Abstract

In this work, an approximate Bayesian approach for Deep Learning is compared with a conventional approach, all within a context of Active Learning. The conventional approach is based on the optimization of a loss function, which results in a single point estimate with no measure of uncertainty in the prediction or the parameters. These conventional methods are the norm in Deep Learning literature. The Bayesian approach, on the other hand, is seldom used, but has the advantage that it gives a joint posterior distribution of the parameters. This distribution of the parameters results in a posterior predictive distribution of the response variable. However, an exact implementation is too onerous, hence an approximation to the Bayesian method is made. This approximation allows a feasible simulation of a distribution that is close to the posterior distribution.

Recent work showed that using dropout in neural networks is equivalent to a variational approximation to a Bayesian neural network [19]. Hence offering a scalable approach to sampling from the posterior distribution. This approach can be used with Convolutional Neural Networks, an architecture widely used in computer vision tasks.

The approximate Bayesian and frequentist CNNs are compared in three image datasets, using the information of a trained model to query the most useful images from a large pool of unlabeled data to add to a set of training data and achieve higher accuracy than a random selection of images. Our results show that using the trained model's uncertainty is better than randomly choosing images, but they also show that there is no evidence in favor of the variational approximation over the frequentist methodology.

Contents

List of Figures

List of Figures

List of Figures

Chapter 1

Introduction

The main goal of this work is to compare approximate Bayesian methods with conventional methods used in Deep Learning within an Active Learning context. Conventional methods are based on the optimization of a loss function, which is rooted in maximum likelihood estimation. Optimization provides only a single point estimate with no measure of uncertainty, in particular for the parameters. On the other hand, the approximate Bayesian approach gives a joint posterior distribution of the parameters, thus giving a posterior predictive distribution of the response variable as well. It is well known that Bayesian methods avoid overfitting as they average over parameter values. However, due to their computational complexity, Bayesian methods are seldom used in Deep Learning, in contrast with conventional methods which are widely used.

Bayesian methods for Artificial Neural Networks date back to the late 1980s and early 1990s. These first approaches focused on Markov Chain Monte Carlo (MCMC) as it generates, albeit asymptotically, samples from the posterior distribution of the parameters, such as Neal [41], Denker and Lecun [11] and MacKay [36]. These approaches are not usually used in Machine Learning due to their highly intensive computational burden. Nonetheless, there have been recent efforts in bridging Bayesian inference to Machine Learning through clever approximations

to the full posterior distribution. An example of this, is the work by Hernández-Lobato and Adams [26]. This algorithm relies on one-dimensional Gaussian distributions that approximate the marginal posterior distribution of the parameters of a neural network at each step of training. Another example is the work done by Graves [25], based on variational inference to approximate the posterior distribution of the parameters. However, these methods do not scale well to very big neural network architectures and data sets. They also have the disadvantage of only working with multi-layer perceptron architectures, making them impossible to use with more recent architectures such as Convolutional Neural Networks (CNNs) or Recurrent Neural Networks (RNNs).

Very recently, Gal and Ghahramani [20] showed that a neural network with arbitrary depth, with dropout applied before each weight layer, is equivalent to the variational approximation to a deep Gaussian process. Dropout is the name of a widely used stochastic regularization technique, which will be discussed in Chapter 2. The equivalence between a neural network that uses dropout and a deep Gaussian process comes from the fact that the loss function minimizes the Kullback Leibler divergence between an approximate distribution and the posterior of a deep Gaussian process. This means that approximate uncertainty estimates can be obtained with neural networks that use dropout without changing anything during training. The only difference comes at prediction time in which instead of doing a single forward pass and multiply each layer by a weight proportional to dropout probability, several forward passes with dropout must be done instead. They also showed in [21] that stochastic regularization techniques in arbitrary neural models can be seen as approximate variational inference in Bayesian Neural Networks.

This work is further extended by the same authors and they showed that the same ideas of dropout as a Bayesian approximation can be used in CNNs [18]. In particular, they showed that dropout can be seen as approximate variational inference in Bayesian Neural Networks, thus permitting the use of operations such as convolution and pooling in probabilistic models. The implementation is a matter of just performing dropout after each convolution layer at training, and by

performing several stochastic forward passes through the model. The CNN model is then used in an Active Learning environment, where the goal is to label images intelligently so that a model has good performance with fewer training examples [22]. Gal, Islam, and Ghahramani [22] are able to achieve 5% test error on the MNIST data set [34] with only 295 labeled images, and 1.64% test error with 1000 labeled images. They compare different acquisition functions and also compare the Bayesian paradigm with frequentist CNNs.

The dropout variational approach can also be used in Recurrent Neural Networks (RNNs), as shown by Gal and Ghahramani in [17]. In this paper, the authors give insight on how to use dropout with RNNs and apply it on Long Short-Term Memory (LSTM) and Gated recurrent unit (GRU) models, outperforming existing techniques in language modeling with the Penn Treebank data set.

An example of how the uncertainty provided by Bayesian Neural Networks can be used is shown by Li and Gal [35]. They use adversarial examples and check if the original image can be told apart from the modified one by examining the uncertainty representation of the Bayesian models. The deterministic Neural Networks predict the wrong label very confidently on these adversarial samples. Dropout models, while also predicting wrong labels, are uncertain about their predictions. They finish by stating that their results suggest that assessing the uncertainty of classification models can be used to identify adversarial examples, but much more research is needed to solve the difficulties faced with adversarial inputs.

In this work, the Bayesian approximation using dropout will be used and compared with conventional methods in the context of Active Learning, following and extending the work in [22]. First, the original paper is reproduced by training the exact same architecture and using the same acquisition functions in the same data set. Then, these ideas are tested in two different and more complex data sets.

The rest of this dissertation is organized as follows. In Chapter 2, a brief overview on Machine Learning, and statistical inference is provided. This will help clarify the main differences between maximum likelihood estimation and uncertainty updating for the unknown parameters of the model. In particular, posterior

sampling is useful to propagate parameter uncertainty into informative posterior predictive distributions. Chapter 3 studies the concept of Bayesian variational approximation to posterior distributions. In Chapter 4 the basic concepts of neural networks are introduced, and then extended to include the theory of convolutional neural networks. In Chapter 5, the concept of active learning is studied and the acquisition functions used in this work are introduced. Concluding remarks are provided in Chapter 7.

Chapter 2

Machine learning

Machine learning is a set of methods used to learn patterns in data, and then use these patterns to make predictions about new unseen data, or to make other decisions [40]. Such decisions include making predictions about new unseen data, clustering, or have an agent taking actions based on a reward signal [40, 55]. Machine learning has seen an increasing interest from the scientific and industry thanks to the advance of computing capabilities. Such computing capabilities have made possible most modern techniques. Computer power has been increasing exponentially since the 1960s, and this growth is known as Moore's law. It was first stated in 1965 [38, 39], claiming that the number of transistors in an integrated computer circuit doubles about every two years for half the cost. Moore's law has helped the development of faster computers, as well as making these computers available in all areas of everyday life. This has led, in turn, to an exponential increase in data availability which has resulted in increased data storage requirements. However, Moore's law is said to be at its limit nowadays. Meaning that it has been increasingly difficult to double the number of transistors in that two year time frame while also being inexpensive. This is mainly due to the physical limitations of heat dissipation and the smallest possible size of an electronic device [32]. This has led to different research directions in the field of computer architecture

to find new ways in which computing power can be enhanced without sacrificing costs. The limit of Moore's law is an important factor in machine learning. The reason is, that it is not enough to harness faster and better computing resources, but also to use them efficiently.

Machine learning is usually divided in supervised learning, unsupervised learning and reinforcement learning. The first involves learning a model that relates a response to a set of possible explanatory features. The second involves learning relationships and the structure of the data. The third involves learning a strategy for a decision maker in order to maximize their reward while interacting with an unknown environment. Reinforcement learning (RL) is similar to supervised learning. Nonetheless, the difference is that in the framework of RL, the learner is continually interacting with its environment. In this setting, the agent receives feedback after every action is taken. Whereas, in supervised learning the agent is expected to receive a dataset from an oracle which, hopefully, includes all the right answers. That is, in supervised learning there is an oracle that tells the learner the right answer, whereas in RL the oracle only tells it how to perform relative to past actions.

For example, in supervised learning one would like to estimate the income of a household given a dataset that includes a set of relevant predictors, such as zip code, home size, number of owned cars, etc. An example of unsupervised learning would be to use the information in the dataset to be able to group together similar households in terms of such predictors. This will hopefully uncover a natural structure in the dataset, where households will be grouped with respect to its characteristics. In this setting, it is expected that different groups will be qualitatively different from one another. An example of RL would be to have an agent learning how to invest in real estate; observing its rewards after the investment period. Recently, supervised learning and reinforcement learning have been combined to achieve harder goals for computers, such as beating the Go world champion [52]. In this work, supervised learning will be discussed in more detail. Eventually, in Chapter 5 supervised learning will be merged with unsupervised learning, giving rise to the main application of this thesis. That is, active learning,

a mix of using known structured data to uncover patterns from non-structured data.

As mentioned above, in supervised learning there is a response associated to a set of relevant features. In this setting, let y denote the response and $\boldsymbol{x} \in \mathbb{R}^p$ denote a vector containing the features available to the analyst. The dataset consists of n observations of such response stored in $\boldsymbol{y} \in \mathbb{R}^n$. The corresponding features are stored in a matrix $\boldsymbol{X} \in \mathbb{R}^{n \times p}$. Consequently, each feature will be denoted by column vectors $\boldsymbol{x}^{(k)} \in \mathbb{R}^n$ for $k \in \{1, \ldots, p\}$, and each datapoint will be represented as a row vector $\boldsymbol{x}_i \in \mathbb{R}^p$ for $i \in \{1, \ldots, n\}$. It is assumed that there exists a relationship between $\boldsymbol{y}$ and $\boldsymbol{X}$, which can be written as

$$\boldsymbol{y} = f(\boldsymbol{X}) + \boldsymbol{\varepsilon}, \tag{2.1}$$

where f is a fixed but unknown function and $\boldsymbol{\varepsilon}$ is a zero-mean random error term, independent of $\boldsymbol{X}$ [27, p. 16]. The goal is to find the best approximation to the function f, which is done with an estimate $\hat{f}$ [27, p. 17]. This estimate $\hat{f}$ is chosen from a family of functions $\mathcal{F}$. In machine learning $\mathcal{F}$ is known as the hypothesis set.

For practical convenience, the family $\mathcal{F}$ is assumed to be indexed by a finite dimensional vector $\boldsymbol{\theta}$. Thus allowing to think of $\mathcal{F}_{\boldsymbol{\theta}}$ as a family of approximations indexed by a finite set of unknown parameters. This changes the task of finding a candidate in an infinite dimensional space to a problem in a finite dimensional space such as $\mathbb{R}^d$. In general, it is desirable for $\hat{f}$ to keep the generalization error to a minimum, where the generalization error is defined by the error of predicting for any unseen vector $\boldsymbol{x}^*$, with $\boldsymbol{x}^*$ randomly sampled from the probability distribution of features. In other terms, the error is usually expressed as the cost (loss) of making a wrong prediction. Of course, the loss function for the generalization error is not known, as it depends on unseen data. However, one minimizes the empirical loss function defined by the set of training data available. That is, the empirical loss function is approximated using the available dataset of n observations $\mathcal{D} = \{(\boldsymbol{X}, \boldsymbol{y})\}$, where $\boldsymbol{X}$ and $\boldsymbol{y}$ have been defined above. Examples of $\mathcal{F}_{\boldsymbol{\theta}}$ are the linear regression model, the general additive model, the logistic regression model and the

neural network.

For example, a linear regression can be assumed. The loss function can be defined as the quadratic error function defined as

$$\hat{f} = \underset{f \in \mathcal{F}_{\boldsymbol{\theta}}}{\arg\min} \frac{1}{n} \sum_{i=1}^{n} (y_i - f(\boldsymbol{x_i}))^2, \tag{2.2}$$

where $\mathcal{F}_{\boldsymbol{\theta}}$ is the family of functions of the form $f_{\boldsymbol{\theta}}(\boldsymbol{X}) = \boldsymbol{X\theta}$ with $\boldsymbol{\theta} \in \mathbb{R}^p$. Thus, the estimate $\hat{f}$ is $\hat{f}(\boldsymbol{X}) = \boldsymbol{X}\hat{\boldsymbol{\theta}}$, where $\hat{\boldsymbol{\theta}}$ minimizes the error function. In particular, the linear regression model belongs to a type of methods called **parametric methods**. In these methods, the modeler makes an assumption about the functional form of f, which depends on a vector of parameters $\boldsymbol{\theta}$ of fixed dimension. After this, the modeler proceeds to train the model by choosing the vector of parameters that minimizes a previously selected loss function [27, p. 21].

An approach to the estimation of parameters that is different to the concept of error function minimization is the one of **maximum likelihood estimation**. This is a probabilistic approach in which a probabilistic model is assumed to the data generating mechanism. In the case of linear regression, it is commonly assumed that the error term in (2.1) has a Gaussian distribution such that $\varepsilon_i \sim \mathcal{N}\left(0, \sigma^2\right)$, then $y_i \sim \mathcal{N}\left(\boldsymbol{x_i}^T\boldsymbol{\theta}, \sigma^2\right)$ for $i \in \{1, \ldots, n\}$. It is also usually assumed that the observations $(\boldsymbol{X}, \boldsymbol{y})$ come from a random sample and are independent of one another. Therefore, the log-likelihood of the sample is

$$L(\boldsymbol{\theta}) = \sum_{i=1}^{n} \log \left[\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i - \boldsymbol{x_i}^T\boldsymbol{\theta})^2}{2\sigma^2} \right) \right]. \tag{2.3}$$

Maximum likelihood estimation uses the **likelihood principle**, which states that all the information given by a sample of data is entirely held within the likelihood function [24, 50]. Maximum likelihood estimation is just an implementation of the likelihood principle which consists in choosing the values for $\boldsymbol{\theta}$ so that they maximize the likelihood of the data, i.e., equation (2.3) [50]. Informally, this is choosing the values that maximize the probability of observing the given data [15, p. 31] [50]. With some algebra, it is fairly easy to see that maximizing equation (2.3) is equivalent to minimizing the squared error loss.

An alternative setting is that of classification. In this case, it is assumed that y_i follows a Bernoulli distribution, such that the probability of y_i of being 1 is $p_i(\boldsymbol{\theta})$, where $\boldsymbol{\theta}$ is the parameter that indexes the probability. Then, the likelihood function in this case is $\prod_{i=1}^{n} p_i(\boldsymbol{\theta})^{y_i} \left(1 - p_i(\boldsymbol{\theta})\right)^{1-y_i}$ and thus, the log-likelihood is

$$L(\boldsymbol{\theta}) = \sum_{i=1}^{n} \left[y_i \log \left(p_i(\boldsymbol{\theta}) \right) + (1 - y_i) \log \left(1 - p_i(\boldsymbol{\theta}) \right) \right]. \tag{2.4}$$

An alternative to maximum likelihood estimation is given by the **Bayesian approach**. In this setting, the uncertainty about the unknown parameters is stated as a probability distribution. First, a prior distribution on the parameters $\boldsymbol{\theta}$ is assumed, and then the knowledge about them is updated with data. That is, the modeler first uses a prior distribution $p\left(\boldsymbol{\theta}\right)$ to quantify the knowledge that they may have about $\boldsymbol{\theta}$, and then computes the posterior distribution of $\boldsymbol{\theta}$ given $\boldsymbol{X}$ and $\boldsymbol{y}$ using Bayes' theorem as such

$$p\left(\boldsymbol{\theta}|\boldsymbol{y}, \boldsymbol{X}\right) = \frac{p\left(\boldsymbol{y}|\boldsymbol{\theta}, \boldsymbol{X}\right) p\left(\boldsymbol{\theta}|\boldsymbol{X}\right)}{p\left(\boldsymbol{y}|\boldsymbol{X}\right)} = \frac{p\left(\boldsymbol{y}|\boldsymbol{\theta}, \boldsymbol{X}\right) p\left(\boldsymbol{\theta}|\boldsymbol{X}\right)}{\int p\left(\boldsymbol{y}|\boldsymbol{\theta}, \boldsymbol{X}\right) p\left(\boldsymbol{\theta}|\boldsymbol{X}\right) d\boldsymbol{\theta}}. \tag{2.5}$$

The posterior distribution represents the knowledge about $\boldsymbol{\theta}$ after the data has been observed. It is a compromise between the prior beliefs and the data. Note that the likelihood of the data $p\left(\boldsymbol{y}|\boldsymbol{\theta}, \boldsymbol{X}\right)$ is used in the numerator of equation (2.5). In fact, the Bayesian paradigm permits another implementation of the likelihood principle, just like maximum likelihood, but with the added benefit of including decision-related requirements [50].

An advantage of the Bayesian approach is that it uses the language of probability to mathematically describe the uncertainty on unknown parameters and other components in the modeling process. In addition, Bayesian inference is logically consistent while dealing with this uncertainty [9, 10, 28, 43]. A philosophical difference between the frequentist and the Bayesian approach is that in the latter, the degree of belief of the researcher is represented in the distribution of the parameters and hence, they are treated as random variables. Whereas in the former, they are treated as a fixed but unknown value [43].

Since the denominator of equation (2.5) does not depend on $\boldsymbol{\theta}$, as it is only a

normalizing constant, it is just usually described up to a proportionality constant

$$p\left(\boldsymbol{\theta}|\boldsymbol{y},\boldsymbol{X}\right) \propto p\left(\boldsymbol{y}|\boldsymbol{\theta},\boldsymbol{X}\right) p\left(\boldsymbol{\theta}|\boldsymbol{X}\right). \tag{2.6}$$

The posterior distribution is also used to predict the values of unobserved data. Let $\boldsymbol{x}^*$ be a vector of features for which a prediction of $\boldsymbol{y}^*$ is desired, then the **posterior predictive distribution** must be used, defined as

$$p\left(\boldsymbol{y}^*|\boldsymbol{X},\boldsymbol{y},\boldsymbol{x}^*\right) = \int p\left(\boldsymbol{y}^*|\boldsymbol{\theta},\boldsymbol{x}^*\right) p\left(\boldsymbol{\theta}|\boldsymbol{X},\boldsymbol{y}\right) d\boldsymbol{\theta}. \tag{2.7}$$

Note that this posterior distribution is, as it name implies, a distribution, i.e., it is not a single prediction but potentially infinitely many predictions, weighted by how probable each value is to be. If the goal is to have a point prediction $\hat{\boldsymbol{y}}^*$ for $\boldsymbol{x}^*$, then the modeler could use Decision Theory to find a point estimate. The idea is to specify a loss function $L(\boldsymbol{y}^*, \hat{\boldsymbol{y}}^*)$ that quantifies the loss of having an estimate $\hat{\boldsymbol{y}}^*$ when the real value is $\boldsymbol{y}^*$, and have the point estimate be the $\hat{\boldsymbol{y}}^*$ that minimizes this loss function. The optimal value for the squared loss $L(\boldsymbol{y}^*, \hat{\boldsymbol{y}}^*) = (\boldsymbol{y}^* - \hat{\boldsymbol{y}}^*)^2$ is the expected value of the posterior predictive distribution, that is,

$$\hat{\boldsymbol{y}}^* = \mathbb{E}_{\boldsymbol{y}^*|\boldsymbol{x}^*,\boldsymbol{y},\boldsymbol{X}}\left[\boldsymbol{y}\right]. \tag{2.8}$$

Another loss function is the absolute loss $L(\boldsymbol{y}^*, \hat{\boldsymbol{y}}^*) = |\boldsymbol{y}^* - \hat{\boldsymbol{y}}^*|$ which results in using the median of the posterior predictive distribution as a point estimate. Finally, the 0-1 loss function $L(\boldsymbol{y}^*, \hat{\boldsymbol{y}}^*) = \mathbb{I}(\boldsymbol{y}^* \neq \hat{\boldsymbol{y}}^*)$ results in using the mode of the posterior predictive distribution as a point estimate. This last estimate is also called a maximum a posteriori (MAP) estimate. It should be noted that under certain conditions the MAP estimate is equivalent to the MLE.

2.1. Example

All these concepts can be illustrated in the context of logistic regression. Suppose that we have n observations of a binary response variable $\boldsymbol{y} \in \{0,1\}^n$ and two continuous features $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)} \in \mathbb{R}^n$. The frequentist approach will be studied first.

Then, it will stated as loss function minimization and eventually as an inference problem in the Bayesian framework.

If someone wanted to build a classifier using a linear regression model, then they would run into a problem because a linear model of the form $\mu(\boldsymbol{x_i}) = \theta_0 + \theta_1 x_i^{(1)} + \theta_2 x_i^{(2)}$ is not bounded. In contrast, the response variable $\boldsymbol{y}$ from the data only takes the values 0 and 1, hence, it is a better idea to model the probability that a feature vector belongs to each class. This way, y_i is a random variable that follows a Bernoulli distribution with parameter p_i, where p_i is the probability of y_i being 1. The range of f as a linear model is $\mathbb{R}$, so it must be mapped to the $[0, 1]$ interval in which probabilities live, and it can be done with what is called a sigmoid function. The **logistic sigmoid function** is a widely used link function for this type of problems [4, p. 114]. It is defined as

$$\sigma(w) = \frac{e^w}{1 + e^w} = \frac{1}{1 + e^{-w}}. \tag{2.9}$$

It is easy to see that $\lim_{w \to -\infty} \sigma(w) = 0$ and $\lim_{w \to \infty} \sigma(w) = 1$.

Another possible sigmoid function is the **probit function**, defined as the inverse of the cumulative distribution function of a Gaussian random variable [15, p. 296]. That is, $\sigma(w) = \Phi^{-1}(x)$, where

$$\Phi(w) = \int_{-\infty}^{w} \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right) dx. \tag{2.10}$$

There are some differences in the theory and behavior of the logistic and the probit functions, but in practice they hardly make any substantial difference and the choice of one over the other may be a matter of taste or convenience [23, p. 118]. Henceforth, we will use the logistic function as the sigmoid function for binary classification problems.

It has been established that $\sigma(\cdot)$ maps from $\mathbb{R}$ to $[0, 1]$, but it is yet to be explained how to relate this to the original problem which is to build a linear classifier using the vectors $\boldsymbol{x}^{(1)}$ and $\boldsymbol{x}^{(2)}$. Using $\sigma(\cdot)$, one can define the probability that y_i belongs to class 1 given features $x_i^{(1)}$ and $x_i^{(2)}$ as

$$p(y_i = 1 | \boldsymbol{x_i}, \boldsymbol{\theta}) = \sigma(\theta_0 + \theta_1 x_i^{(1)} + \theta_2 x_i^{(2)}) = \frac{1}{1 + e^{-\left(\theta_0 + \theta_1 x_i^{(1)} + \theta_2 x_i^{(2)}\right)}}, \tag{2.11}$$

with $\boldsymbol{x_i} = \left[x_i^{(1)}, x_i^{(2)}\right]^T$. Thus

$$p\left(y_i = 0|\boldsymbol{x_i}, \boldsymbol{\theta}\right) = 1 - p\left(y_i = 1|\boldsymbol{x_i}\right) = \frac{1}{1 + e^{\theta_0 + \theta_1 x_i^{(1)} + \theta_2 x_i^{(2)}}}. \tag{2.12}$$

This means that y_i is fully described as a Bernoulli distribution with parameter $p_i = p\left(y_i = 1|\boldsymbol{x_i}, \boldsymbol{\theta}\right)$. When the response variable is Bernoulli distributed, the likelihood of n independent observations is

$$\prod_{i=1}^{n} p_i(\boldsymbol{\theta})^{y_i} \left(1 - p_i(\boldsymbol{\theta})\right)^{1-y_i}. \tag{2.13}$$

Then, in consequence, the log-likelihood of n independent observations is

$$L(\boldsymbol{\theta}) = \sum_{i=1}^{n} \left[y_i \log\left(p_i(\boldsymbol{\theta})\right) + (1 - y_i) \log\left(1 - p_i(\boldsymbol{\theta})\right)\right], \tag{2.14}$$

where $p_i(\boldsymbol{\theta}) = \sigma(\theta_0 + \theta_1 x_i^{(1)} + \theta_2 x_i^{(2)})$.

The maximum likelihood estimator of $\boldsymbol{\theta} = [\theta_0, \theta_1, \theta_2]^T$ is the vector $\hat{\boldsymbol{\theta}}$ that maximizes $L(\boldsymbol{\theta})$ in equation (2.14); that is $\hat{\boldsymbol{\theta}} = \arg\max L(\boldsymbol{\theta})$.

Another way to interpret the problem of binary classification, is to consider the learning problem from an optimization perspective in which the goal is to minimize an appropriate loss function. It has been shown above that maximizing equation (2.14) leads to the maximum likelihood estimate. However, if the sign is changed, the result is a loss function that can be minimized, and it provides the exact same solution as before.

The new objective function that needs to be minimized is

$$L(\boldsymbol{\theta}) = -\sum_{i=1}^{n} \left[y_i \log\left(p_i(\boldsymbol{\theta})\right) + (1 - y_i) \log\left(1 - p_i(\boldsymbol{\theta})\right)\right]. \tag{2.15}$$

This loss function is commonly known in the literature as the **binary entropy** loss.

Intuitively, this is an error function because if $y_i = 1$, then $1 - y_i = 0$, so the second term in the loss function vanishes and what is left remaining is $\log\left(p_i(\boldsymbol{\theta})\right)$.

Now, when $p_i(\boldsymbol{\theta})$ is big, that is, close to 1, then $\log(p_i(\boldsymbol{\theta})) \to 0$. However, if $p_i(\boldsymbol{\theta})$ is small, that is, close to 0, then $\log(p_i(\boldsymbol{\theta})) \to -\infty$, and so the overall loss function tends to infinity. This means that when the real value of y_i is 1 and a low probability is assigned to it, then the loss function is large; but if a high probability is assigned to it, then the loss function is small. The same reasoning works for when $y_i = 0$. So, when there is an incorrect classification with very high probability, the loss function takes a large value. Hence, minimizing the loss function leads to choosing values of $\boldsymbol{\theta}$ that yield a high probability to the cases when $y_i = 1$ and a low one when $y_i = 0$.

The logistic regression model can be reformulated from a Bayesian approach. As before, it will be assumed that y_i follows a Bernoulli distribution such that the probability of being 1 is $p_i(\boldsymbol{\theta}) = \sigma(\theta_0 + \theta_1 x_i^{(1)} + \theta_2 x_i^{(2)})$, where $\sigma(\cdot)$ is the logistic sigmoid function. In the Bayesian paradigm, the unknown parameter vector $\boldsymbol{\theta} = [\theta_0, \theta_1, \theta_2]^T$ must have a joint prior distribution. A good idea is to assume a Gaussian distribution for the unknown unobservable parameter $\boldsymbol{\theta}$. Furthermore, independence between the components of $\boldsymbol{\theta}$ can be assumed which in turn allows one to write the prior distribution as

$$\boldsymbol{\theta} \sim \mathcal{N}\left(\boldsymbol{\mu}, \sigma^2 \boldsymbol{I}\right), \tag{2.16}$$

where $\boldsymbol{\mu} \in \mathbb{R}^3$, $\sigma^2 \in \mathbb{R}^+$ and $\boldsymbol{I}$ is a 3×3 identity matrix. Assuming $\boldsymbol{\mu} = [\mu_0, \mu_1, \mu_2]^T$, then the prior distribution takes the following form

$$p(\boldsymbol{\theta}) = \prod_{k=0}^{2} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(\theta_k - \mu_k)^2}{2\sigma^2}}. \tag{2.17}$$

The likelihood can be written as

$$p(\boldsymbol{X}|\boldsymbol{\theta}) = \prod_{i=1}^{n} p_i(\boldsymbol{\theta})^{y_i} \left(1 - p_i(\boldsymbol{\theta})\right)^{1-y_i}, \tag{2.18}$$

which follows from the independence assumption in the data-generating mechanism. Moreover, Bayes' theorem in equation (2.6) allows one to write the posterior

distribution as

$$p\left(\boldsymbol{\theta}|\boldsymbol{y},\boldsymbol{X}\right)=\frac{\left[\prod_{i=1}^{n}p_i(\boldsymbol{\theta})^{y_i}\left(1-p_i(\boldsymbol{\theta})\right)^{1-y_i}\right]\left[\prod_{k=0}^{2}\frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{(\theta_k-\mu_k)^2}{2\sigma^2}}\right]}{\int\left[\prod_{i=1}^{n}p_i(\boldsymbol{\theta})^{y_i}\left(1-p_i(\boldsymbol{\theta})\right)^{1-y_i}\right]\left[\prod_{k=0}^{2}\frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{(\theta_k-\mu_k)^2}{2\sigma^2}}\right]d\boldsymbol{\theta}}. \tag{2.19}$$

It should be noted that the integral in the denominator is a constant with respect to $\boldsymbol{\theta}$, thus, the posterior distribution can be computed up to a normalizing constant as

$$p\left(\boldsymbol{\theta}|\boldsymbol{y},\boldsymbol{X}\right)\propto\left[\prod_{i=1}^{n}p_i(\boldsymbol{\theta})^{y_i}\left(1-p_i(\boldsymbol{\theta})\right)^{1-y_i}\right]\left[\prod_{k=0}^{2}\frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{(\theta_k-\mu_k)^2}{2\sigma^2}}\right]. \tag{2.20}$$

Unfortunately, for most practical problems the integral in the denominator cannot be solved analytically, therefore numerical methods are used. One class of widely used algorithms is Markov Chain Monte Carlo (MCMC), which includes Metropolis-Hastings and Hamiltonian Monte Carlo. The main goal of MCMC methods is to generate samples in an attempt to summarize and report posterior inferences on the unknown parameters. Another class of numerical methods is Variational Inference, which is discussed in Chapter 3.

Note how the maximum likelihood and the loss function minimization approaches only provide a point estimate of the parameters and of the predictions. On the other hand, the Bayesian approach provides a posterior distribution of the parameters and a posterior predictive distribution. As mentioned above, these distributions reflect the knowledge about the parameters, and are a compromise between the prior distribution and the data. A prior distribution that has a very low variance may give rise to a posterior distribution that does not deviate too much from the prior. Whereas a prior distribution with a very high variance will be driven more by the data, and hence the posterior distribution may deviate a lot from the prior distribution. The prior distribution should reflect the knowledge that the modeler has about the phenomenon being modeled.

2.2. Overfitting

In general, when training a model for prediction, it is not the main goal to minimize training error, but prediction error. It does not matter whether one chooses to follow the function minimization, frequentist or Bayesian paradigm. That is, one wants to minimize the error of any future observation $(\boldsymbol{x}^*, \boldsymbol{y}^*)$. This is because these models are trained to be used with data that has not been seen before, therefore, it is desirable to have a model that generalizes well to future observations. This means that one wants to minimize the expected prediction error, hence, it is common to divide the data set $(\boldsymbol{X}, \boldsymbol{y})$ in two: the training set and the test set. To do this, from the original data set, a random sample of observations is assigned to be part of the training set and the rest are part of the test set. The idea is that the test set should be used to measure the predictive performance of the model, and this also helps to diagnose an overfitted model, a concept which is discussed later in this chapter. The data is sometimes divided in three different sets. This third set is called the validation set, and it is used to measure the generalization performance when tuning hyperparameters, such as regularization parameter values, discussed further in this document.

The idea of separating the data in different sets comes from the **bias-variance trade-off**, a property of learning models. The idea of the bias-variance trade-off is that the expected prediction mean squared error of a learned model $\hat{f}$ can be decomposed in the sum of three quantities: the squared bias of $\hat{f}$, the variance of $\hat{f}$, and an irreducible error. The bias of the model refers to the error that comes from approximating a complicated phenomenon with a simpler model, the variance refers to how much the estimated model $\hat{f}$ would change with a different data set, and the irreducible error refers to the noise that comes from the problem itself which cannot be decreased [15, 27]. In general, a modeler wants to have models with low bias and low variance, so that the expected prediction mean squared error is low, but as the name implies, there is a trade-off. Flexible models tend to have low bias but high variance, and rigid models tend to have low variance but high bias.

When training highly flexible models, the separation of data in training and validation sets is more important. This is because very flexible models tend to have higher variance, and thus they may fit the training data very well, but may not generalize very well with data that has not been previously observed. This is due to the fact that part of the error is being captured by the model. This concept is known as **overfitting**, and is a very common problem in machine learning applications.

2.3. Regularization

One way to control overfitting is with **regularization**, in which a penalization term is added to the loss function to prevent the parameters from taking values that are too high. A very simple regularizer is the L_2 norm in which the sum of the squared parameters is added to the loss function.

For example, in linear regression, the usual loss function that is sought to be minimized is

$$\sum_{i=1}^{n} \left(y_i - \sum_{k=1}^{p} \theta_k x_k \right)^2, \tag{2.21}$$

but with the L_2 regularizer (also known as ridge regression), equation (2.21) is modified so that the loss function now is

$$\sum_{i=1}^{n} \left(y_i - \sum_{k=1}^{p} \theta_k x_k \right)^2 + \lambda \left(\sum_{k=2}^{p} \theta_k^2 \right), \tag{2.22}$$

where $\lambda > 0$ is called the regularization parameter, and controls the relative importance between the two main terms of the sum. The value of this parameter can have a big impact in the loss function: if it is too big, then it will penalize too much and most parameters will be near 0, but if it is too small, then there is little regularization effect and it is as if there were no penalization at all. In practice, it is common to choose several concrete values of λ (such as 0.001, 0.01 and 0.1) and select the final value using the validation set previously mentioned, so that the final chosen value of λ minimizes the validation set error.

Another type of regularization that is widely used in neural network literature, presented further in this work, is **dropout** [54]. Dropout consists in randomly dropping out, that is temporarily removing, units of the neural network in each feed-forward mini-batch pass, which results in a thinned network. Each unit is removed, along with all its input and output connections, with a certain probability p. The original idea of dropout is that by randomly dropping units, the modeler is essentially training different networks, so the end result is the combination of different architectures, which result in less overfitting. At test time, the prediction is usually just the average of the different thinned networks. These concepts will be clearer after Chapter 4 in which neural networks are discussed.

In this chapter, the concepts of machine learning were introduced, presenting both the frequentist and the Bayesian paradigms for prediction, as well as a brief introduction to regularization. In the following chapters, an introduction to Variational Inference (VI) will be given. After that, a brief overview of Artificial Neural Network models will be discussed. Afterwards, the Active Learning methodology will be presented in Chapter 5 and, finally, in Chapter 6 numerical experiments will be shown.

Chapter 3

Variational Inference

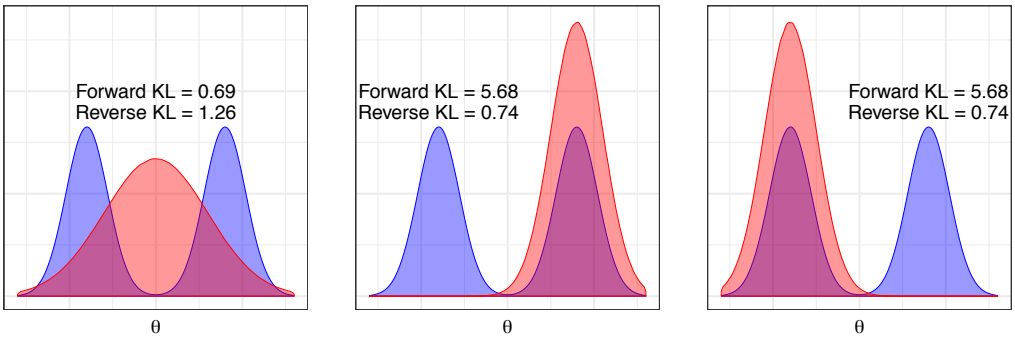
As mentioned in Chapter 2, when performing Bayesian inference, it is of interest to compute the posterior distribution $p(\boldsymbol{\theta}|\boldsymbol{y}, \boldsymbol{X})$, which is usually intractable, so one must resort to numerical approximations. In this chapter, a brief overview of variational inference (VI) is given, which is one of several numerical approximations, such as Markov Chain Monte Carlo (MCMC) or Integrated Nested Laplace Approximations (INLA). The idea of VI is to use optimization to approximate the target distribution $p(\boldsymbol{\theta}|\boldsymbol{y}, \boldsymbol{X})$ with some distribution $q(\boldsymbol{\theta})$ that is close to the posterior. The Kullback-Leibler (KL) divergence is used in VI as a measure of closeness of the proposed distribution, which is known as the variational approximation, to the true posterior [5]. The KL divergence is defined as

$$\mathbb{KL}(q||p) = \mathbb{E}_q\left[\log\left(\frac{q(\boldsymbol{\theta})}{p(\boldsymbol{\theta}|\boldsymbol{y}, \boldsymbol{X})}\right)\right] = \int_{-\infty}^{\infty} q(\boldsymbol{\theta})\left[\log\left(\frac{q(\boldsymbol{\theta})}{p(\boldsymbol{\theta}|\boldsymbol{y}, \boldsymbol{X})}\right)\right] d\boldsymbol{\theta}. \quad (3.1)$$

Note that the KL divergence is not symmetrical, i.e., $\mathbb{KL}(q||p) \neq \mathbb{KL}(p||q)$. That is the reason it is called a divergence and not a distance. The former is called the **reverse KL divergence** and the latter is called the **forward KL divergence**, and each one of them prioritizes different aspects of the approximation. Reverse KL is said to be "zero-forcing" because it forces q to be zero in some areas, effectively ignoring the value of p in those areas, even if p is bigger than zero.

On the contrary, forward KL is said to be "zero-avoiding" because q avoids areas where it happens simultaneously that q is zero and p is bigger than zero.

This behavior can be appreciated in figure 3.1, which shows forward and reverse KL divergence of a mixture of Gaussians, denoted by p, and three different unimodal Gaussian distributions, each denoted by q. The blue distribution in the figure is the mixture of Gaussians, and the red distributions are the unimodal Gaussians. In subfigure (a), the zero-avoiding behavior can be appreciated because the red distribution q has a mode where the blue distribution p has zero values and q avoids the areas where p is bigger than zero. It can also be appreciated that the reverse KL divergence is lower than the forward KL divergence. In subfigures (b) and (c), the zero-forcing behavior is evident because the algorithm is forcing q to be zero even if p is bigger than zero. Because of this, it has a lower forward KL divergence. These two subfigures also show the two local optima gotten from minimizing the reverse KL divergence, each being placed in the two modes of the p distribution. In fact, both forward and reverse KL divergence on subfigure (b) are equal to the ones on subfigure (c).



(a) $\mathbb{KL}(p||q) < \mathbb{KL}(q||p)$ **(b)** $\mathbb{KL}(p||q) > \mathbb{KL}(q||p)$ **(c)** $\mathbb{KL}(p||q) > \mathbb{KL}(q||p)$

Figure 3.1: *Comparison of forward and backward KL divergence of a mixture of Gaussians p (in blue), and three different unimodal Gaussian distributions, each denoted by q (in red). On the left, the forward KL divergence is smaller than the reverse KL divergence. On the center and on the right, the opposite happens.*

In VI, the goal is to minimize the reverse KL divergence $\mathbb{KL}(q||p)$ instead of

the forward KL divergence $\mathbb{KL}\left(p||q\right)$ because the latter requires averaging with respect to $p\left(\boldsymbol{\theta}|\boldsymbol{y},\boldsymbol{X}\right)$, which is what the modeler is trying to approximate in the first place. Methods to deal with forward KL divergence exist, such as expectation propagation, but they are not of concern in this work.

Although the main goal of VI methods is to minimize the reverse KL divergence defined in equation (3.1), in practice what is usually done is to maximize a related quantity called the ELBO (Evidence Lower Bound), defined as

$$\mathcal{L}(q) = \mathbb{E}_q\left[\log p(\boldsymbol{y},\boldsymbol{X},\boldsymbol{\theta})\right] - \mathbb{E}_q\left[\log q(\boldsymbol{\theta})\right]. \tag{3.2}$$

The relationship between the ELBO and KL is shown as follows. Starting from the KL divergence definition in (3.1), using the quotient rule of the logarithm, and using the fact that taking expectations is a linear operator, the KL divergence can be written as

$$\mathbb{KL}\left(q||p\right) = \mathbb{E}_q\left[\log\left(\frac{q(\boldsymbol{\theta})}{p\left(\boldsymbol{\theta}|\boldsymbol{y},\boldsymbol{X}\right)}\right)\right] = \mathbb{E}_q\left[\log q(\boldsymbol{\theta})\right] - \mathbb{E}_q\left[\log p\left(\boldsymbol{\theta}|\boldsymbol{y},\boldsymbol{X}\right)\right]. \tag{3.3}$$

Using the definition of conditional probability yields

$$\mathbb{KL}\left(q||p\right) = \mathbb{E}_q\left[\log q(\boldsymbol{\theta})\right] - \mathbb{E}_q\left[\log \frac{p(\boldsymbol{\theta},\boldsymbol{y}|\boldsymbol{X})}{p(\boldsymbol{y}|\boldsymbol{X})}\right]. \tag{3.4}$$

Using the definition of conditional probability a second time, one obtains

$$\mathbb{KL}\left(q||p\right) = \mathbb{E}_q\left[\log q(\boldsymbol{\theta})\right] - \mathbb{E}_q\left[\log \frac{p(\boldsymbol{\theta},\boldsymbol{y},\boldsymbol{X})}{p(\boldsymbol{y}|\boldsymbol{X})p(\boldsymbol{X})}\right]. \tag{3.5}$$

Using the quotient rule of the logarithm one more time results in

$$\mathbb{KL}\left(q||p\right) = \mathbb{E}_q\left[\log q(\boldsymbol{\theta})\right] - \mathbb{E}_q\left[\log p(\boldsymbol{\theta},\boldsymbol{y},\boldsymbol{X}) - \log p(\boldsymbol{y}|\boldsymbol{X}) - \log p(\boldsymbol{X})\right]. \tag{3.6}$$

But since $p(\boldsymbol{X})$ and $p(\boldsymbol{y}|\boldsymbol{X})$ do not depend on q, the expected value of each of these two densities is a constant under taking expectations under q, hence

$$\mathbb{KL}\left(q||p\right) = \mathbb{E}_q\left[\log q(\boldsymbol{\theta})\right] - \mathbb{E}_q\left[\log p(\boldsymbol{\theta},\boldsymbol{y},\boldsymbol{X})\right] + \log p(\boldsymbol{y}|\boldsymbol{X}) + \log p(\boldsymbol{X}). \tag{3.7}$$

Since the goal is to find the optimal $q(\cdot)$, then $\log p(\boldsymbol{X})$ and $\log p(\boldsymbol{y}|\boldsymbol{X})$ are just constants in the optimization process, hence they can be ignored for optimization purposes, and hence, one just needs to minimize the following equation

$$\mathbb{E}_q\left[\log q(\boldsymbol{\theta})\right] - \mathbb{E}_q\left[\log p(\boldsymbol{\theta}, \boldsymbol{y}, \boldsymbol{X})\right], \tag{3.8}$$

which is the negative of the ELBO, defined in equation (3.2). Therefore, minimizing the KL divergence is equivalent to maximizing the ELBO.

Let $\boldsymbol{\lambda}$ be the parameter vector of the variational distribution $q(\boldsymbol{\theta})$. The objective is to approximate $p(\boldsymbol{\theta}|\boldsymbol{y}, \boldsymbol{X})$ by finding the values of $\boldsymbol{\lambda}$ that maximize equation (3.2). For this, several techniques could be used, such as coordinate ascent and gradient ascent. In order to perform gradient ascent, it is necessary to compute the gradient of the objective function with respect to the variational parameters, that is, $\nabla_{\boldsymbol{\lambda}}\mathcal{L}(q, \boldsymbol{\lambda})$. This gradient is computed by

$$\nabla_{\boldsymbol{\lambda}}\mathcal{L}(q, \boldsymbol{\lambda}) = \mathbb{E}_q\left[\left(\nabla_{\boldsymbol{\lambda}} \log q(\boldsymbol{\theta}|\boldsymbol{\lambda})\right)\left(\log p(\boldsymbol{y}, \boldsymbol{X}, \boldsymbol{\theta}) - \log q(\boldsymbol{\theta}|\boldsymbol{\lambda})\right)\right]. \tag{3.9}$$

In general, the gradient cannot be computed analytically because it may not be possible to write down an explicit formula for the expectation. It may be possible to take the expectation for some models, but for most cases, some approximation must be made. A good approach is to approximate the expected value with Monte Carlo simulation. Let $\{\boldsymbol{z}_1, ..., \boldsymbol{z}_S\}$ be samples taken from $q(\boldsymbol{\theta}|\boldsymbol{\lambda})$, then one can approximate the expected value with an arithmetic mean as such,

$$\begin{aligned}\nabla_{\boldsymbol{\lambda}}\mathcal{L}(q, \boldsymbol{\lambda}) &= \mathbb{E}_q\left[\left(\nabla_{\boldsymbol{\lambda}} \log q(\boldsymbol{\theta}|\boldsymbol{\lambda})\right)\left(\log p(\boldsymbol{y}, \boldsymbol{X}, \boldsymbol{\theta}) - \log q(\boldsymbol{\theta}|\boldsymbol{\lambda})\right)\right] \\ &\approx \frac{1}{S}\sum_{k=1}^{S}\left(\nabla_{\boldsymbol{\lambda}} \log q(\boldsymbol{z}_k|\boldsymbol{\lambda})\right)\left(\log p(\boldsymbol{y}, \boldsymbol{X}, \boldsymbol{z}_k) - \log q(\boldsymbol{z}_k|\boldsymbol{\lambda})\right).\end{aligned} \tag{3.10}$$

This approach gives noisy but unbiased estimates of the expected value. For more details about this see [31], [46] and [47].

A family that is often used to approximate p is the **mean-field variational family**, where each parameter is independent from on another, such that

$$q(\boldsymbol{\theta}) = \prod_i q(\theta_i|\lambda_i). \tag{3.11}$$

Because of how it is defined, the mean-field variational family can capture any marginal distribution of the parameters. However, it cannot recover explicit correlation between the parameters since it assumes that they are independent from one another. In consequence of this, the marginal representation of each parameter may underestimate the variance of the target distribution [5].

The following example will help illustrate the logic behind the Monte Carlo approximation for the variational gradient under the mean field variational family. In this example, the goal is to approximate the posterior distribution of a two-class logistic regression.[1] Assume a data matrix $\boldsymbol{X} \in \mathbb{R}^{n \times p}$ and a response vector $\boldsymbol{y}$ with values 1 or 0. Each observation y_i is modeled as a Bernoulli distribution such that

$$y_i | \boldsymbol{x_i}, \boldsymbol{\theta} \sim \text{Bern}(\sigma(\boldsymbol{\theta}^T \boldsymbol{x_i})) \tag{3.12}$$

where $\sigma(\cdot)$ is the logistic sigmoid function. The prior distribution for $\boldsymbol{\theta}$ is a Gaussian $\boldsymbol{\theta} \sim \mathcal{N}\left(\mathbf{0}, \sigma_0^2 \boldsymbol{I}_p\right)$ where $\boldsymbol{I}_p$ is the $p \times p$ identity matrix and $\sigma_0^2 \in \mathbb{R}^+$.

In this case, the mean-field variational distribution is

$$q(\boldsymbol{\theta}|\boldsymbol{\lambda}) = \prod_{j=1}^{p} \mathcal{N}\left(\theta_j | \mu_j, \sigma_j^2\right) \tag{3.13}$$

with $\boldsymbol{\lambda} = \left[\mu_1, \ldots, \mu_p, \sigma_1^2, \ldots, \sigma_p^2\right]^T$ and where $\mathcal{N}\left(\theta_j | \mu_j, \sigma_j^2\right)$ denotes the value of a Gaussian density function with mean μ_j and variance σ_j^2 that is evaluated in θ_j.

According to equation (3.9), to compute the gradient of the ELBO, one must be able to compute $\nabla_{\boldsymbol{\lambda}} \log q(\boldsymbol{\theta}|\boldsymbol{\lambda})$. Hence, it is necessary to have the partial derivative of $\log q(\boldsymbol{\theta}|\boldsymbol{\lambda})$ with respect to each μ_j and σ_j^2, with $j \in \{1, \ldots, p\}$.

[1] The idea of this example comes from Keyon Vafa, posted in `http://keyonvafa.com/logistic-regression-bbvi/`.

The process to compute ∇_{μ_j} goes as follows,

$$\begin{aligned}
\nabla_{\mu_j} \log q(\boldsymbol{\theta}|\boldsymbol{\lambda}) &= \nabla_{\mu_j} \log \left(\prod_{k=1}^{p} \mathcal{N}\left(\theta_k|\mu_k, \sigma_k^2\right) \right) \\
&= \nabla_{\mu_j} \sum_{k=1}^{p} \log \left(\mathcal{N}\left(\theta_k|\mu_k, \sigma_k^2\right) \right) \\
&= \nabla_{\mu_j} \log \left(\mathcal{N}\left(\theta_j|\mu_j, \sigma_j^2\right) \right) \\
&= \nabla_{\mu_j} \log \left(\frac{1}{\sqrt{2\pi\sigma_j^2}} \exp\left(-\frac{(\theta_j - \mu_j)^2}{2\sigma_j^2} \right) \right) \\
&= \frac{\theta_j - \mu_j}{\sigma_j^2}.
\end{aligned} \tag{3.14}$$

The process for σ_j^2 is similar, although it is easier to work with the logarithm because of the positivity restriction and because it leads to simpler algebra. Let $\alpha_j = \log \sigma_j^2$, so that instead of computing $\nabla_{\sigma_j^2}$, ∇_{α_j} will be computed.

The process goes as follows,

$$\begin{aligned}
\nabla_{\alpha_j} \log q(\boldsymbol{\theta}|\boldsymbol{\lambda}) &= \nabla_{\alpha_j} \log \left(\prod_{k=1}^{p} \mathcal{N}\left(\theta_k|\mu_k, \sigma_k^2\right) \right) \\
&= \nabla_{\alpha_j} \sum_{k=1}^{p} \log \left(\mathcal{N}\left(\theta_k|\mu_k, \sigma_k^2\right) \right) \\
&= \nabla_{\alpha_j} \log \left(\mathcal{N}\left(\theta_j|\mu_j, \sigma_j^2\right) \right) \\
&= \nabla_{\alpha_j} \log \left(\frac{1}{\sqrt{2\pi\sigma_j^2}} \exp\left(-\frac{(\theta_j - \mu_j)^2}{2\sigma_j^2} \right) \right) \\
&= \nabla_{\alpha_j} \log \left(\frac{1}{\sqrt{2\pi \mathrm{e}^{\alpha_j}}} \exp\left(-\frac{(\theta_j - \mu_j)^2}{2\mathrm{e}^{\alpha_j}} \right) \right) \\
&= -\nabla_{\alpha_j} \log \left(\sqrt{2\pi \mathrm{e}^{\alpha_j}} \right) - \frac{(\theta_j - \mu_j)^2}{2} \nabla_{\alpha_j} \mathrm{e}^{-\alpha_j} \\
&= -\frac{1}{2} + \frac{(\theta_j - \mu_j)^2}{2} \mathrm{e}^{-\alpha_j} \\
&= -\frac{1}{2} + \frac{(\theta_j - \mu_j)^2}{2\sigma_j^2}.
\end{aligned} \tag{3.15}$$

To compute the gradient in equation (3.9), it is necessary to also compute the log-likelihood of the data $p(\boldsymbol{y}, \boldsymbol{X}, \boldsymbol{\theta})$. Using the chain rule of probability, this can be written as $\log p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\theta}) + \log p(\boldsymbol{X}|\boldsymbol{\theta}) + \log p(\boldsymbol{\theta})$.

Note that in order to compute the gradient, $\log p(\boldsymbol{X}|\boldsymbol{\theta})$ is not needed, because in this case, the model is a discriminative model and not a generative one, hence $\boldsymbol{\theta}$ does not affect the distribution of the inputs $\boldsymbol{X}$. Mathematically, this can be seen in the following way

$$\begin{aligned}
\nabla_{\boldsymbol{\lambda}}\mathcal{L} &= \mathbb{E}_q\left[(\nabla_{\boldsymbol{\lambda}} q(\boldsymbol{\theta}|\boldsymbol{\lambda}))\left(\log p(\boldsymbol{y}, \boldsymbol{X}, \boldsymbol{\theta}) - \log q(\boldsymbol{\theta}|\boldsymbol{\lambda})\right)\right] \\
&= \mathbb{E}_q\left[(\nabla_{\boldsymbol{\lambda}} q(\boldsymbol{\theta}|\boldsymbol{\lambda}))\left(\log p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\theta}) + \log p(\boldsymbol{X}|\boldsymbol{\theta}) + \log p(\boldsymbol{\theta}) - \log q(\boldsymbol{\theta}|\boldsymbol{\lambda})\right)\right] \\
&= \mathbb{E}_q\left[(\nabla_{\boldsymbol{\lambda}} q(\boldsymbol{\theta}|\boldsymbol{\lambda}))\left(\log p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\theta}) + \log p(\boldsymbol{\theta}) - \log q(\boldsymbol{\theta}|\boldsymbol{\lambda})\right)\right] + \log p(\boldsymbol{X}|\boldsymbol{\theta})\mathbb{E}_q\left[\nabla_{\boldsymbol{\lambda}} q(\boldsymbol{\theta}|\boldsymbol{\lambda})\right] \\
&= \mathbb{E}_q\left[(\nabla_{\boldsymbol{\lambda}} q(\boldsymbol{\theta}|\boldsymbol{\lambda}))\left(\log p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\theta}) + \log p(\boldsymbol{\theta}) - \log q(\boldsymbol{\theta}|\boldsymbol{\lambda})\right)\right] + c\mathbb{E}_q\left[\nabla_{\boldsymbol{\lambda}} q(\boldsymbol{\theta}|\boldsymbol{\lambda})\right].
\end{aligned}$$

In the last line, $\log p(\boldsymbol{X}|\boldsymbol{\theta})$ has been reduced to just a constant c.

Furthermore, it can be proven that $\mathbb{E}_q\left[\nabla_{\boldsymbol{\lambda}} q(\boldsymbol{\theta}|\boldsymbol{\lambda})\right] = 0$ because of the definition of expectation and the dominated convergence theorem as was done in [47],

$$\int \nabla_{\boldsymbol{\lambda}} q(\boldsymbol{\theta}|\boldsymbol{\lambda}) d\boldsymbol{\theta} = \nabla_{\boldsymbol{\lambda}} \int q(\boldsymbol{\theta}|\boldsymbol{\lambda}) d\boldsymbol{\theta}. \tag{3.16}$$

And finally,

$$\nabla_{\boldsymbol{\lambda}} \int q(\boldsymbol{\theta}|\boldsymbol{\lambda}) d\boldsymbol{\theta} = 0. \tag{3.17}$$

Therefore, the derivative of the ELBO can be written as

$$\nabla_{\boldsymbol{\lambda}}\mathcal{L} = \mathbb{E}_q\left[(\nabla_{\boldsymbol{\lambda}} q(\boldsymbol{\theta}|\boldsymbol{\lambda}))\left(\log p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\theta}) + \log p(\boldsymbol{\theta}) - \log q(\boldsymbol{\theta}|\boldsymbol{\lambda})\right)\right]. \tag{3.18}$$

Since $\boldsymbol{y}$ is a Bernoulli random vector, then

$$\begin{aligned}
\log p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\theta}) &= \log\left(\prod_{i=1}^{n} \sigma(\boldsymbol{\theta}^T \boldsymbol{x_i})^{y_i}(1 - \sigma(\boldsymbol{\theta}^T \boldsymbol{x_i}))^{1-y_i}\right) \\
&= \sum_{i=1}^{n}\left[y_i \log\left(\sigma(\boldsymbol{\theta}^T \boldsymbol{x_i}\right) + (1 - y_i)(1 - \sigma(\boldsymbol{\theta}^T \boldsymbol{x_i}))\right].
\end{aligned} \tag{3.19}$$

And since $\boldsymbol{\theta}$ was assumed to have Gaussian prior, then

$$\log p(\boldsymbol{\theta}) = \log \prod_{j=1}^{p} \phi(\theta_j) = \sum_{j=1}^{p} \log \phi(\theta_j) = \frac{1}{\sqrt{2\pi}} e^{\left(-\frac{\theta_j^2}{2}\right)}. \tag{3.20}$$

With these derivations, a gradient ascent approach can be taken in order to optimize the ELBO. This is the same as gradient descent presented in Appendix A, except that instead of taking the negative of the gradient to move in the direction of maximum descent, the gradient is taken as it is so that the algorithm moves in the direction of maximum ascent. Like in gradient descent, the modeler starts with a random initial $\boldsymbol{\lambda}$ vector, and this vector is updated in each iteration using the gradient of the loss function. However, in this case, the gradient must be approximated with Monte Carlo simulation, so in each step t the algorithm takes S samples $\boldsymbol{z}_k$, with $\boldsymbol{z}_k \sim q(\boldsymbol{\theta}|\boldsymbol{\lambda})$ for $k \in \{1, \cdots, S\}$. This way, the approximate gradient is computed as

$$\nabla_{\boldsymbol{\lambda}} \mathcal{L} \approx \Delta_{\boldsymbol{\lambda}} \mathcal{L} = \frac{1}{S} \sum_{k=1}^{S} \left(\nabla_{\boldsymbol{\lambda}} \log q(\boldsymbol{z}_k|\boldsymbol{\lambda})\right) \left(\log p(\boldsymbol{y}, \boldsymbol{X}, \boldsymbol{z}_k) - \log q(\boldsymbol{z}_k|\boldsymbol{\lambda})\right), \quad (3.21)$$

and then the $\boldsymbol{\lambda}$ parameter is updated as $\boldsymbol{\lambda}_{t+1} = \boldsymbol{\lambda}_t + \eta_t \Delta_{\boldsymbol{\lambda}_t} \mathcal{L}$ until a certain stopping criterion is met, with each η_t chosen beforehand. The choice of η_t depends on the modeler, and can be chosen to be the same for all iterations, or it can change with each iteration like in the AdaGrad method [12].

An implementation of this algorithm was made in the R programming language with a simulated data set of $n = 500$ data points and $p = 1$ feature. The response vector was created with the logistic sigmoid function as $y_i = \sigma(\theta_0 + \theta_1 \boldsymbol{x_i})$, with $\theta_0 = -5$ and $\theta_1 = 5$.

The initial value for $\boldsymbol{\lambda} = [\mu_1, \mu_2, \alpha_1, \alpha_2]^T$ was chosen to be zero, and in each iteration, $S = 50$ Monte Carlo samples are taken to compute the approximate gradient. The step size $\boldsymbol{\eta}_t$ is chosen using the AdaGrad method [12] in which $\boldsymbol{\eta}_t = (\text{diag}(\boldsymbol{G}_t))^{\frac{1}{2}}$, where $\text{diag}(\boldsymbol{G}_t)$ denotes the diagonal of the matrix $\boldsymbol{G}_t = (\Delta_{\boldsymbol{\lambda}_t})(\Delta_{\boldsymbol{\lambda}_t})^T$. In this case $\boldsymbol{\eta}_t$ is a vector, hence the product $\boldsymbol{\eta}_t \Delta_{\boldsymbol{\lambda}_t}$ refers to an element-wise multiplication. The stopping criterion was chosen to be $\left\|\boldsymbol{\mu}_{t+1} - \boldsymbol{\mu}_t\right\| < 0.005$, where $\boldsymbol{\mu}_t = [\mu_1^t, \mu_2^t]^T$.

Figure 3.2 shows the results of the implementation, and the values estimated by the `glm` package in R are also shown for comparison. The values estimated by the `glm` package are taken as Gold standard. The top-left image shows the value of the objective function, i.e., the ELBO, in each iteration, slowly increasing in

zigzag because of the noise of the estimations. The top-right image shows the real values of $\boldsymbol{\theta}$ (-5 and 5) with gray dashed lines, the values of $\boldsymbol{\theta}$ estimated by the `glm` package with colored dotted lines, and the approximated values of the variational distribution $q(\boldsymbol{\theta}|\boldsymbol{\lambda})$. The bottom-left image shows the values of μ_1 and μ_2 in each iteration, with horizontal colored dotted lines showing the estimated values by the `glm` package; it can be seen how they slowly approach their real values. The bottom-right image shows the values of μ_1 and μ_2 in each iteration in the plane, where the noise of the gradient estimation is seen in the wiggliness of the line; the red dot shows once again the value estimated by the `glm` package. There are ways to have better and less noisy estimates, but they are beyond the scope of this work. For more details about better approximations of the gradient, see [31] and [47].

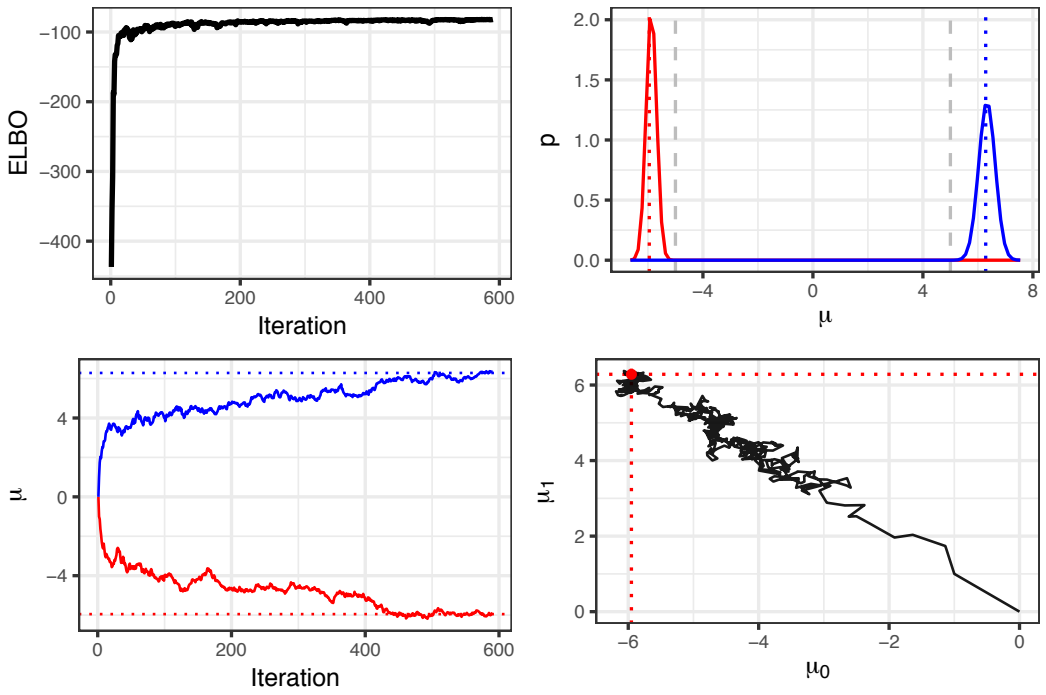


Figure 3.2: *Example of mean-field variational inference for Bayesian logistic regression.*

The posterior predictive distribution of y^* for a new vector of features $\boldsymbol{x}^*$ can be approximated using the variational approximation. In particular, the modeler

could take T samples from the variational posterior distribution as such: for $j \in \{1, \ldots, T\}$,

1. Sample a vector $\boldsymbol{\theta}_j \sim q(\boldsymbol{\theta}|\boldsymbol{\lambda})$
2. Use that value to sample $y_j^* \sim p(y^*|\boldsymbol{\theta}_j, \boldsymbol{x}^*)$

Then $\left\{y_j^*\right\}_{j=1}^T$ is a set of T independent samples from the posterior predictive distribution $p\left(y^*|\boldsymbol{y}, \boldsymbol{X}, \boldsymbol{x}^*\right)$.

In this chapter, Variational Inference has been discussed as a way to approximate the posterior predictive distribution. This was posed as an optimization problem in which the Kullback-Leibler divergence is minimized, and it was shown to be equivalent to maximizing the ELBO. In addition, a numerical approximation to the gradient using Monte Carlo simulation was introduced, thus allowing the modeler to perform the optimization process. Finally, an example of VI on a logistic regression problem was presented. In the following chapter, a brief overview of Artificial Neural Network models will be presented to motivate the methodology presented in Chapter 5 and illustrated with numerical experiments in Chapter 6.

Chapter 4

Artificial Neural Networks

In Chapter 2 it was mentioned that in supervised learning there is a response variable $\boldsymbol{y} \in \mathbb{R}^n$ and a data matrix $\boldsymbol{X} \in \mathbb{R}^{n \times p}$. It is assumed that there exists a relationship between $\boldsymbol{y}$ and $\boldsymbol{X}$ such that $\boldsymbol{y} = f(\boldsymbol{X}) + \boldsymbol{\varepsilon}$, with f being a fixed but unknown function. Since f is unknown, it is estimated with a function $\hat{f}$ that is chosen from a parameterized family of functions $\mathcal{F}_{\boldsymbol{\theta}}$. Artificial neural networks (ANNs) belong to one of the families from which $\mathcal{F}_{\boldsymbol{\theta}}$ can be chosen. In this chapter, two types of ANNs will be studied: feed-forward neural networks and convolutional neural networks (CNNs).

4.1. Artificial Neural Networks

The most basic, and perhaps the best known, type of ANN is the feed-forward neural network, also widely called multilayer perceptron (MLP). This concept will be first introduced in the frequentist framework and then they will be explained in the Bayesian framework.

To illustrate the multilayer perceptron, consider a simple example: a researcher wants to model a continuous variable $\boldsymbol{y}$ from a single feature $\boldsymbol{x}$. A very simple model would be a linear regression, in which each observation i is mod-

eled as $y_i = \theta_0 + \theta_1 x_i$, with θ_0 and θ_1. ANNs go further and take non-linear transformations of this linear predictor with some function $\sigma(\cdot)$, such that $y_i = \theta_0^{[1]} + \sum_{j=1}^{m} \theta_j^{[1]} \sigma\left(\theta_0^{[0]} + \theta_j^{[0]} x_i\right)$, where m is a tuning hyperparameter previously chosen. Function $\sigma(\cdot)$ is called the **activation function**, and is very often chosen to be the logistic sigmoid function $\sigma(x) = (1 + e^{-x})^{-1}$, the hyperbolic tangent $\sigma(x) = \tanh(x)$, or the Rectified Linear Unit (ReLU) $\sigma(x) = \max(0, x)$.

Figure 4.1 shows these relationships in a graphic way for a model with $m = 3$ and where $a_{k,i}$ is defined as $a_{k,i} = \sigma\left(\theta_0^{[0]} + \theta_k^{[0]} x_i\right)$ for $k \in \{1, 2, 3\}$. For simplicity, this image does not explicitly show the parameters $\theta_0^{[0]}$ and $\theta_0^{[1]}$, which are called the **bias** parameters.

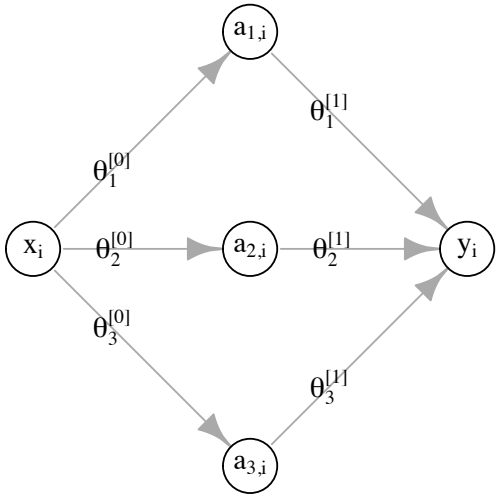


Figure 4.1: *Diagram of a multilayer perceptron with $m = 3$.*

If the response variable $\boldsymbol{y}$ happened to be a binary variable, then one last transformation needs to be applied to map the model to the $[0, 1]$ interval and model $\boldsymbol{y}$ as a probability, just as in logistic regression. In the case of a binary response variable, y_i could be written as

$$y_i = \phi\left(\theta_0^{[1]} + \sum_{j=1}^{m} \theta_j^{[1]} \sigma\left(\theta_0^{[0]} + \theta_j^{[0]} x_i\right)\right) = \phi\left(\theta_0^{[1]} + \sum_{j=1}^{m} \theta_j^{[1]} a_{j,i}\right), \tag{4.1}$$

with $\phi(\cdot)$ the logistic sigmoid function to map from $\mathbb{R}$ to $[0, 1]$. The choice of function $\phi(\cdot)$ depends on the nature of the response variable. In the case of a continuous response variable, $\phi(\cdot)$ is the identity function; in a binary classification problem, it could be the logistic sigmoid function; and in the case of a classification problem with more than two categories, a softmax function could be used.

A more complex model can be created by taking linear combinations of non-linear functions of the previous result. Let

$$a_{k,i}^{[1]} = \sigma^{[1]}\left(\theta_0^{[1]} + \theta_k^{[0]} x_i\right) \tag{4.2}$$

for $k \in \{1, \ldots, m\}$ and

$$a_{k,i}^{[2]} = \sigma^{[2]}\left(\theta_{0,k}^{[1]} + \sum_{j=1}^{m} \theta_{j,k}^{[1]} a_{j,1}^{[0]}\right) \tag{4.3}$$

for $k \in \{1, \ldots, r\}$.

Then

$$y_i = \theta_0^{[2]} + \sum_{j=1}^{r} \theta_j^{[2]} a_{j,i}^{[2]} \tag{4.4}$$

where r, like m, is chosen beforehand. The values m and r are called the number of nodes or units of each layer. This is a deeper model than the last one because it has more layers. Notice that there are two different activation functions $\sigma^{[1]}(\cdot)$ and $\sigma^{[2]}(\cdot)$. Each layer can have a different function. The first one could be a sigmoid function and the second one a hyperbolic tangent, or vice versa, or they could both be the same function.

Figure 4.2 shows this graphically with $m = 3$ and $r = 2$. Layers corresponding to $a_{k,i}^{[1]}$ for $k \in \{1, \ldots, m\}$ and $a_{k,i}^{[2]}$ for $k \in \{1, \ldots, r\}$ are called the **hidden layers**. Figure 4.1 shows a multilayer perceptron with one hidden layer, and figure 4.2 shows a multilayer perceptron with two hidden layers.

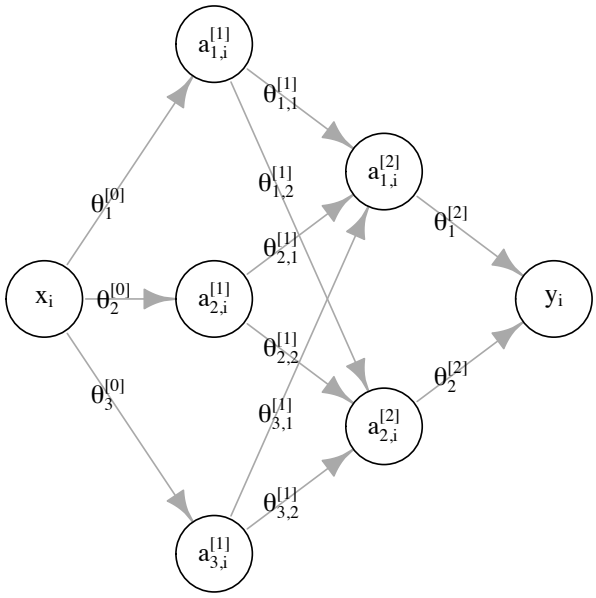


Figure 4.2: *Diagram of a multilayer perceptron with two hidden layers,* $m = 3$ *and* $r = 2$.

After these two introductory examples of MLPs, they can be defined in a more formal and general fashion. As before, $\boldsymbol{y}$ denotes the response variable, and y_i denotes the i-th element of $\boldsymbol{y}$. Once again, $\boldsymbol{X} \in \mathbb{R}^{n \times p}$ denotes the data matrix; $\boldsymbol{x}_i \in \mathbb{R}^p$ denotes the i-th row in $\boldsymbol{X}$, representing the values of the features for the i-th element in the data; $\boldsymbol{x}^{(k)} \in \mathbb{R}^n$ denotes the k-th column in $\boldsymbol{X}$; and $x_i^{(k)}$ denotes the k-th column-wise element and the i-th row-wise element.

A multilayer perceptron can have any integer number of layers, and this number will be denoted by L. The number of nodes of each layer l will be denoted by $n^{[l]}$. And the activation function for each layer l will be denoted by $\sigma^{[l]}(\cdot)$, for $l \in \{1, \ldots, L\}$. The activation function for the last layer will be denoted by $\phi(\cdot)$, as in the binary classification example. The parameter that connects the j-th node from the $(l-1)$-th layer with the k-th node in the l-th layer is denoted by $\theta^{[l]}_{j,k}$ and, as before, $a^{[l]}_{k,i}$ denotes the result of the activation function corresponding to the l-th layer and the i-th observation, for $k \in \left\{1, \ldots, n^{[l]}\right\}$; where $\theta^{[l]}_{0,k}$ is the bias term

of the k-th node in the l-th layer. This notation can be summarized as follows,

$$a_{k,i}^{[l]} = \sigma^{[l]} \left(\theta_{0,k}^{[l-1]} + \sum_{j=1}^{n^{[l-1]}} \theta_{j,k}^{[l-1]} a_{j,i}^{[l-1]} \right) \tag{4.5}$$

for $k \in \left\{1, \ldots, n^{[l]}\right\}$, $j \in \left\{1, \ldots, n^{[l-1]}\right\}$, $l \in \{1, \ldots, L\}$ and $i \in \{1, \ldots, n\}$.

To be consistent, $a_{k,i}^{[0]}$ is defined as $x_i^{[k]}$, hence, $\boldsymbol{a}_k^{[0]} = \boldsymbol{x}^{(k)} \in \mathbb{R}^n$; and $a_{0,i}^{[0]} = 1$ for all $i \in \{1, \ldots, n\}$, so $\boldsymbol{a}_0^{[0]}$ is a vector of ones of dimension n and $n^{[0]}$ is the number of features, i.e., $n^{[0]} = p$.

In general, a feed-forward neural network or multilayer perceptron with L hidden layers is such that

$$y_i = \phi \left(\theta_{0,1}^{[L]} + \sum_{j=1}^{n^{[L]}} \theta_{j,1}^{[L]} a_{k,i}^{[L]} \right), \tag{4.6}$$

where equation (4.5) must be held. As a way to summarize the whole notation, a multilayer perceptron with parameters $\boldsymbol{\Theta}$ and data matrix $\boldsymbol{X}$ will be denoted as $\text{MLP}\left(\boldsymbol{X}, \boldsymbol{\Theta}\right)$, where parameter $\boldsymbol{\Theta}$ is a general way to refer to all parameters $\theta_{j,k}^{[l]}$ for $k \in \left\{1, \ldots, n^{[l]}\right\}$, $j \in \left\{1, \ldots, n^{[l-1]}\right\}$ and $l \in \{1, \ldots, L\}$.

Figure 4.3 shows an example of an architecture with 2 hidden layers ($L = 2$), 4 features ($p = n^{[0]} = 4$), 3 nodes in the first hidden layer ($n^{[1]} = 3$) and 4 nodes in the second hidden layer ($n^{[2]} = 4$).

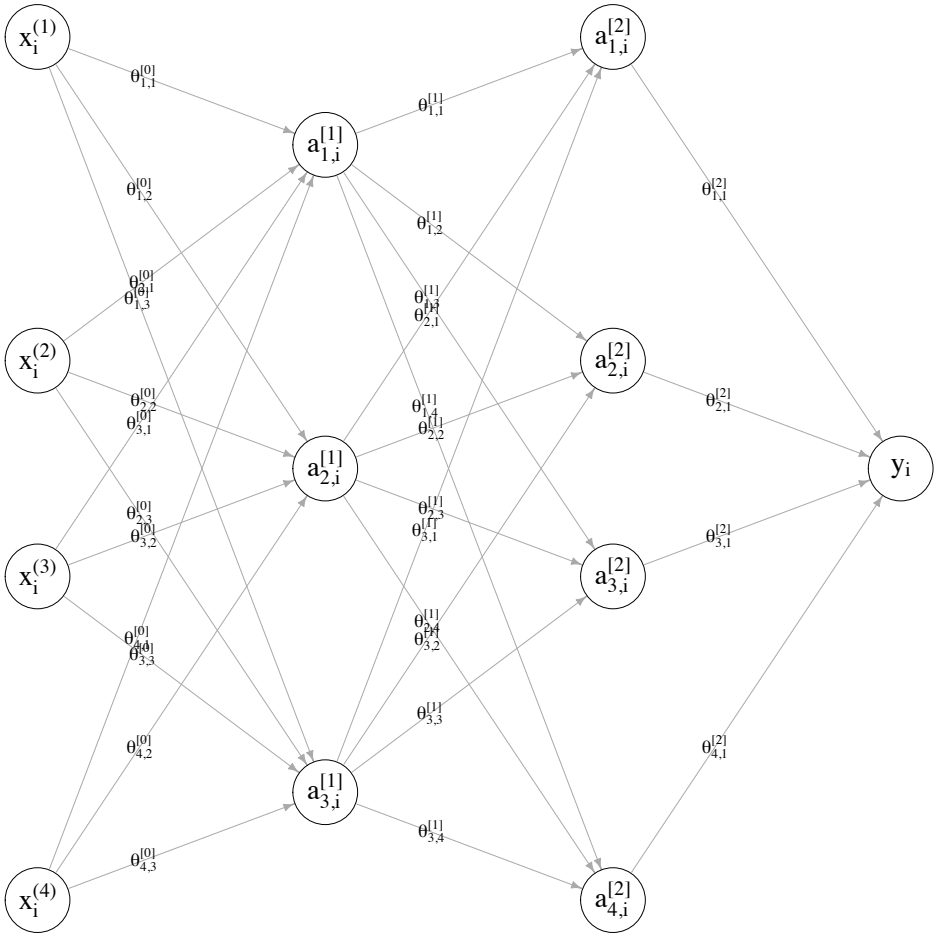


Figure 4.3: *Diagram of a multilayer perceptron with two hidden layers using the general notation with $L = 2$, $p = n^{[0]} = 4$, $n^{[1]} = 3$ and $n^{[2]} = 4$.*

In the ANN literature, the most common approach to estimate the network parameters is the loss function minimization approach. In most cases, this approach results in the same estimate as in the maximum likelihood approach because of the way in which loss functions are defined. Hyperparameters, such as the number of layers and the number of neurons, are usually chosen by a cross-validation scheme, or with a validation set.

As mentioned in Chapter 2, when the response variable is continuous, the

quadratic loss is often used, and it is equivalent to the maximization of a Gaussian likelihood function. In the case of a binary classification problem, the binary entropy loss defined in (2.15), is usually used, and it is equivalent to maximizing the likelihood of independent Bernoulli response variables.

In the case of a multiple classification problem, the cross-entropy loss is extended to accommodate for more classes using the same maximum likelihood idea. In a K class classification problem, the loss is defined as

$$L(\boldsymbol{\Theta}) = -\sum_{i=1}^{n}\left[\sum_{j=1}^{K} y_{i,j}\log\left(\hat{y}_{i,j}\right)\right] \tag{4.7}$$

where $y_{i,j} = 1$ if the i-th training point belongs to the j-th class and $\hat{y}_{i,j}$ is the model's predicted probability of the i-th training point belonging to the j-th class.

In the Bayesian approach, the idea is the same as was explained in Chapter 2. Each of the parameters is assigned a prior distribution and the response variable $\boldsymbol{y}$ is assumed to be generated by a probabilistic model with unknown parameters. The latter are assumed to be random variables as well, to be able to represent the uncertainty on their values. For example, in the continuous case, a Gaussian conditional distribution to $\boldsymbol{y}$ can be assigned, such that

$$\boldsymbol{y}|\boldsymbol{\Theta}, \boldsymbol{X} \sim \mathcal{N}\left(\text{MLP}(\boldsymbol{X}, \boldsymbol{\Theta}), \sigma^2\right) \tag{4.8}$$

where $\text{MLP}(\boldsymbol{X}, \boldsymbol{\Theta})$ denotes the architecture of a multilayer perceptron with parameters $\boldsymbol{\Theta}$ and data matrix $\boldsymbol{X}$. As was mentioned before, since the $\boldsymbol{\Theta}$ parameters can be any real value, it is also common to assign a Gaussian distribution to them, so that the prior distribution is such that

$$\theta_{i,k}^{[l]} \sim \mathcal{N}\left(\mu_{i,k}^{[l]}, \sigma_{i,k}^{[l]}\right). \tag{4.9}$$

Bayes' theorem is used to update the knowledge about the parameters given the data in the following way

$$p\left(\boldsymbol{\Theta}|\boldsymbol{X}, \boldsymbol{y}\right) = \frac{p\left(\boldsymbol{y}|\boldsymbol{\Theta}, \boldsymbol{X}\right) p\left(\boldsymbol{\Theta}|\boldsymbol{X}\right)}{p\left(\boldsymbol{y}|\boldsymbol{X}\right)}. \tag{4.10}$$

In the example above, $p(\boldsymbol{y}|\boldsymbol{\Theta}, \boldsymbol{X})$ and $p(\boldsymbol{\Theta}|\boldsymbol{X})$ are the joint density functions of Gaussian random variables with their corresponding parameter values. To make predictions, the posterior predictive distribution defined in equation (2.7) is used.

As mentioned in Chapter 3, complex Bayesian models such as Bayesian ANN can be intractable or too slow for typical numerical approaches such as MCMC. The reason is that ANNs tend to have a high-dimensional parameter vector, for which off-the-shelf MCMC approaches struggle to reach stationarity of the Markov Chain. In this sense, dropout, the stochastic regularization technique mentioned in Chapter 2, can be seen as a variational approximation to a Bayesian neural network and to a deep Gaussian process [19, 20, 18, 21]. Particularly, applying dropout before each weight layer in a neural network with arbitrary depth is mathematically equivalent to a variational approximation that minimizes the KL divergence to the posterior distribution of a Bayesian neural network. This approximation includes convolutional neural networks, presented in the next section.

4.2. Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are neural networks with a particular architecture and are widely used in computer vision problems, such as image classification or object detection in videos. They were introduced by LeCun in 1989 as a way to constrain the number of parameters needed to perform automatic image classification [33]. The main idea is that in images, pixels close to each other tend to be similar, whereas pixels far away from each other tend to be different. This notion of proximity is used by CNN's architecture, and leverages three concepts: sparse interactions, parameter sharing and equivariant representations [3, p. 335].

- Sparse interactions: In MLPs, every unit is connected to all other units in the next layer, but in CNNs, only few units are connected to other few of units. This means fewer parameters and, hence, less memory usage and faster computation.
- Parameter sharing: Different units share the same set of parameters. The

same convolution is used many times in the image and, thus, the number of parameters is further reduced.

- Equivariant representations: The architecture of CNNs provides equivariance in translations, meaning that if the input is changed, the output is changed in the same fashion. For example: in an image, if an object is moved in the input, its representation will be moved in the same way in the output.

4.2.1. Convolution operation

The base of CNNs is the convolution operation. This is a general operation, but for this work, it will be described assuming that the input is a digital image. This image is represented as an $m \times n$ matrix $\boldsymbol{I}$, where $I_{i,j}$ represents the pixel in the i-th row and j-th column as such

$$\boldsymbol{I} = \begin{bmatrix} I_{1,1} & I_{1,2} & I_{1,3} & \dots & I_{1,n} \\ I_{2,1} & I_{2,2} & I_{2,3} & \dots & I_{2,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ I_{m,1} & I_{m,2} & I_{m,3} & \dots & I_{m,n} \end{bmatrix}. \tag{4.11}$$

The convolution operation uses another matrix called a **filter** or a **kernel**, with dimensions $p \times r$, where $p < m$ and $r < n$. This means that the kernel matrix must be of a lower dimension than the image $\boldsymbol{I}$. The filter matrix is denoted by $\boldsymbol{K}$ as

$$\boldsymbol{K} = \begin{bmatrix} \theta_{1,1} & \theta_{1,2} & \theta_{1,3} & \dots & \theta_{1,q} \\ \theta_{2,1} & \theta_{2,2} & \theta_{2,3} & \dots & \theta_{2,q} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \theta_{p,1} & \theta_{p,2} & \theta_{p,3} & \dots & \theta_{p,q} \end{bmatrix}. \tag{4.12}$$

The convolution operation between $\boldsymbol{I}$ and $\boldsymbol{K}$ is denoted as $\boldsymbol{I} * \boldsymbol{K}$, and its result is another matrix $\boldsymbol{S}$. Each element of $\boldsymbol{S}$ is defined as

$$S_{i,j} = (\boldsymbol{I} * \boldsymbol{K})_{i,j} = \sum_k \sum_l I_{i+k-1,j+l-1} K_{k,l}. \tag{4.13}$$

An example of this is shown in figure 4.4, in which $\boldsymbol{I}$ is a 7×7 image matrix and the kernel $\boldsymbol{K}$ is a 3×3 matrix. The result is a 5×5 matrix in which each element is computed using equation (4.13). For instance, the element on the first row and fourth column of $\boldsymbol{S}$, that is, $S_{1,4}$ is computed as

$$\begin{aligned} S_{1,4} &= \sum_{k=1}^{3}\sum_{l=1}^{3} I_{1+k-1,4+l-1}K_{k,l} \\ &= I_{1,4}K_{1,1} + I_{1,5}K_{1,2} + I_{1,6}K_{1,3} \\ &+ I_{2,4}K_{2,1} + I_{2,5}K_{2,2} + I_{2,6}K_{2,3} \\ &+ I_{3,4}K_{3,1} + I_{3,5}K_{3,2} + I_{3,6}K_{3,3} \\ &= (1 \times 1) + (0 \times 0) + (0 \times 1) \\ &+ (1 \times 0) + (1 \times 1) + (0 \times 0) \\ &+ (1 \times 1) + (1 \times 0) + (1 \times 1) \\ &= 4. \end{aligned} \tag{4.14}$$

The rest of the elements are computed in a similar way. It should be noted that the resulting matrix is smaller than the original image matrix.

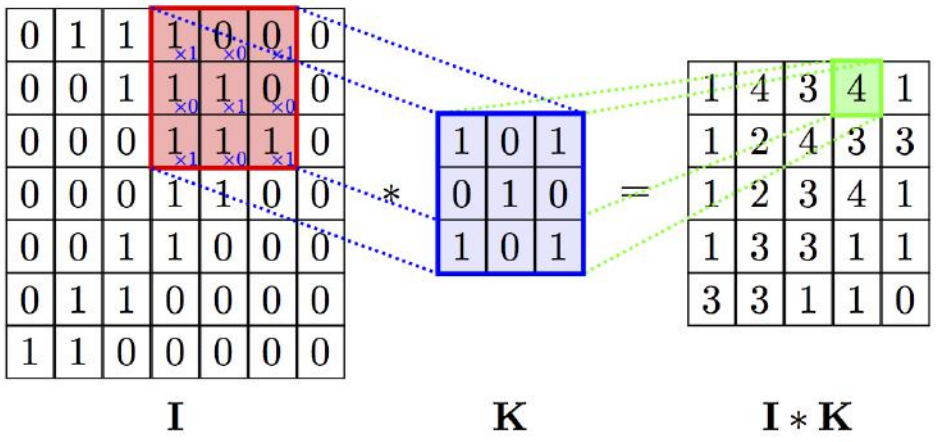


Figure 4.4: *Example of a convolution operation. Source:* `https://github.com/PetarV-/TikZ`

The convolution kernel in the example only has 9 parameters which are used by the whole image matrix, and these parameters are usually estimated with data.

An image classification problem could be solved by turning the input image into a long vector, but then this would be fed to a fully connected layer, which would mean that a large number of parameters would have to be fitted. This is one of the reasons convolutions are used: instead of estimating a large number of parameters, the convolution operation needs a small number of parameters (9 in the case of the example) to be estimated. Additionally, the convolution operation takes into account the topology of the image input, which has a very local structure [34].

4.2.2. Pooling

When using CNNs, a pooling layer is also usually added after each convolutional layer. This pooling layer summarizes adjacent pixels, and it is used because it helps to achieve invariance and reduce the image of the output so that there are fewer parameters in the next layers [3]. A very commonly used pooling function is **max pooling**, which returns the maximum value of a neighborhood of pixels. An example of max pooling is shown in figure 4.5 for a 4×4 matrix, resulting in a 2×2 matrix after the function is applied.

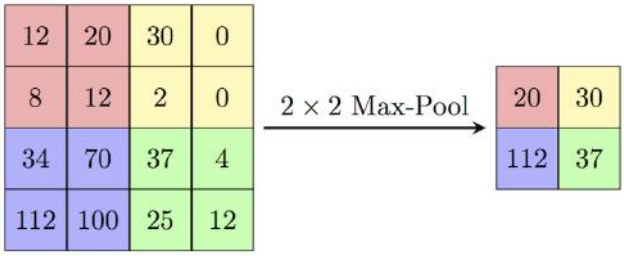


Figure 4.5: *Example of max pooling function. Source:* `https://computerscience wiki.org/index.php/File:MaxpoolSample2.png`

4.2.3. Architecture example

To illustrate how all these concepts are put together, an example of an architecture that is very commonly used in image classification problems is discussed. This architecture is called the **LeNet architecture**, and it was introduced by LeCun

et al. in 1998 for a digit classification problem [34]. The architecture assumes a grayscale input image of 32×32 pixels, which is then fed to 6 convolutional filters of size 5×5, each followed by an activation function and a 2×2 max pooling layer, then 16 convolutional filters of size 5×5, each with their corresponding activation functions and then the 2×2 max pooling layer, followed by two fully connected layers of sizes 120 and 84, and finally a softmax transformation to map to the 10 digit classification problem probabilities.

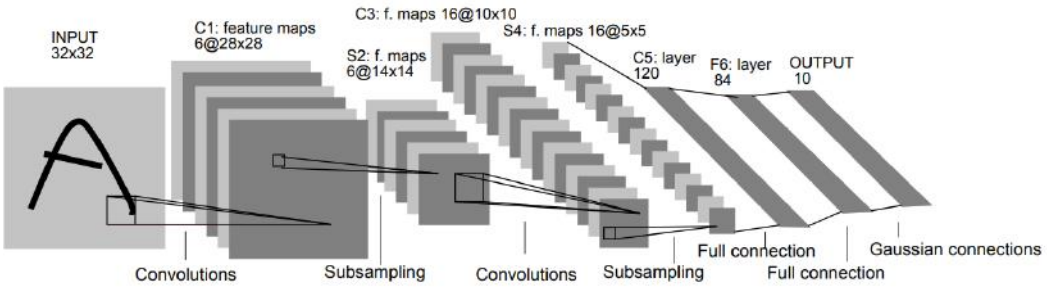


Figure 4.6: *LeNet architecture (picture taken from [34]). The feature maps correspond to the convolutional filters and the subsampling refers to max pooling.*

This chapter described the basic idea of ANNs and CNNs. Chapter 5 will discuss the Active Learning methodology and Chapter 6 will present numerical experiments in which the ideas of the previous chapters are used together and compared with each other.

Chapter 5

Active Learning

An active learning problem is one in which the learner is able to select its own training data so that the performance will be better with less training data. This is attractive because many times, labeled data is expensive or hard to acquire, hence the desire to use the available labeled data as efficiently as possible.

For example, a modeler may have access to a lot of unlabeled data points and few labeled data points. These labeled data points may be expensive to label by human annotators, and they also may be used in a supervised learning problem, but it is desirable to somehow take advantage of the unlabeled data. An active learning task would be to train a model $\hat{f}$ with an initial set of labeled data $\mathcal{D}$. Afterwards, the model (the learner) could choose new data points $\boldsymbol{x}^*$ from a pool of unlabeled data $\mathcal{U}$, to ask to an *oracle* (e.g., a human annotator) what the corresponding output y^* is, and finally add the pair $(\boldsymbol{x}^*, y^*)$ to the training data. This process would be repeated until certain criteria are met, with the training set increasing in size with each iteration.

Figure 5.1 shows this cycle in a diagram. The main goal of active learning is to select which $\boldsymbol{x}^*$ to incorporate to the training data [8]. In the end, it is expected that this procedure leads to a better predictive performance than randomly selecting new observations to add to the training data.

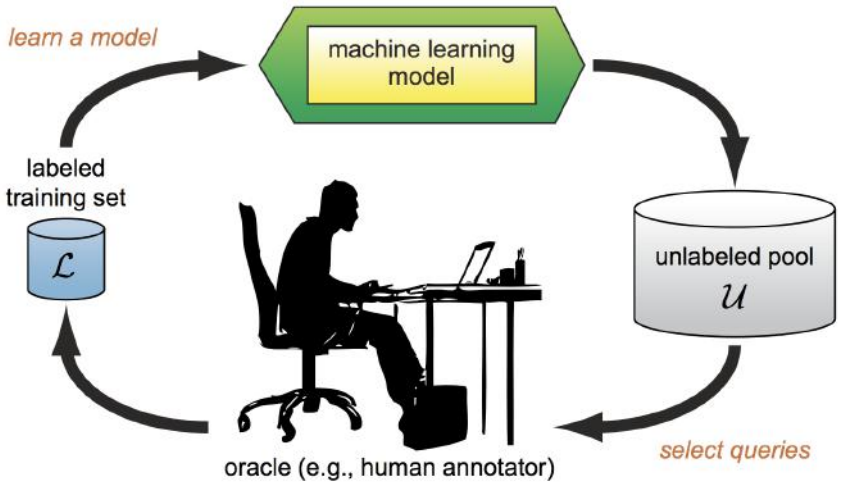


Figure 5.1: *Active learning cycle. A model is trained from the labeled data $\mathcal{D}$, data points are taken from the unlabeled pool of data $\mathcal{U}$, labeled by an oracle and then added to the training set of labeled data $\mathcal{D}$ in order to retrain the model and continue the cycle. Image taken from [51].*

The type of active learning described in the previous example, in which there is a large collection of unlabeled data $\mathcal{U}$ and a small set of labeled data $\mathcal{D}$, is called *pool-based sampling*. Pool-based sampling will be the main focus of this work. There are other types of active learning but, since they are not of interest to this work, they will not be discussed. However, [44] and [51] provide a good overview of these techniques and reference to more sources of information.

As mentioned before, in pool-based sampling, a model is trained with labeled training data from $\mathcal{D}$, and then, new data points are chosen to be labeled from the unlabeled data pool $\mathcal{U}$. The new data points are chosen using an **acquisition function** $a(x, \hat{f})$ that proposes candidates given a specific criterion. For example, the model's prediction uncertainty can be used to select new unlabeled data points. Usually, one would like to explore the feature space where uncertainty is higher as this corresponds to providing the learner the most informative feature. Also, some other entropy-based methods can be used. The new observation $\boldsymbol{x}^*$ is chosen so that it maximizes the acquisition function of all the observations in the pool set. The role of the acquisition function is to evaluate the gain of information of each unlabeled data point.

The acquisition functions that will be used and compared in this work are four, three of which use the posterior predictive distribution. Particularly, the posterior predictive probability of an observation $\boldsymbol{x}^*$ having a label y^* belonging to a class c, denoted as $p(y^* = c|\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{x}^*)$. Because of the definition of these acquisition functions, they only work in classification problems. The four acquisition functions are the following:

1. Predictive entropy:

   $\mathbb{H}[y^*|\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{x}^*] = -\sum_c p(y^* = c|\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{x}^*) \log p(y^* = c|\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{x}^*)$.

2. Variation ratios: $1 - \max_y p(y^*|\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{x}^*)$.

3. Bayesian Active Learning by Disagreement (BALD):

   $\mathbb{H}[y^*|\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{x}^*] - \mathbb{E}_{p(\boldsymbol{\theta}|\boldsymbol{X}, \boldsymbol{y})}[\mathbb{H}[y^*|\boldsymbol{x}^*, \boldsymbol{\theta}]]$.

4. Random: Choosing an observation uniformly random from the pool of unlabeled data $\mathcal{U}$.

Note that predictive entropy and variation ratios can be used with a frequentist classifier because instead of using the posterior predictive distribution, one can use the predicted probability of each class. The BALD acquisition function, however, can only be used with a Bayesian classifier because in the case of a frequentist classifier, the result of this function is always zero. This is easy to see given the definition of the acquisition function

$$\mathbb{H}[y^*|\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{x}^*] - \mathbb{E}_{p(\boldsymbol{\theta}|\boldsymbol{X}, \boldsymbol{y})}[\mathbb{H}[y^*|\boldsymbol{x}^*, \boldsymbol{\theta}]], \tag{5.1}$$

where $\mathbb{H}[y^*|\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{x}^*]$ is the predictive entropy, previously defined as

$$\mathbb{H}[y^*|\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{x}^*] = -\sum_c p(y^* = c|\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{x}^*) \log p(y^* = c|\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{x}^*). \tag{5.2}$$

In the Bayesian case, one has a positive number of samples from the posterior predictive distribution, so the second part of equation (5.1), i.e., the expected value $\mathbb{E}_{p(\boldsymbol{\theta}|\boldsymbol{X}, \boldsymbol{y})}[\mathbb{H}[y^*|\boldsymbol{x}^*, \boldsymbol{\theta}]]$ is approximated by averaging the predictive entropy

of each predictive sample. This way, the BALD acquisition function is computed by taking the difference of this quantity and the first part of equation (5.1). In the frequentist case, there is only one point estimate, so that difference is zero, hence the BALD acquisition function is always zero.

If the data points in the pool of unlabeled data points $\mathcal{U}$ are in random order, then using BALD as an acquisition function with a frequentist classifier is equivalent to a random acquisition function because all data points in $\mathcal{U}$ have the same acquisition function value.

A simple example of the active learning cycle was implemented in the R programming language. This figure summarizes the results of a two class classification problem using a frequentist logistic regression model and the variation ratios acquisition function. Similarly to what was done in Chapter 2, a data set was generated with a data matrix $\boldsymbol{X} \in \mathbb{R}^{n \times 10}$ with $n = 1000$ such that $\boldsymbol{x_i^{(k)}} \sim N(0, 1)$ for each $i \in \{1, ..., n\}$ and $k \in \{1, ..., 10\}$. Then an auxiliary vector was computed, $p_i = \frac{1}{1+\exp\left(-\sum_{k=1}^{10} \theta_k x_i^{(k)}\right)}$, with each θ_k sampled from a continuous uniform distribution from -5 to 5. Finally, the response variable $\boldsymbol{y}$ was built simulating Bernoulli random variables such that $y_i \sim \text{Bern}(p_i)$.

For the example, an initial random training set of 20 observations $\mathcal{D}$ was generated, with 10 examples of each class. Of the 980 points that were not used, 300 were used as a validation set, and the remaining 680 points were used as the unlabeled pool set $\mathcal{U}$. Then, 180 acquisition iterations were made, in each iteration training a new logistic regression model, making predictions on the pool set $\mathcal{U}$, and using these predictions to compute the frequentist variation ratios and to choose the data point with the highest value so that it would be added to the training set set $\mathcal{D}$. In order to remove sampling noise, this process was repeated 50 times, each time choosing different initial random training, validation and pool sets. The same was done using a random acquisition function to compare with the active learning setting.

Figure 5.2 shows the mean accuracy on the validation set in each acquisition iteration, with the light colors showing the 25-th and 75-th percentiles of the

accuracies in each iteration. It is clear that using active learning leads to a better performance of the model with less data compared to a random sample from $\mathcal{U}$.

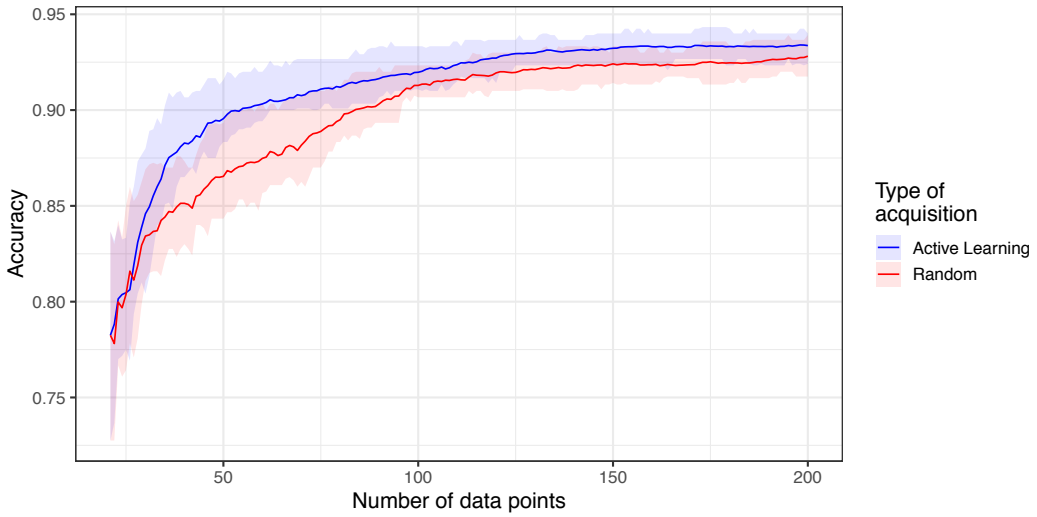


Figure 5.2: *Accuracies of a logistic regression model using a variation ratios acquisition function (blue) and a random acquisition function (red).*

To see what kind of points the acquisition function chooses, another example was implemented. This example is very similar to the last one, but in this case the parameter vector $\boldsymbol{\theta}$ is a two-dimensional vector such that $\boldsymbol{\theta} = [1, 1]^T$. Again, 1000 points were generated and an initial random training set of 10 training examples (5 of each class) was randomly chosen from these 1000 points. This was followed by 70 active learning iterations, using both a variation ratios acquisition function and a random acquisition function.

Figure 5.3 shows the results of this process. On the left, the generated data are shown, with the thin black line representing the real decision boundary of the generated data. The center image highlights the data points selected by the random acquisition function. Again, the thin black line is the real decision boundary, whilst the bold dashed black line is the decision boundary of the trained model. The right image highlights the data points selected by the variation ratios acquisition function, where the black lines represent the same as in the center image. It

can be seen that when using the variation ratios acquisition function, the selected data points lie close to the decision boundary, whilst using the random acquisition function, points from all the domain are chosen. This is because points lying close to the decision boundary give more information than points in the outer parts of the quadrant. In the end, the variation ratios acquisition function helped choose the points that lead to a better approximation to the real decision boundary.

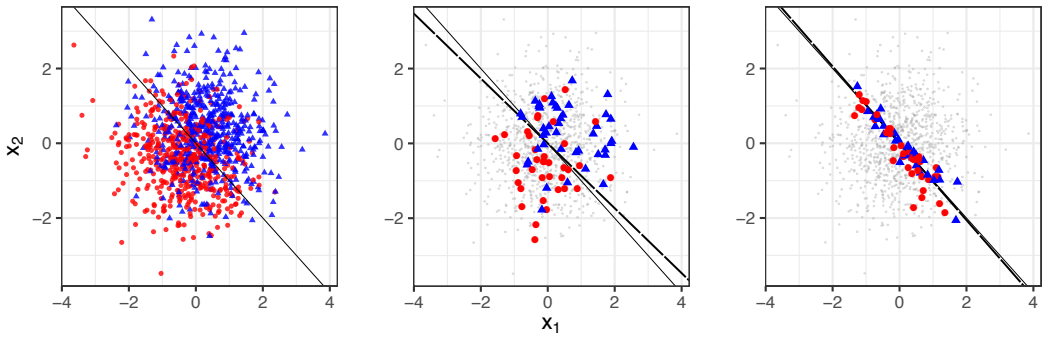


Figure 5.3: *Comparison of points chosen by a random acquisition function and a variation ratios acquisition function.*

This chapter provided a brief introduction to pool-based active learning and four acquisition functions. Chapter 6 will show different implementations of active learning problems using Convolutional Neural Networks and different acquisition functions. It will also compare the predictive performance of these acquisition functions in three different data sets.

Chapter 6

Experimental results

In the previous chapters, a brief overview of machine learning, Bayesian inference, variational inference, ANNs, and active learning was presented. In this chapter, three experiments are presented in the context of Bayesian active learning. These consist of well known machine learning benchmarks, such as the MNIST dataset [34], the CIFAR-10 dataset [30], and a dataset of cat and dog pictures [13].

In all the experiments, a deep convolutional network was trained to classify the images in each data set, as was done in [22]. The experiment setup was very similar to the one in Chapter 5, having each model trained with an initial small random set of images. Then, an acquisition function is used to select new images from a pool set of unlabeled data $\mathcal{U}$. This allows to increase the training set $\mathcal{D}$ and finally train a new model with this new bigger data set of labeled images. Some of the images with highest uncertainty according to the acquisition functions are shown in Appendix B.

As discussed in Chapter 5, a testbed of acquisition functions is considered. That is, in all experiments the Bayesian predictive entropy, frequentist predictive entropy, Bayesian variation ratios, frequentist variation ratios, BALD, and random acquisition functions are used to choose points to extend the training set. The Bayesian CNNs were trained using the dropout variational approximation

mentioned in Chapter 4.

All models were trained using Keras [7] with Tensorflow [1] as a backend. Most of the code is writen in R with some Python scripts called from R using the `reticulate` package [2].

6.1. MNIST dataset

The MNIST dataset is a collection of 70,000 labeled images of 32×32 pixels representing handwritten digits in black and white. The dataset has a pre-selected test set consisting of 10,000 images. Figure 6.1 shows 100 randomly chosen digits from the training set.

Figure 6.1: *Example of 225 randomly chosen digits from the MNIST train set.*

The MNIST dataset was used in the paper of Gal, Islam, and Ghahramani to

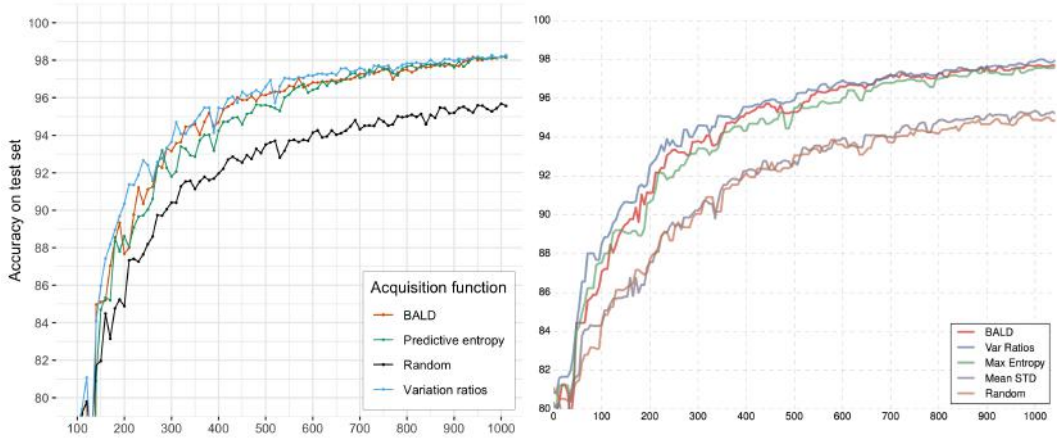
compare the performance of approximate Bayesian CNNs with frequentist CNNs in the context of Active Learning. This experiment was replicated in this work by using the same architecture that was explained in [22].

The experiment consists in starting with a random but balanced sample of 20 images from the MNIST dataset. Next, a CNN is trained with these images, to subsequently take a random sample of 5000 images from the pool set to save computation time. Afterwards, the probability that each of the 5000 images belongs to each of the 10 classes is computed. Finally, the 10 images that maximize the acquisition function are acquired and added to the training set. This process was repeated in 100 acquisition steps for each acquisition function. This whole process is repeated 3 times with different initial images to have the results averaged and reduce the noise inherent in the stochasticity of the process.

The acquisition functions used in the original paper are the Bayesian predictive entropy, frequentist predictive entropy, Bayesian variation ratios, frequentist variation ratios, BALD, random, and an additional one called "Mean STD" which was not implemented in this work because of its bad performance in the original paper.

The architecture of the CNN used was convolution-relu-convolution-relu-max pooling-dropout-dense-relu-dropout-dense-softmax, with 32 convolution kernels, 4×4 kernel size, 2×2 pooling, dense layer with 128 units, and dropout probabilities 0.25 and 0.5. Dropout was only applied to densely connected layers. The CNNs were trained for 50 epochs. For the Bayesian acquisition functions, 100 posterior predictive distribution points were sampled by using the dropout trick mentioned in Chapter 4.

The results of the Bayesian and random acquisition functions can be seen in figure 6.2, where the results of our implementation are shown on the left and the original article's results are shown on the right. Our results are very similar with the article's results: apparently, using the variation ratios results in a slightly better performance than BALD or predictive entropy, and more importantly, using the Bayesian acquisition functions results in a much better performance than using a random acquisition function.

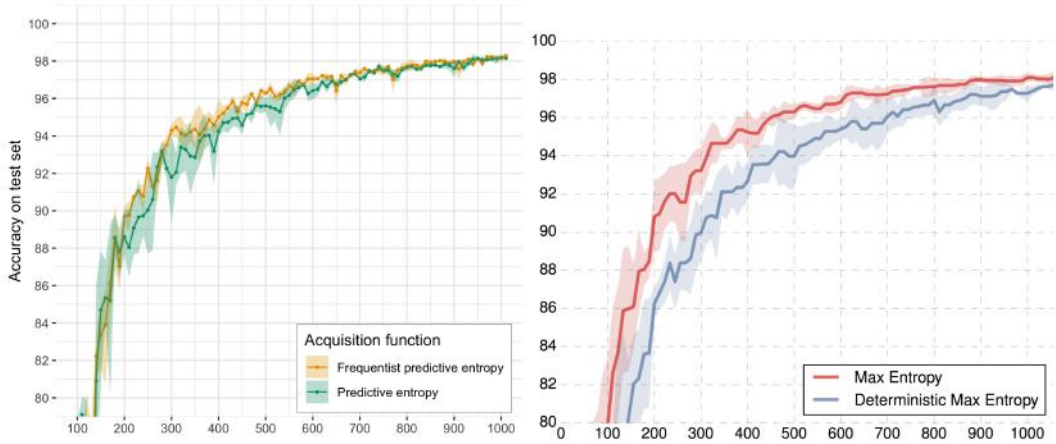


(a) *Our results.* **(b)** *Paper's results.*

Figure 6.2: *Accuracy of models in each acquisition step. The left picture shows our implementation and the right picture shows Gal, Islam, and Ghahramani's implementation. In the latter case, Max Entropy refers to predictive entropy. The x axis is the number of images.*

Our implementation achieved the goal of outperforming a random acquisition function. However, the results differ when one compares the Bayesian and the frequentist approach. The article's authors claim that the use of an approximate Bayesian approach in the acquisition process of Active Learning leads to better accuracy with fewer images, but in our implementation this does not happen. This can be seen with more detail in figures in 6.3 and 6.4 that show our results on the left and the paper's results on the right. In the original paper, the frequentist acquisition functions lead to a worse performance than their Bayesian counterparts, while in our implementation there is no evidence of distinction between the two.

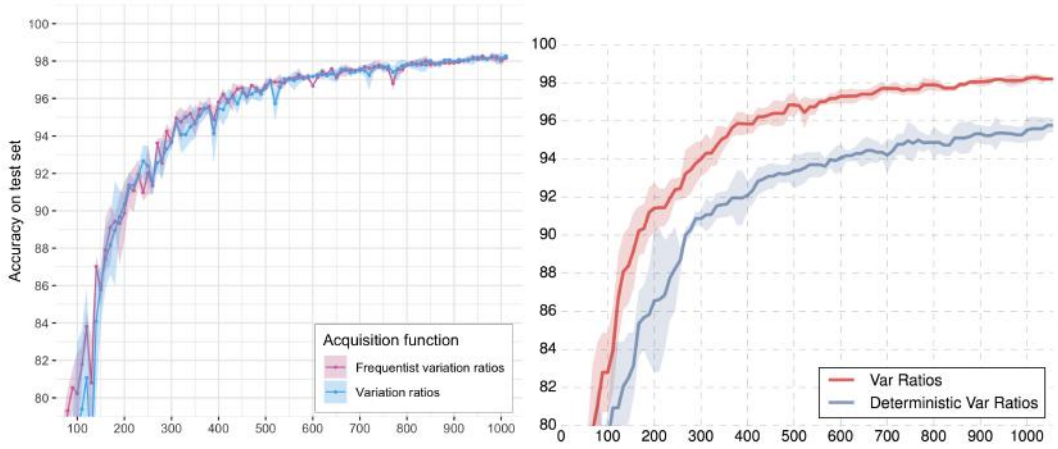
For example, with predictive entropy in figure 6.3, the results seem to be backward and the frequentist acquisition function seems to have a slightly better performance that the Bayesian one. Another difference is that the frequentist acquisition function in the original article achieves a 90% accuracy with around 300 images, while in our implementation this accuracy is first achieved with around 200 images.



(a) *Our results.* **(b)** *Paper's results.*

Figure 6.3: *Accuracy of approximate Bayesian and frequentist models in each acquisition step using predictive entropy as acquisition function. The left picture shows our implementation and the right picture shows Gal, Islam, and Ghahramani's implementation. In the latter case, Max Entropy refers to predictive entropy.*

Figure 6.4 shows the results of the variation ratios acquisition functions. While the roles of the frequentist and Bayesian acquisition functions are not reversed as in the previous case, there is apparently no distinction between the two acquisition functions. Moreover, in the original article, a 90% accuracy is first achieved using the frequentist acquisition function with a little below 300 images, while in our implementation this accuracy is first achieved with around 200 images. Another difference is that with 1000 images and using the frequentist acquisition function, the authors' accuracy is close to 96%, while in our implementation it is close to 98%.



(a) *Our results.* **(b)** *Paper's results.*

Figure 6.4: *Accuracy of approximate Bayesian and frequentist models in each acquisition step using variation ratios as acquisition function. The left picture shows our implementation and the right picture shows Gal, Islam, and Ghahramani's implementation.*

6.2. Cats and dogs dataset

The cats and dogs dataset was first used for a CAPTCHA developed by Microsoft Research called Asirra [13]. It is a collection of 25,000 pictures of size 64×64, of which half are cats and half are dogs. Figure 6.5 shows 30 randomly chosen cats and 30 randomly chosen dogs from the training set.

Figure 6.5: *Example of 60 randomly chosen cats and 30 randomly chosen dogs from the dataset.*

The process of this experiment is very similar to the one followed in the MNIST dataset. First, a balanced set of 100 images was randomly chosen and used to train a first model. Then a random sample of 2000 images was chosen from the pool set, from which the predicted probabilities of belonging to each class were computed. Subsequently, the 50 images with the highest acquisition function value were chosen and added to the training set. The final number of acquisition steps was 50 and, once more, the whole process was repeated 3 times with different initial images.

The architecture used was convolution-relu-convolution-relu-max pooling-dropout-convolution-relu-max pooling-dropout-dense-relu-dropout-dense-softmax with 32 convolution kernels, 3×3 kernel size, 2×2 pooling, dense layer with 64 units, and dropout probabilities 0.25, 0.25 and 0.5. The CNNs were trained for 200 epochs. Once more, for the Bayesian acquisition functions 100 posterior predictive distribution points were sampled by using the dropout trick.

The results of the Bayesian and random acquisition functions can be seen in

figure 6.6. In this case, it is difficult to say that using any of the Bayesian acquisition functions results in a better performance than using a random acquisition function. In the last steps it is clear that the variation ratios acquisition function results in a better accuracy, but the difference is not as abrupt as it was in the MNIST dataset.

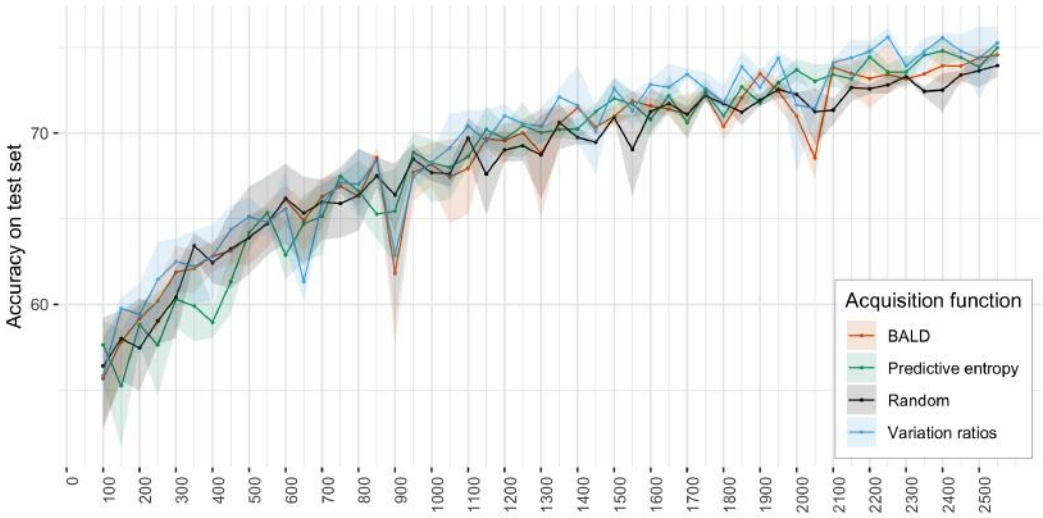
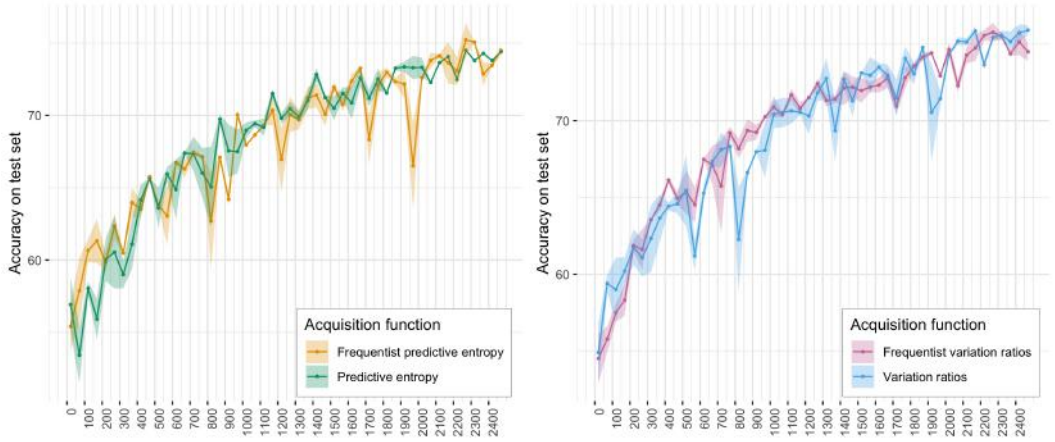


Figure 6.6: *Accuracy of models in each acquisition step in the cats and dogs dataset.*

Figure 6.7 shows the comparison of the frequentist and Bayesian acquisition functions, with predictive entropy on the left and variation ratios on the right. As with the MNIST dataset, there is no clear distinction between the frequentist and the Bayesian acquisition functions.



(a) *Predictive entropy.* **(b)** *Variation ratios.*

Figure 6.7: *Accuracy of models in each acquisition step in the cats and dogs dataset.*

6.3. CIFAR-10 dataset

The CIFAR10 was introduced in 2009. The name stands for the Canadian Institute for Advanced Research, who funded the project in which it was first used [30]. It consists of 60,000 color images of size 32×32 representing 10 different types of objects, these being *airplane*, *automobile*, *bird*, *cat*, *deer*, *dog*, *frog*, *horse*, *ship* and *truck*. The classes are balanced in the dataset, meaning that each class has the same number of images. Of the 60,000 images, 50,000 are used as a training set and the rest as test set. The training set and the test are also balanced. Figure 6.8 shows 200 randomly chosen images from the training set.

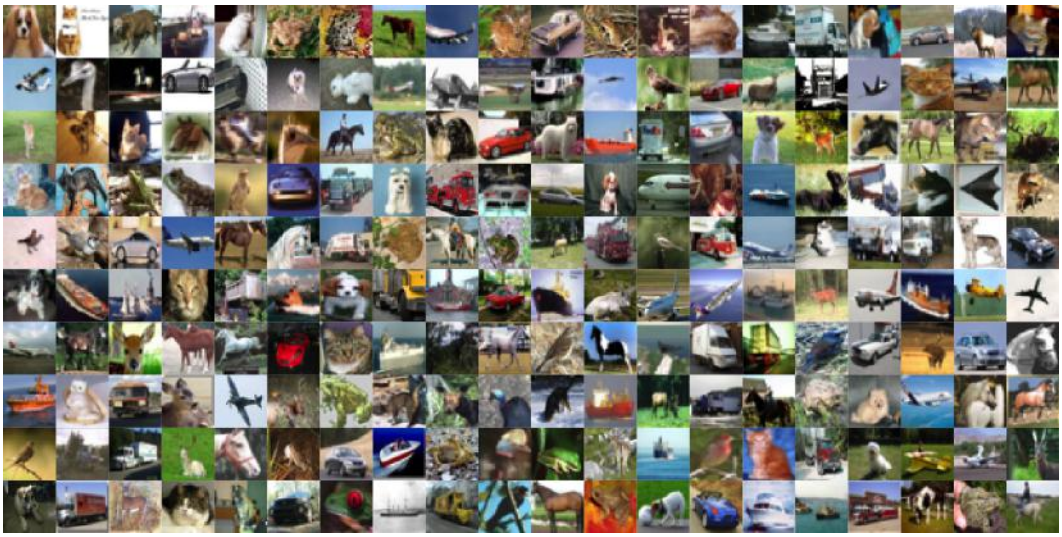


Figure 6.8: *Example of 200 randomly chosen images from the dataset.*

The process of this last experiment is very similar to the ones followed in the two previous datasets. First, as in the cats and dogs dataset, a balanced set of 100 images was randomly chosen and used to train a first model. Then a random sample of 5000 images was chosen from the pool set, from which the predicted probabilities of belonging to each class were computed. Afterwards, the 1000 images with the highest acquisition function value were chosen and added to the training set. The final number of acquisition steps was 40 and, as before, the whole process was repeated 3 times with different initial images.

The architecture used was convolution-relu-convolution-relu-max pooling-dropout-convolution-relu-convolution-relu-max pooling-dropout-dense-relu-dropout-dense-softmax with 32 convolution kernels, 3×3 kernel size, 2×2 pooling, dense layer with 512 units, and dropout probabilities 0.25, 0.25 and 0.5. The CNNs were trained for 100 epochs. Like in the previous experiments, for the Bayesian acquisition functions 100 posterior predictive distribution points were sampled by using the dropout trick.

The results of the Bayesian and random acquisition functions can be seen in figure 6.9. Here, the difference between the random acquisition function and the

rest is a little clearer than in the cats and dogs dataset, especially from around the 13,000 image mark onward, but it is not as clear as in the MNIST dataset.

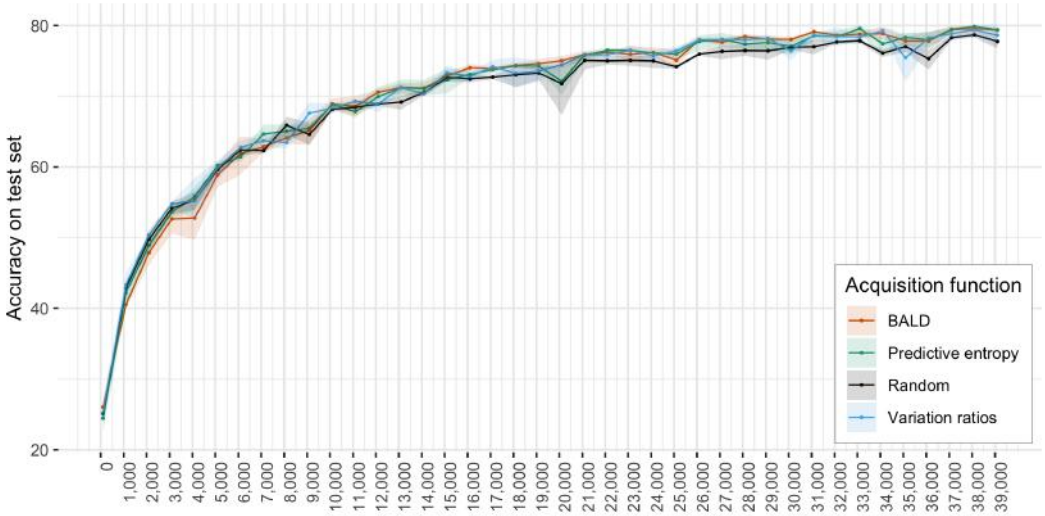
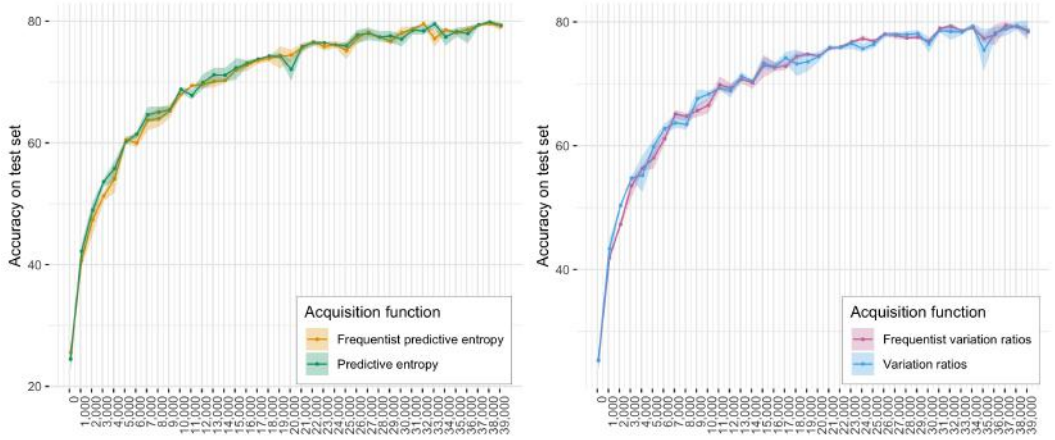


Figure 6.9: *Accuracy of models in each acquisition step in the CIFAR10 dataset.*

Figure 6.10 shows the comparison of the frequentist and Bayesian acquisition functions, with predictive entropy on the left and variation ratios on the right. As with the MNIST and with the cats and dogs datasets, the distinction between the frequentist and the Bayesian acquisition functions is not clear.



(a) *Predictive entropy.* **(b)** *Variation ratios.*

Figure 6.10: *Accuracy of models in each acquisition step in the CIFAR10 dataset.*

6.4. Discussion

In this chapter, the results of several Active Learning experiments using different acquisition functions in three different datasets were shown. In the MNIST dataset there is a clear improvement in using the predicted probabilities of a trained model to select the images to add to a training set when compared to a random selection. In the CIFAR10 dataset there seems to be a small performance improvement when using the non-random acquisition functions. In the cats and dogs dataset there is an even smaller performance improvement, but it is not as clear as in the MNIST dataset.

However, it should be noted that in the case of the CIFAR10 and the cats and dogs datasets, the problem is much more high-dimensional than in the MNIST case. In higher dimensions there is more space between images, which makes the active learning procedure behave more similarly to a random acquisition function.

In all cases, there is no clear distinction between using an approximate Bayesian model against a frequentist one, as the accuracies did not seem to be different. This again may be because of the dimensionality of the problems that are being solved.

In general, finding posterior distributions of high-dimensional spaces is difficult. In addition to this, the theoretical approximation of Dropout to a Bayesian CNN does not specify the quality of the approximation. Perhaps in the cases that were tested in this work, the variational approximation is not good and, therefore, the uncertainty given by the Bayesian acquisition functions is very similar to that of the frequentist ones.

In the following chapter, final concluding remarks are discussed.

Chapter 7

Conclusions

In this work, three Bayesian acquisition functions were compared with two frequentist acquisition functions through a series of experiments in three different image datasets: the MNIST, the CIFAR10 and a cats and dogs dataset. An attempt was made to replicate the experiments performed with the MNIST dataset in [22], but failed in obtaining the same results in the comparison of approximate Bayesian and frequentist acquisition functions. However, the attempt was successful in achieving a better performance using a non-random acquisition function in contrast to a random one. The same experimental framework was used in the other two datasets. In the CIFAR10 dataset there seems to be a small performance improvement when using the non-random acquisition functions and in the cats and dogs dataset there is an even smaller performance improvement. As in the MNIST dataset, in these two other datasets there is no clear distinction between using an approximate Bayesian model against a frequentist one. The cause of this may be that the variational approximation to the posterior distribution is not very good. With an improvement of this approximation, the results may turn out to be better.

The use of a full posterior predictive distribution is of use in many different areas. One such area is autonomous vehicles, in which knowledge of the model's

predictive uncertainty can help prevent accidents. If a model is uncertain about a prediction, then human assistance can be requested [16, 29, 37]. Another area of opportunity lies in adversarial attacks, in which the use of Bayesian CNNs helps in providing less confident predictions for adversarial examples [35, 48, 53]. An additional area is the use of uncertainty for Reinforcement Learning to accelerate learning [16]. One more general and very broad area of opportunity is in automatic classification systems using machine learning to make decisions, such as a post office automatically sorting letters according to a zip code, in which the model can decide to ask for the help of a human when the it is uncertain about certain predictions [16].

The code used to write this document is available in `https://github.com/mariobecerra/msc_thesis`.

And the code used for the experiments is available in `https://github.com/mariobecerra/Active_Learning_CNNs`.

Appendices

Appendix A

Loss function optimization

As mentioned before, in machine learning the goal is to find the parameters that minimize a loss function such as the one in equation (2.15). In some cases like linear regression, it is possible to use the first and second order conditions to find a closed formula to find the parameters; but in many other cases this cannot be done, like in logistic regression, so we resort to numerical optimization techniques. The general problem of unconstrained optimization [42] is

$$\min_{\boldsymbol{\theta} \in \mathbb{R}^p} L(\boldsymbol{\theta}). \tag{A.1}$$

In machine learning, L is usually a convex loss function and $\boldsymbol{\theta}$ is a parameter or vector of parameters. A solution is a vector $\boldsymbol{\theta}^*$ called local minimizer, which minimizes the function L in a neighborhood around $\boldsymbol{\theta}^*$. Formally, a vector $\boldsymbol{\theta}^*$ is a local minimizer if there exists a neighborhood $\mathcal{V}$ of $\boldsymbol{\theta}^*$ such that $L(\boldsymbol{\theta}^*) \leq L(\boldsymbol{\theta})$ for all $\boldsymbol{\theta} \in \mathcal{V}$.

The sufficient second order conditions are used in numerical optimization. Suppose that the Hessian matrix $\nabla^2 L$ is continuous in an open neighborhood of $\boldsymbol{\theta}^*$, that the gradient $\nabla L(\boldsymbol{\theta}^*) = 0$ and that $\nabla^2 L(\boldsymbol{\theta}^*)$ is positive definite; then $\boldsymbol{\theta}^*$ is a local minimizer of L. In general, all algorithms search for a point $\boldsymbol{\theta}^*$ such that $\nabla L(\boldsymbol{\theta}^*) = 0$ [42].

Many numerical optimization algorithms are iterative, such as coordinate descent, Newton and quasi-Newton methods, gradient free methods, and gradient descent. Coordinate descent algorithms work by fixing all but one parameter in each iteration and then minimizing the loss function with respect to the remaining parameter [14, 56]. Newton methods use a second-order Taylor series approximation to the loss function and iteratively descend by computing the Hessian matrix [42, p. 22]. Quasi-Newton methods use this same idea, but instead of computing the Hessian matrix exactly, they use some approximation to it [6] [42, p. 23]. Gradient free algorithms are used when the gradient of the loss function is not available, which could be for computational reasons, for unavailability, or practical reasons [49]. They include simulated annealing, swarm algorithms and genetic algorithms. Gradient descent is explained in the next subsection.

A.0.1. Gradient descent (GD)

Gradient descent belongs to a family of optimization algorithms called **line search algorithms**. In each iteration, these algorithms search for a direction in which to move and then update the current value in accordance to that direction [42, p. 19]. That is, in the k-th iteration, $\boldsymbol{\theta}$ has the value $\boldsymbol{\theta}_k$, and the algorithms look for a direction $\boldsymbol{p}_k$ to update to a new value $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha_k \boldsymbol{p}_k$, where $\alpha_k > 0$ is the "distance" in which the algorithm moves toward direction $\boldsymbol{p}_k$, and is called **step length**. Once that the value of the parameter is updated, the algorithm finds a new direction in which to move forward and then updates the parameter value again. This is done until a stopping criterion is met. This usually is that the gradient vector norm is smaller than a certain small positive scalar.

In gradient descent, the direction $\boldsymbol{p}_k$ in which the algorithm moves is the maximum descent direction, that is, the negative of the gradient $-\nabla L(\boldsymbol{\theta}_k)$. So, in each iteration $\boldsymbol{\theta}$ is updated as such

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha_k \nabla L(\boldsymbol{\theta}_k). \tag{A.2}$$

Choosing the step length α_k is problematic because it is desirable to find a value such that the function L decreases as much as possible, but it is not desirable to

spend too much time choosing the value. The best option is the global minimizer of the auxiliary function $\phi(\alpha_k) = L(\boldsymbol{\theta}_k + \alpha_k \boldsymbol{p}_k)$, but it may be too expensive to compute [42, p. 31]. Generally, heuristics are used to choose the sequence of values for α_k and try which one satisfies those conditions. One of those conditions is called Armijo conditions, and finds the α_k that allows a sufficient descent in the function L, measured as

$$L(\boldsymbol{\theta}_k + \alpha_k \boldsymbol{p}_k) \leq L(\boldsymbol{\theta}_k) + c_1 \alpha_k \nabla L(\boldsymbol{\theta}_k)^T \boldsymbol{p}_k, \tag{A.3}$$

for a constant $c_1 \in (0, 1)$. Usually c_1 is small, such as 10^{-4}. This condition may not be enough, because for very small values of α_k the condition can be met, and very small step lengths are not always desirable. One way to fix this is to use backtracking, which consists in choosing a big value of α_k (such as $\alpha_k = 1$), and then an iterative sub-algorithm is initiated to decrease the value of α_k until the Armijo condition is met. Another way to choose the step length α_k is using the outer product of the gradient with itself, as shown in [12].

In the following paragraphs, an example of logistic regression is shown to understand how gradient descent works.

The loss function to be minimized, defined in equation (2.15), is

$$\begin{aligned} L(\boldsymbol{\theta}) &= -\sum_{i=1}^{n} \Big[y_i \log(\sigma(\boldsymbol{\theta}^T \boldsymbol{x_i})) + (1 - y_i) \log(1 - \sigma(\boldsymbol{\theta}^T \boldsymbol{x_i})) \Big] \\ &= -\sum_{i=1}^{n} \ell_i(\boldsymbol{\theta}) \end{aligned} \tag{A.4}$$

where $\sigma(\cdot)$ is the logistic sigmoid function, $\boldsymbol{\theta}^T \boldsymbol{x_i} = \sum_{j=0}^{p} \theta_j x_{ij}$, $x_{i1} = 1$ for all $i \in \{1, ..., n\}$ and $\ell_i(\boldsymbol{\theta})$ is defined as

$$\ell_i(\boldsymbol{\theta}) = y_i \log(\sigma(\boldsymbol{\theta}^T \boldsymbol{x_i})) + (1 - y_i) \log(1 - \sigma(\boldsymbol{\theta}^T \boldsymbol{x_i})). \tag{A.5}$$

Taking the partial derivatives of the loss function with respect to the parameters the result is

$$\frac{\partial L}{\partial \theta_j} = -\sum_{i=1}^{n} \frac{\partial \ell_i}{\partial \theta_j} \tag{A.6}$$

and, using the fact that $\sigma'(w) = \sigma(w)(1-\sigma(w))$, then each partial derivative from the sum

$$\begin{aligned}
\frac{\partial \ell_i}{\partial \theta_j} &= \frac{y_i \sigma'(\boldsymbol{\theta}^T \boldsymbol{x_i}) x_{ij}}{\sigma(\boldsymbol{\theta}^T \boldsymbol{x_i})} + \frac{(1-y_i)(-1)\sigma'(\boldsymbol{\theta}^T \boldsymbol{x_i}) x_{ij}}{1-\sigma(\boldsymbol{\theta}^T \boldsymbol{x_i})} \\
&= \frac{\sigma'(\boldsymbol{\theta}^T \boldsymbol{x_i}) x_{ij} y_i}{\sigma(\boldsymbol{\theta}^T \boldsymbol{x_i})} - \frac{(1-y_i)\sigma'(\boldsymbol{\theta}^T \boldsymbol{x_i}) x_{ij}}{1-\sigma((\boldsymbol{\theta}^T \boldsymbol{x_i}))} \\
&= \sigma'(\boldsymbol{\theta}^T \boldsymbol{x_i}) x_{ij} \left(\frac{y_i}{\sigma(\boldsymbol{\theta}^T \boldsymbol{x_i})} - \frac{1-y_i}{1-\sigma(\boldsymbol{\theta}^T \boldsymbol{x_i})} \right) \\
&= \sigma'(\boldsymbol{\theta}^T \boldsymbol{x_i}) x_{ij} \left(\frac{y_i - y_i \sigma(\boldsymbol{\theta}^T \boldsymbol{x_i}) - \sigma(\boldsymbol{\theta}^T \boldsymbol{x_i}) + y_i \sigma(\boldsymbol{\theta}^T \boldsymbol{x_i})}{\sigma(\boldsymbol{\theta}^T \boldsymbol{x_i})(1-\sigma(\boldsymbol{\theta}^T \boldsymbol{x_i}))} \right) \\
&= x_{ij}(y_i - \sigma(\boldsymbol{\theta}^T \boldsymbol{x_i})).
\end{aligned} \tag{A.7}$$

Finally, the partial derivative is

$$\frac{\partial L}{\partial \theta_j} = -\sum_{i=1}^{n} x_{ij}(y_i - \sigma(\sum_{j=0}^{p} \theta_j x_{ij})). \tag{A.8}$$

Hence, the gradient descent direction for each iteration is

$$\nabla_{\boldsymbol{\theta}} L = \left(\frac{\partial L}{\partial \theta_1}, ..., \frac{\partial L}{\partial \theta_p} \right)^T. \tag{A.9}$$

To minimize the loss function, an initial vector of parameters $\boldsymbol{\theta}^0 \in \mathbb{R}^p$ is chosen, and in each iteration this vector is updated using equation (A.2) until certain criteria are met.

This algorithm was implemented in the R programming language [45] by generating a data matrix $\boldsymbol{X} \in \mathbb{R}^{n \times 1}$ with $n = 1000$ such that $\boldsymbol{x_i} \sim N(0,1)$ for each $i \in \{1, ..., n\}$. Then an auxiliary vector was computed, $p_i = \frac{1}{1+\exp\left(-\theta_0 - \theta_1 x_i^{(1)}\right)}$, with $\theta_0 = -5$ and $\theta_1 = 5$. Finally, the response variable $\boldsymbol{y}$ was built simulating Bernoulli random variables, such that $y_i \sim \text{Bern}(p_i)$. The implementation is compared with the result of the `glm` package.

The initial vector of parameters was $\boldsymbol{\theta}^0 = (0,0)^T$ and a constant value of $\alpha_k = 0.1$ was used. The stopping criterion was that the norm of the gradient, i.e. $\|\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})\|$, should be less than 10^{-8}, this being the approximate square root of

the double-precision floating-point format; or that the norm of the difference of the parameters from one iteration to the other, i.e. $\|\boldsymbol{\theta}_{k+1} - \boldsymbol{\theta}_k\|$, should be less than 10^{-8}. This was achieved after 3,704 iterations.

Figure A.1 shows the results of the implementation. On the left, the value of the loss function in each iteration can be seen. On the right, the parameter vector's value in each iteration. The red dot is the value of the estimate by the `glm` package. It can be seen that the implemented algorithm converges to this value.

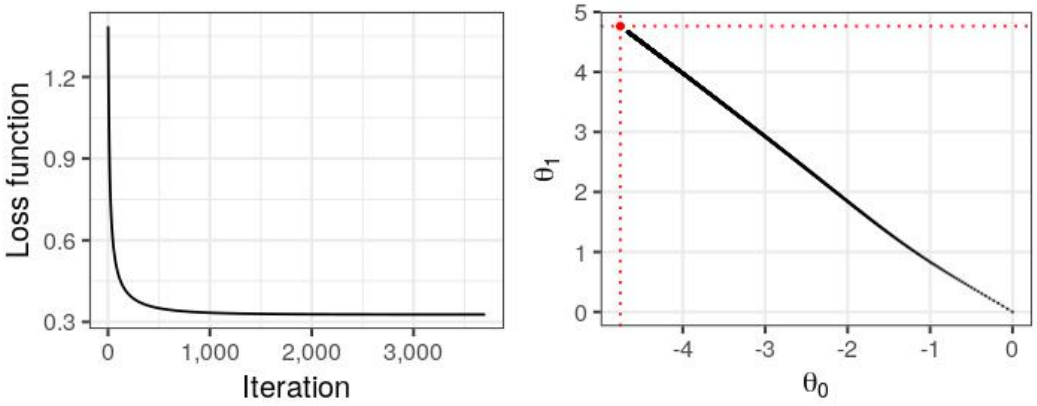


Figure A.1: *Example of gradient descent for logistic regression.*

A.0.2. Stochastic gradient descent (SGD)

In machine learning it is common to assume that observations are independent and identically distributed, thus, it is also common to find the need to solve optimization problems of the form

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^p} L(\boldsymbol{\theta}), \quad \text{with } L(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} \ell_i(\boldsymbol{\theta}). \tag{A.10}$$

That is, the loss function that needs to be minimized is usually a sum or average of several indexed functions in which each indexed function depends only on one observation of the data set. For example, in logistic regression, the loss function that is expressed in that way.

Gradient descent uses iterations in the form

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha_k \nabla L(\boldsymbol{\theta}_k) := \boldsymbol{\theta}_k - \frac{\alpha_k}{n} \sum_{i=1}^{n} \nabla \ell_i(\boldsymbol{\theta}_k), \tag{A.11}$$

which involves evaluating n gradients (one for each observation) and then taking an average. In some cases of machine learning, n can be really big; hence computing all of those gradients in each iteration is expensive. That is why methods such as SGD are used, in which the number of gradients to compute does not depend on n but instead it is constant. SGD uses iterations of the form

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha_k \nabla \ell_{i_k}(\boldsymbol{\theta}_k), \tag{A.12}$$

where $i_k \in \{1, 2, ..., n\}$ is randomly chosen. The gradient $\nabla \ell_{i_k}(\boldsymbol{\theta}_k)$ is an unbiased estimator of $\nabla L(\boldsymbol{\theta}_k)$. This way, each iteration is computationally inexpensive because it involves the computation of only one gradient. It can happen that some $\nabla \ell_{i_k}(\boldsymbol{\theta}_k)$ in particular does not give a descent direction from $\boldsymbol{\theta}_k$, but on average they yield descent directions. Thus, the sequence $\{\boldsymbol{\theta}_0, \boldsymbol{\theta}_1, ...\}$, under certain conditions, leads to a minimizer $\boldsymbol{\theta}^*$.

A.0.3. Mini-batch gradient descent

An approach that lies between the two extremes of computing the gradients of all observations and the gradient of just one observation is **mini-batch gradient descent**. In mini-batch gradient descent, one chooses a fixed integer l, then the data set is divided in batches of size l, where the values in each batch are randomly chosen. Then, each of these batches is used to compute the gradient and update the values of the parameters. Usually l is a small number compared to the size of the data set, but big enough so that the gradient estimation is not too noisy, such as $l = 32$ or $l = 100$. This way, each iteration is cheaper because it involves the computation of only l gradients instead of n, with $l << n$. Stochastic gradient descent (SGD) is just mini-batch gradient descent with $l = 1$.

So, mini-batch gradient descent updates the value of the parameters in each

iteration as such

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \frac{\alpha_k}{n} \sum_{i=1}^{l} \nabla \ell_i(\boldsymbol{\theta}_k). \tag{A.13}$$

We implemented mini-batch gradient descent for logistic regression in R and test it with the same simulated data set as in the previous example. Figure A.2 shows the results of the implementation. Each of the plots shows the values of the parameters in each iteration, but the difference in each plot is the size of the mini-batch l. It is clear that with bigger l, the descent directions are less noisy, but in the end they all converge to more or less the same value.

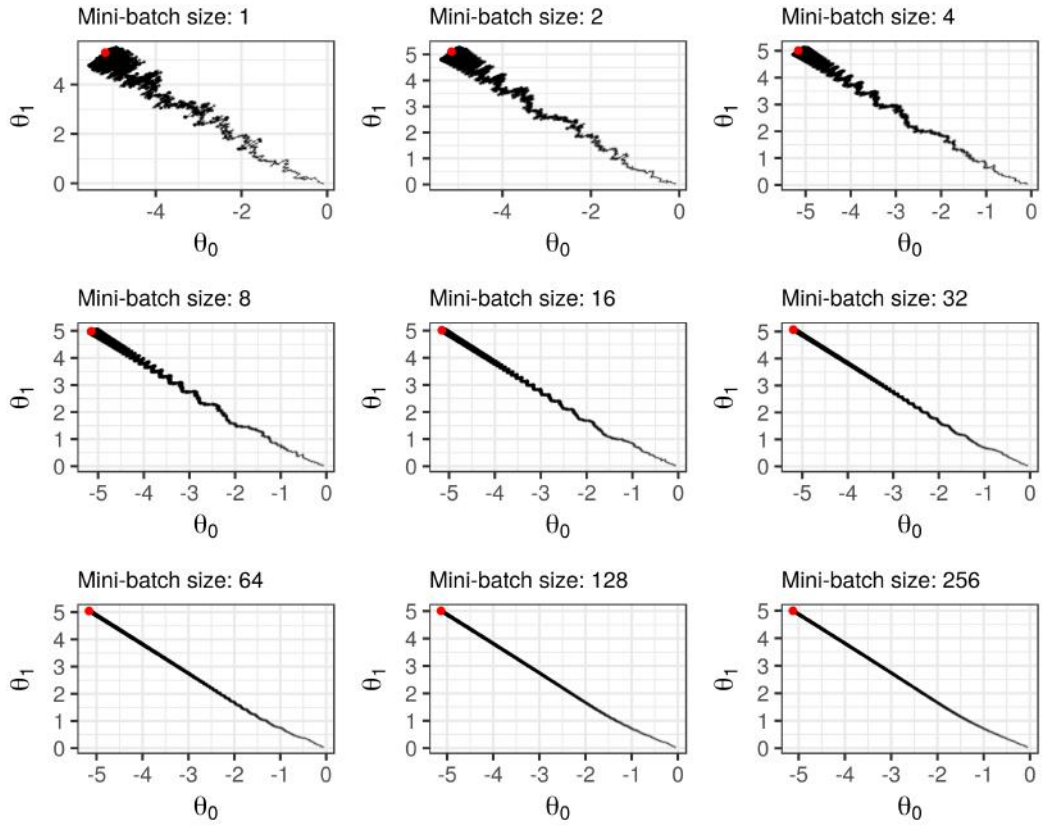


Figure A.2: *Example of mini-batch gradient descent for logistic regression comparing mini-batch sizes.*

Appendix B

Images with highest uncertainty

Figures B.1, B.2 and B.3 show the most uncertain examples for a Bayesian CNN using the variation ratios acquisition function with the MNIST, cats and dogs, and CIFAR10 datasets, respectively.

In the case of the MNIST and cats and dogs datasets, some unusual examples can be seen. For instance, in the case of the third row and first column of figure B.1, it is not clear what number it is. The same happens with the ninth row and second column. In figure B.2, the element in the second row and second column is a cat on books in a book-shelf, but it is not clear at first glance. And the picture in the fourth row and fourth column is peculiar because it has a star shape. On the contrary to the previous two datasets, in the case of the cats and dogs dataset, figure B.3 does not show any obvious deviation from the rule.

Appendix B: Images with highest uncertainty

Figure B.1: *Images with highest uncertainties in the MNIST dataset.*

Figure B.2: *Images with highest uncertainties in the cats and dogs dataset.*

Appendix B: Images with highest uncertainty

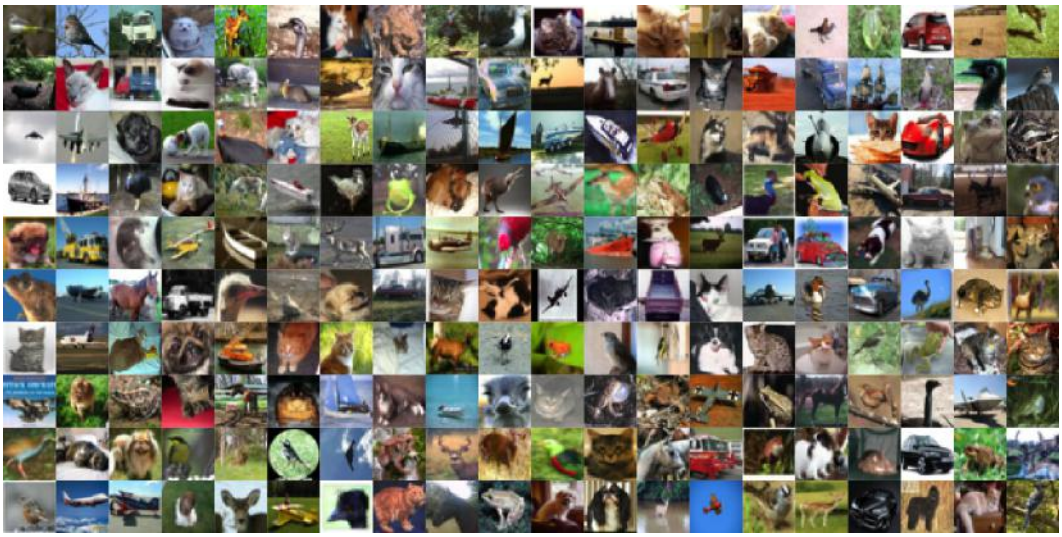


Figure B.3: *Images with highest uncertainties in the CIFAR10 dataset.*

Bibliography

[1] Martín Abadi and Agarwal A Barham P TensorFlow. «Large-scale machine learning on heterogeneous distributed systems». In: *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI'16)(Savannah, GA, USA*. 2016, pp. 265–283.

[2] JJ Allaire, Kevin Ushey, and Yuan Tang. *reticulate: Interface to 'Python'*. R package version 1.10. 2018. URL: `https://CRAN.R-project.org/package=reticulate`.

[3] Yoshua Bengio, Ian J Goodfellow, and Aaron Courville. «Deep learning». In: (2015).

[4] Christopher M.. Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[5] David M Blei, Alp Kucukelbir, and Jon D McAuliffe. «Variational inference: A review for statisticians». In: *Journal of the American Statistical Association* 112.518 (2017), pp. 859–877.

[6] Richard H Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. «A limited memory algorithm for bound constrained optimization». In: *SIAM Journal on Scientific Computing* 16.5 (1995), pp. 1190–1208.

[7] François Chollet et al. *Keras*. `https://keras.io`. 2015.

[8] David A Cohn, Zoubin Ghahramani, and Michael I Jordan. «Active learning with statistical models». In: *Journal of artificial intelligence research* 4 (1996), pp. 129–145.

[9] Richard T Cox. «Probability, frequency and reasonable expectation». In: *American journal of physics* 14.1 (1946), pp. 1–13.

[10] Richard T Cox. «The algebra of probable inference». In: *American Journal of Physics* 31.1 (1963), pp. 66–67.

[11] John Denker and Yann Lecun. «Transforming Neural-Net Output Levels to Probability Distributions». In: *Advances in Neural Information Processing Systems 3*. Citeseer. 1991.

[12] John Duchi, Elad Hazan, and Yoram Singer. «Adaptive subgradient methods for online learning and stochastic optimization». In: *Journal of Machine Learning Research* 12.Jul (2011), pp. 2121–2159.

[13] Jeremy Elson, John JD Douceur, Jon Howell, and Jared Saul. «Asirra: a CAPTCHA that exploits interest-aligned manual image categorization». In: (2007).

[14] Jerome Friedman, Trevor Hastie, Holger Höfling, Robert Tibshirani, et al. «Pathwise coordinate optimization». In: *The Annals of Applied Statistics* 1.2 (2007), pp. 302–332.

[15] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*. Vol. 1. Springer series in statistics Springer, Berlin, 2001.

[16] Yarin Gal. «Uncertainty in deep learning». In: *University of Cambridge* (2016).

[17] Yarin Gal and Zoubin Ghahramani. «A theoretically grounded application of dropout in recurrent neural networks». In: *Advances in neural information processing systems*. 2016, pp. 1019–1027.

[18] Yarin Gal and Zoubin Ghahramani. «Bayesian convolutional neural networks with Bernoulli approximate variational inference». In: *arXiv preprint arXiv:1506.02158* (2015).

[19] Yarin Gal and Zoubin Ghahramani. «Dropout as a Bayesian approximation: Insights and applications». In: *Deep Learning Workshop, ICML*. Vol. 1. 2015, p. 2.

[20] Yarin Gal and Zoubin Ghahramani. «Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning». In: *arXiv preprint arXiv:1506.02142* (2015).

[21] Yarin Gal and Zoubin Ghahramani. «On modern deep learning and variational inference». In: *Advances in Approximate Bayesian Inference workshop, NIPS*. Vol. 2. 2015.

[22] Yarin Gal, Riashat Islam, and Zoubin Ghahramani. «Deep Bayesian Active Learning with Image Data». In: *Bayesian Deep Learning workshop, NIPS*. 2016.

[23] Andrew Gelman and Jennifer Hill. *Data analysis using regression and multilevel/hierarchical models*. Cambridge university press, 2006.

[24] Andrew Gelman, Hal S Stern, John B Carlin, David B Dunson, Aki Vehtari, and Donald B Rubin. *Bayesian data analysis*. Chapman and Hall/CRC, 2013.

[25] Alex Graves. «Practical variational inference for neural networks». In: *Advances in Neural Information Processing Systems*. 2011, pp. 2348–2356.

[26] José Miguel Hernández-Lobato and Ryan Adams. «Probabilistic backpropagation for scalable learning of bayesian neural networks». In: *International Conference on Machine Learning*. 2015, pp. 1861–1869.

[27] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning*. Vol. 6. Springer, 2013.

[28] Edwin T Jaynes. *Probability theory: The logic of science*. Cambridge university press, 2003.

[29] Alex Kendall and Yarin Gal. «What uncertainties do we need in bayesian deep learning for computer vision?» In: *Advances in neural information processing systems*. 2017, pp. 5574–5584.

[30] Alex Krizhevsky and Geoffrey Hinton. *Learning multiple layers of features from tiny images*. Tech. rep. Citeseer, 2009.

[31] Alp Kucukelbir, Dustin Tran, Rajesh Ranganath, Andrew Gelman, and David M Blei. «Automatic differentiation variational inference». In: *The Journal of Machine Learning Research* 18.1 (2017), pp. 430–474.

[32] Suhas Kumar. «The End of Moore's Law and Reinventing Computing». In: *High-Speed and Lower Power Technologies: Electronics and Photonics* (2018).

[33] Yann LeCun. «Generalization and network design strategies». In: *Connectionism in perspective* (1989), pp. 143–155.

[34] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. «Gradient-based learning applied to document recognition». In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[35] Yingzhen Li and Yarin Gal. «Dropout inference in bayesian neural networks with alpha-divergences». In: *arXiv preprint arXiv:1703.02914* (2017).

[36] David JC MacKay. «A practical Bayesian framework for backpropagation networks». In: *Neural computation* 4.3 (1992), pp. 448–472.

[37] Rhiannon Michelmore, Marta Kwiatkowska, and Yarin Gal. «Evaluating Uncertainty Quantification in End-to-End Autonomous Driving Control». In: *arXiv preprint arXiv:1811.06817* (2018).

[38] Gordon E Moore. «Cramming more components onto integrated circuits, Reprinted from Electronics, volume 38, number 8, April 19, 1965, pp. 114 ff.» In: *IEEE Solid-State Circuits Society Newsletter* 11.3 (2006), pp. 33–35.

[39] Gordon E Moore et al. «Progress in digital integrated electronics». In: *Electron Devices Meeting*. Vol. 21. 1975, pp. 11–13.

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

[41] Radford M Neal. «Bayesian Learning for Neural Networks». In: (1996).

[42] Jorge Nocedal and Stephen J Wright. *Numerical Optimization, Second Edition*. Springer New York, 2006.

[43] Antony O'Hagan and Jon Forster. «The advanced theory of statistics, Vol. 2B: Bayesian inference». In: *London, UK: Arnold* (2004).

[44] Fredrik Olsson. «A literature survey of active machine learning in the context of natural language processing». In: (2009).

[45] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2016. URL: `https://www.R-project.org/`.

[46] Rajesh Ranganath. «Black Box Variational Inference: Scalable, Generic Bayesian Computation and its Applications». PhD thesis. Princeton University, 2017.

[47] Rajesh Ranganath, Sean Gerrish, and David Blei. «Black box variational inference». In: *Artificial Intelligence and Statistics*. 2014, pp. 814–822.

[48] Ambrish Rawat, Martin Wistuba, and Maria-Irina Nicolae. «Adversarial Phenomenon in the Eyes of Bayesian Deep Learning». In: *arXiv preprint arXiv:1711.08244* (2017).

[49] Luis Rios, Nikolaos Sahinidis, et al. «Derivative-free optimization: a review of algorithms and comparison of software implementations». In: *Journal of Global Optimization* 56.3 (2013), pp. 1247–1293.

[50] Christian Robert. *The Bayesian choice: from decision-theoretic foundations to computational implementation*. Springer Science & Business Media, 2007.

[51] Burr Settles. *Active Learning Literature Survey*. Computer Sciences Technical Report 1648. University of Wisconsin–Madison, 2009.

[52] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. «Mastering the game of Go without human knowledge». In: *Nature* 550.7676 (2017), p. 354.

[53] Lewis Smith and Yarin Gal. «Understanding Measures of Uncertainty for Adversarial Example Detection». In: *arXiv preprint arXiv:1803.08533* (2018).

[54] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. «Dropout: a simple way to prevent neural networks from overfitting». In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[55] Richard S Sutton, Andrew G Barto, Francis Bach, et al. *Reinforcement learning: An introduction*. MIT press, 1998.

[56] Stephen J Wright. «Coordinate descent algorithms». In: *Mathematical Programming* 151.1 (2015), pp. 3–34.