and was implemented in Chapter 7 on the MNIST dataset.

We followed with a probabilistic approach to the NN model in Section 6.3.1 and defined the Bayesian Neural Network (BNN). Similar to logistic regression, we defined a standard Gaussian prior distribution over all unknown weight parameters in the NN. The output of the simple NN used in Section 6.1.5 was a class-probability for a binary outcome. To that end, a sigmoid activation function was sufficient. The BNN showed similar prediction accuracy to the dropout NN and provided useful information relating to uncertainty which we extracted from the posterior predictive distribution. We saw that observations near the decision boundary and further from the training data carried high uncertainty as well as areas where classes couldn't be separated by the decision boundary without having to over-fit to the training data.

In Chapter 7 we defined a deep Bayesian feedforward NN and a Convolutional Bayesian Neural Network (CBNN) and tested them using the MNIST handwritten digits dataset. The likelihood from the output of the Bayesian models were chosen as the softmax function to account for multiple class-probabilities and allow them to sum to 1. The posterior distributions for both Bayesian models were approximated using Automatic Differentiation Variational Inference (ADVI). The BCNN scored an accuracy of over 98% which is similar to state-of-the-art traditional CNNs trained with gradient-based methods. However, the ADVI algorithm required more time to converge in comparison to a traditional CNN trained with SGD

with the same batch-size. This is most likely due to the fact that the update-size with ADVI is more difficult to configure than with SGD, which causes to ADVI to require a large number of iterations before convergence.

For each test observation, we drew 1000 samples from the posterior predictive distribution. Using these samples we proposed three different uncertainty measurements, namely:

- posterior predictive standard deviation for class-specific uncertainty with each observation
- counts-based model consistency using the categorical outcomes from posterior predictive samples.
- Gaussian Approximation (GA) uncertainty: using the Gaussian approximation of the Binomial distribution from each sample of categorical outcomes for a test observation.

The GA uncertainty measure scored the highest performance in its ability to separate the correctly classified test observations from the misclassified ones. However, for different problems to MNIST, we may need to define new uncertainty measures which fit the task at hand.

In conclusion, Bayesian Neural Networks are a powerful tool to use in deep learning environments when accurate predictions are not the only important factor. With NN implementations surfacing in the medical and autonomous vehicles industries, we are in need of deep learning methods which could supplement more informed decision-making in addition to accurate forecasting. As proved in this dissertation, powerful methods exist which we could use to approximate the posterior of deep BCNNs and extract useful information to better understand the uncertainty within deep learning.

8.1 Contribution

In this dissertation we compared existing methods designed to approximate probability distributions in the context of BNNs. We extended the use of the ADVI algorithm to a CNN architecture for handwritten digit recognition and extracted uncertainty information from the posterior predictive distribution. Using this uncertainty information we demonstrated the additional inference that a BCNN provides above that of a traditional BNN.

The metric which we used to evaluate uncertainty measurements as well as the GA uncertainty for classification models are methods that are unique to this dissertation. For a classification problem, the GA uncertainty measure allows the isolation of observations deemed uncertain by the classification model and can be used to flag future data in need of more attention.

8.2 Limitations and future work

The variational method which we used to approximate the posterior distribution of a BNN has demonstrated satisfying results when applied to complex models such as the BCNN. However, current software that allows the ADVI algorithm to approximate a posterior distribution using a GPU with mini-batch iterations remains computationally expensive despite being more efficient than popular MCMC sampling methods. Training a CNN from a frequentist approach reaches convergence significantly faster than from a Bayesian perspective. In addition to computational challenges, there is limited research currently available around uncertainty measures from the output of a Bayesian multi-class classification model.

There is an opportunity for future research endeavours to investigate the scalability to more complex models when implementing ADVI on a GPU. In particular, more sophisticated learning rate calibration when updating variational parameter values as well as utilizing multiple GPU's to reach convergence faster would prove beneficial to BNN research. Further research into the GA uncertainty measure could provide interesting results, such as a Kolmogorov-Smirnov goodness-of-fit test for distributions over posterior predictive samples in a classification environment.

Bibliography

[1] N. Ganesan, K. Venkatesh, M. Rama, and A. M. Palani, "Application of neural networks in diagnosing cancer disease using demographic data," *International Journal of Computer Applications*, vol. 1, no. 26, pp. 76–85, 2010.

[2] S. Lawrence, C. L. Giles, A. C. Tsoi, and A. D. Back, "Face recognition: A convolutional neural-network approach," *IEEE transactions on neural networks*, vol. 8, no. 1, pp. 98–113, 1997.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[4] T. Mikolov, M. Karafiát, L. Burget, J. Černockỳ, and S. Khudanpur, "Recurrent neural network based language model," in *Eleventh Annual Conference of the International Speech Communication Association*, 2010.

[5] Y. LeCun, P. Haffner, L. Bottou, and Y. Bengio, "Object recognition with gradient-based learning," in *Shape, contour and grouping in computer vision*, pp. 319–345, Springer, 1999.

[6] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[7] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain.," *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[8] A. B. Novikoff, "On convergence proofs for perceptrons," tech. rep., STANFORD RESEARCH INST MENLO PARK CA, 1963.

[9] M. Minsky, S. A. Papert, and L. Bottou, *Perceptrons: An introduction to computational geometry*. MIT press, 1969.

[10] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," tech. rep., California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[11] A. Graves, M. Liwicki, S. Fernández, R. Bertolami, H. Bunke, and J. Schmidhuber, "A novel connectionist system for unconstrained handwriting recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 31, no. 5, pp. 855–868, 2009.

[12] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pp. 6645–6649, IEEE, 2013.

[13] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[14] Y. LeCun, "The mnist database of handwritten digits," *http://yann. lecun. com/exdb/mnist/*, 1998.

[15] P. Y. Simard, D. Steinkraus, J. C. Platt, *et al.*, "Best practices for convolutional neural networks applied to visual document analysis.," in *ICDAR*, vol. 3, pp. 958–962, 2003.

[16] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 248–255, IEEE, 2009.

[17] 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*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[18] T. M. Therneau, "rpart: Recursive partitioning. r package version 3," *http://www. mayo. edu/hsr/Sfunc. html*, pp. 1–23, 2005.

[19] V. Gulshan, L. Peng, M. Coram, M. C. Stumpe, D. Wu, A. Narayanaswamy, S. Venugopalan, K. Widner, T. Madams, J. Cuadros, *et al.*, "Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs," *Jama*, vol. 316, no. 22, pp. 2402–2410, 2016.

[20] H.-T. L. Chiang, A. Faust, and L. Tapia, "Deep neural networks for swept volume prediction between configurations," *arXiv preprint arXiv:1805.11597*, 2018.

[21] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, "Tensorflow: A system for large-scale machine learning.," in *OSDI*, vol. 16, pp. 265–283, 2016.

[22] D. F. Specht, "Probabilistic neural networks," *Neural networks*, vol. 3, no. 1, pp. 109–118, 1990.

[23] A. Kendall and Y. Gal, "What uncertainties do we need in bayesian deep learning for computer vision?," in *Advances in Neural Information Processing Systems*, pp. 5580–5590, 2017.

[24] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[25] S. J. Nowlan and G. E. Hinton, "Simplifying neural networks by soft weight-sharing," *Neural computation*, vol. 4, no. 4, pp. 473–493, 1992.

[26] Y. Gal, "Uncertainty in deep learning," *University of Cambridge*, 2016.

[27] J. S. Denker and Y. Lecun, "Transforming neural-net output levels to probability distributions," in *Advances in neural information processing systems*, pp. 853–859, 1991.

[28] N. Tishby, E. Levin, and S. A. Solla, "Consistent inference of probabilities in layered networks: Predictions and generalization," in *IJCNN International Joint Conference on Neural Networks*, vol. 2, pp. 403–409, IEEE New York, 1989.

[29] R. M. Neal, *Bayesian learning for neural networks*, vol. 118. Springer Science & Business Media, 1995.

[30] G. E. Hinton and D. Van Camp, "Keeping the neural networks simple by minimizing the description length of the weights," in *Proceedings of the sixth annual conference on Computational learning theory*, pp. 5–13, ACM, 1993.

[31] N. De Freitas, P. Højen-Sørensen, M. I. Jordan, and S. Russell, "Variational mcmc," in *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*, pp. 120–127, Morgan Kaufmann Publishers Inc., 2001.

[32] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *The journal of chemical physics*, vol. 21, no. 6, pp. 1087–1092, 1953.

[33] W. K. Hastings, "Monte carlo sampling methods using markov chains and their applications," *Biometrika*, vol. 57, no. 1, pp. 97–109, 1970.

[34] L. Tierney, "Markov chains for exploring posterior distributions," *the Annals of Statistics*, pp. 1701–1728, 1994.

[35] A. Gelman and D. B. Rubin, "Inference from iterative simulation using multiple sequences," *Statistical science*, pp. 457–472, 1992.

[36] C. Robert and G. Casella, *Monte Carlo statistical methods*. Springer Science & Business Media, 2013.

[37] C. Peterson, "A mean field theory learning algorithm for neural networks," *Complex systems*, vol. 1, pp. 995–1019, 1987.

[38] H. Kushner and G. G. Yin, *Stochastic approximation and recursive algorithms and applications*, vol. 35. Springer Science & Business Media, 2003.

[39] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe, "Variational inference: A review for statisticians," *Journal of the American Statistical Association*, vol. 112, no. 518, pp. 859–877, 2017.

[40] C. E. Rasmussen, "Gaussian processes in machine learning," in *Advanced lectures on machine learning*, pp. 63–71, Springer, 2004.

[41] J. Lee, Y. Bahri, R. Novak, S. S. Schoenholz, J. Pennington, and J. Sohl-Dickstein, "Deep neural networks as gaussian processes," *arXiv preprint arXiv:1711.00165*, 2017.

[42] A. Y. Ng and M. I. Jordan, "On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes," in *Advances in neural information processing systems*, pp. 841–848, 2002.

[43] K. P. Murphy, "Machine learning: A probabilistic perspective. adaptive computation and machine learning," 2012.

[44] J. Friedman, T. Hastie, and R. Tibshirani, *The elements of statistical learning*, vol. 1. Springer series in statistics New York, 2001.

[45] D. A. Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," in *Advances in neural information processing systems*, pp. 305–313, 1989.

[46] E. Jones, T. Oliphant, and P. Peterson, "{SciPy}: open source scientific tools for {Python}," 2014.

[47] J. Dongarra and F. Sullivan, "Guest editors' introduction: The top 10 algorithms," *Computing in Science & Engineering*, vol. 2, no. 1, pp. 22–23, 2000.

[48] S. Kullback and R. A. Leibler, "On information and sufficiency," *The annals of mathematical statistics*, vol. 22, no. 1, pp. 79–86, 1951.

[49] A. Kucukelbir, D. Tran, R. Ranganath, A. Gelman, and D. M. Blei, "Automatic differentiation variational inference," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 430–474, 2017.

[50] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, "Automatic differentiation in machine learning: a survey.," *Journal of machine learning research*, vol. 18, no. 153, pp. 1–153, 2017.

[51] L. Bottou, "Stochastic gradient descent tricks," in *Neural networks: Tricks of the trade*, pp. 421–436, Springer, 2012.

[52] M. D. Hoffman, D. M. Blei, C. Wang, and J. Paisley, "Stochastic variational inference," *The Journal of Machine Learning Research*, vol. 14, no. 1, pp. 1303–1347, 2013.

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

[54] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.

[55] D. E. Rumelhart, R. Durbin, R. Golden, and Y. Chauvin, "Backpropagation: The basic theory," *Backpropagation: Theory, architectures and applications*, pp. 1–34, 1995.

[56] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, p. 436, 2015.

[57] D. H. Hubel and T. N. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex," *The Journal of physiology*, vol. 160, no. 1, pp. 106–154, 1962.

[58] J. Salvatier, T. V. Wiecki, and C. Fonnesbeck, "Probabilistic programming in python using pymc3," *PeerJ Computer Science*, vol. 2, p. e55, 2016.

[59] R. Al-Rfou, G. Alain, A. Almahairi, C. Angermueller, D. Bahdanau, N. Ballas, F. Bastien, J. Bayer, A. Belikov, A. Belopolsky, *et al.*, "Theano: A python framework for fast computation of mathematical expressions," *arXiv preprint*, 2016.

[60] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with cuda," in *ACM SIGGRAPH 2008 classes*, p. 16, ACM, 2008.