Delft University of Technology
Faculty Electrical Engineering, Mathematics and Computer Science
Delft Institute of Applied Mathematics

Estimating model uncertainty for conditional prepayment rate predictions using artificial neural networks with dropout

Report for the
Delft Institute of Applied Mathematics
as part of

the degree of

MASTER OF SCIENCE
in
APPLIED MATHEMATICS

by

STIJN BART PIETER VERBERNE

Delft, The Netherlands
May 2019

MSc thesis APPLIED MATHEMATICS

”Estimating model uncertainty for conditional prepayment rate predictions using artificial neural networks with dropout.”

STIJN BART PIETER VERBERNE

Delft University of Technology

Thesis advisor

Prof. dr. ir. C.W. Oosterlee

Other members of the graduation committee

Dr. A.I. Borovykh

Dr. P. Cirillo

Dr. P.W. den Iseger

Delft, May 2019

Abstract

Clients with a mortgage loan may prepay a part of their loan before the contractual date. This is called prepayment. In the case of a prepayment, the bank who issued the loan earns less interest than initially agreed. It is therefore essential to build accurate models for predicting prepayment behavior. In this thesis, machine learning models called artificial neural networks are used to predict conditional prepayment rates. In particular, the possibility to quantify the model uncertainty is studied.
It is important to acknowledge that the accuracy of a model is not constant over its domain. Most machine learning methods do not provide information about the uncertainty regarding a prediction and, unfortunately, methods that do so are often computationally expensive. This thesis studies uncertainty estimates generated by a neural network with dropout applied. Dropout is a method that randomly drops out neurons in each layer and has been suggested to prevent overfitting. We will see that this method can also be used to extract a measure of uncertainty regarding the prediction of a neural network efficiently.
This method is used first to evaluate uncertainty for three simple functions under different distributions of the train data. Then the uncertainty estimates are evaluated for a neural network that predicts conditional prepayment rates. It will be illustrated that the uncertainty estimates provide an accurate indication of regions of the domain where predictions are inaccurate. Moreover, using a different model in the regions identified as uncertain can result in higher model performance.

Keywords: Artificial neural networks, Dropout, Uncertainty, Mortgages, Conditional Prepayment Rate.

Acknowledgements

This thesis has been submitted in partial fulfillment of the requirements for the degree of Master of Science in Applied Mathematics at Delft University of Technology. The academic supervisor of the project has been Prof. dr. ir. C.W. Oosterlee of the Numerical Analysis group at the Delft Institute of Applied Mathematics. The research has been conducted under the supervision of dr. P.W. den Iseger at the ABN AMRO office in Amsterdam.

I would like to thank Kees Oosterlee, Anastasia Borovykh, Peter den Iseger and Dirk Broersen for their help and guidance during this research.

Stijn Verberne
Delft, May 2019

Contents

List of symbols

Symbols related to artificial neural networks

$\mathbf{b}^{(l)}$	The bias vector of layer l
d_l	The amount of neurons in layer l
$E(\cdot)$	Loss function
$\mathbf{f}(\cdot)$	Output of the neural network
L	The amount of layers in a neural network
$\mathcal{L}(\cdot)$	Minimization objective
$M^{(l)}$	The optimized weight matrix of layer l
$\mathbf{n}^{(l)}$	The optimized bias vector of layer l
$p(\cdot)$	Likelihood
p_l	The neuron activity rate of layer l
$\mathbf{v}^{(l)}$	Vector of random variables used to parametrize the weight matrix of layer l
$\mathbf{v}$	Collection of vectors $\mathbf{v}^{(l)}$
$\hat{\mathbf{v}}$	Realization of $\mathbf{v}$
$W^{(l)}$	The weight matrix of layer l
$\mathbf{w}_i^{(l)}$	The i-th row of weight matrix $W^{(l)}$
$w_{ij}^{(l)}$	The element on row i and column j of weight matrix $W^{(l)}$
$\mathbf{x}_i$	Vector representing feature values of the i-th observation
$\mathbf{y}_i$	Vector representing behavior/output of the i-th observation
$\gamma^{(l)}$	Prior length-scale of layer l
η	Learning rate of the optimizer
θ	The collection of optimized weights and biases used in a neural network
$\lambda^{(l)}$	Regularization coefficient of layer l
$\sigma(\cdot)$	Activation function
τ	Model precision
ω	Collection of weights and biases used in a neural network
$\hat{\omega}$	Realized parametrization of ω

Other symbols

$\mathbb{E}(\cdot)$	Expected value
$\mathcal{H}(\cdot)$	Differential entropy
$\mathcal{MN}$	Matrix Gaussian distribution
$\mathcal{N}$	Gaussian distribution
$\mathbb{N}$	Natural numbers
$\mathbb{P}(\cdot)$	(Predicted) probability of an event
$\mathbb{R}$	Real numbers
$\mathbb{R}_{\geq 0}$	Positive real numbers
$\delta(\cdot)$	Dirac delta function
∇	Differential operator
$\circ$	Hadamard product
$\|\cdot\|$	Euclidean distance
$\|\cdot\|_A$	Mahalanobis distance with covariance matrix A

Abbreviations / Acronyms

AUC	Area Under the Curve
BNN	Bayesian Neural Network
CE	Cross Entropy
CPR	Conditional Prepayment Rate
HPI	House Price Index
KL	Kullback-Leibler
MC	Monte Carlo
MSE	Mean Squared Error
ReLU	Rectified Linear Unit
ROC	Receiver Operating Characteristic
VI	Variational Inference

Chapter 1

Introduction

A *mortgage* is a loan provided to a client for funding real estate. For many banks, mortgages play a large role on the balance sheet which makes it important to quantify the risks paired with these loans accurately.
A key risk concerning mortgages is the possibility that a client defaults and is unable to repay the full outstanding loan. On the other hand, a client may repay (a part of) his/her loan before the contractual date. Such a transaction is called a *prepayment*. In the case of a prepayment, the bank receives less interest for the loan than initially agreed. The money that is paid back earlier causes a funding gap and forces the bank to invest with a return that is possibly lower than on the issued mortgage.
To calculate the prepayment risk to which a bank is exposed, it is important to make an accurate estimate of the fraction of clients that will prepay their mortgage.
Within this group of clients, there are several types in which prepayments can be divided varying from full prepayment to a fraction of the outstanding loan that is prepaid. To make an accurate estimate of the risk, the bank has to treat these scenarios separately.
The prepayment behavior predictions could be used to get insight in how macro-economic changes may affect the balance sheet of a bank. Also it can be used to calculate the size of penalties that should be given to clients that prepay their mortgages in a specific way.
In this thesis, we will discuss methods to predict prepayment behavior by using a machine learning method called *artificial neural networks*.

Artificial neural networks

Artificial neural networks are systems that are inspired by the human brain where neurons process incoming information. In general, a neural network cannot be written as an explicit function that directly connects input to output but rather consists of many nonlinear computations that jointly generate an

output.
The main criticism on neural networks is that they do not give rise to an 'interpretable' function. The model is shaped by the data on which it is trained. This results in a complex network in which input data is proceeded by multiple nonlinear functions. In general, the resulting model can not be simplified to an understandable function. There are two important reasons to be cautious when dealing with these kind of 'black box' models:

1. *The uncertainty of the prediction is not always clear.* The output of a machine learning algorithm is in general a point estimate. A prediction generated by a regular neural network does not provide direct information about the network's uncertainty regarding the prediction. A user can therefore be mislead when he/she is unaware that a prediction can be highly inaccurate.

2. *The model lacks mathematical justification from which governing variable structures can be extracted easily.* When one builds a model based on direct relationships between the inputs of the model and the outputs, the sensitivity of the prediction with respect to the variables is clear. Whereas in a neural network, highly nonlinear relationships could be contained in the model while they can not be directly observed in a representing function. This is especially the case when one deals with a multidimensional space and the characteristics of the resulting network on the domain can not be examined completely.

This thesis will focus on the uncertainty regarding predictions of a neural network. The governing variable structures of a neural network will be discussed briefly.

Outline of the thesis

In chapter 2, we start with a brief history of artificial neural networks and discuss earlier research about predicting prepayment behavior using neural networks. Then a description will be given of the existing literature on how uncertainty regarding a neural network's prediction can be quantified. In order to model prepayment behavior, a distinction will be made between different types of prepayments and possible factors that could influence them. Also a basic model for predicting prepayment behavior is discussed.
In chapter 3, artificial neural networks will be discussed mathematically. First the basic structure of a neural network is explained. A distinction will be made between regression and classification tasks. Then the process of training a network is described.
Chapter 4 deals with the problem of uncertainty estimates for the predictions of a neural network. A method called dropout is suggested to extract a measure of uncertainty for predictions. Also, a proof to justify this method will be provided.

Chapter 5 discusses the capability of specific setups of neural networks to capture relevant functions under different data structures. It will be illustrated that the performance of a trained neural network strongly depends on the data on which it is trained. We also evaluate the uncertainty estimates obtained using neural networks trained with dropout.

In chapter 6, the characteristics of datasets concerning prepayments will be treated. Then, a neural network is trained to predict a specific type of prepayment and its performance is assessed. Subsequently we will train a neural network with dropout to predict prepayment behavior. Uncertainty estimates will be generated for this neural network to study how they can be used as an indication for the performance of a neural network. Also, the sensitivity of predictions with respect to the different driving variables is discussed briefly.

Chapter 7 contains a summary of the results and suggests further research opportunities.

Chapter 2

Background

This chapter contains a brief history of neural networks. In particular, suggested methods to obtain model uncertainty are compared. We will also mention two studies that used neural networks to predict prepayment behavior. In order to understand how prepayment behavior is modelled, a distinction between different types of prepayment and the variables influencing them are treated. The chapter concludes with the description of a basic model to predict prepayment behavior.

2.1 Artificial neural networks

The past decades, machine learning has become an increasingly popular research topic. Neural networks in particular, as an example of machine learning algorithms, have attracted a lot of attention. The increasing amount of available data in combination with growing computational power provides many opportunities for this method.

Neural networks are inspired by the human brain where perceptions are processed by many neurons [Graupe, 2013]. A neural network does not follow predefined rules but rather depends on many (nonlinear) functions that jointly construct an output.
A huge step was made by Rumelhart et al. [1986] by showing that neural networks are able to iteratively learn interpretations of data by using the back propagation algorithm. Another important discovery was made later in the 1980's when neural networks with as much as a single hidden layer were proven to be universal approximators [Hornik et al., 1989] [Cybenko, 1989]. This important step proved that (relatively simple) networks are able to capture highly nonlinear and complex relationships [Lek et al., 1994].
Since then, neural networks have been widely used in for example image recognition [Ciresan et al., 2012] and ecological modelling [Gevrey et al., 2002] [Gevrey et al., 2006]. With increasing speed,

new applications of neural networks are studied, amongst others for predicting prepayment behavior.

Sirignano et al. [2018] and Saito [2018] have modelled prepayment behavior using neural networks. Sirignano et al. [2018] used a dataset containing information on 120 million mortgages in the US. Their focus is mainly on the influence of the different drivers of prepayment behavior. The relative sensitivity of prepayments with respect to the variables is studied in their paper. Also, they treated the sensitivity with respect to pairs and triplets of variables.
Saito [2018] has modelled prepayment behavior using artificial neural networks and random forests. The focus is mainly on the structure of the dataset of historical prepayments. Prepayments are rare events which means that they occur with a low probability. As a result, there are relatively few prepayment observations. Saito [2018] treats the necessary dataset adjustments in order to train a neural network well.

The sensitivity with respect to different variables (see section 6.6) as well as the dataset adjustments (see section 6.1) will be discussed in this thesis. The main focus however will be on uncertainty of predictions made by a neural network.

2.1.1 Output uncertainty

Although neural networks offer us a powerful tool to construct models, they have important limitations. Where probabilistic models allow the user to reason about the confidence of a prediction, many machine learning techniques provide no more than a point estimate. This is also the case for regular artificial neural networks.
Multiple methods are suggested to quantify the uncertainty regarding the prediction of a neural network.

The most intuitive method is to train multiple neural networks independently on the same dataset. A combination of the individual networks can be taken as the prediction and the difference between the predictions can be used to generate confidence bounds. The use of a combination of multiple neural networks to compute a more general model is called a neural network *ensemble* [Hansen and Salamon, 1990] [Dietterich, 2000]. Since this method implies training multiple networks it is computationally very demanding which makes it impractical for large problems.

Leonard et al. [1992] use the idea that the accuracy of a model relies on the distribution of the data. They suggest a network based on radial basis functions to make computations within the neural network. The observations that lie close to the input of the network contribute more to the network's

output. This way, also a measure for local fit can be computed which could also act as an indicator of the confidence in the output of the network.
This method requires evaluation of the neighbourhood for all observations. When dealing with prepayments, exploration of this neighourhood is computationally very expensive since the domain is multidimensional. Therefore this method is impractical for generating uncertainties regarding prepayment predictions.

Another suggestion uses Bayesian neural networks (BNNs). In a BNN, a distribution is defined over the model parameters [MacKay, 1992]. From there, distributions over predictions can be generated which allow the user to reason about confidence. On the downside, BNNs are computationally very expensive [Gal, 2016].
This problem can be overcome by using a method called dropout. When dropout is applied to a network, a fraction of the neurons is dropped out during each training iteration. Gal and Ghahramani [2015a] and Gal and Ghahramani [2016] show that a neural network of arbitrary depth with dropout applied can be interpreted as a Bayesian neural network. This interpretation allows us to extract uncertainty measures from a neural network trained with dropout.

In this thesis, we will focus on computing and evaluating uncertainty estimates for predictions by using neural networks with dropout applied. In particular, the relationship between the estimated uncertainty and the performance of a neural network is studied. Also the use of ensembles in a specific scenario will be discussed.

2.2 Prepayment risk

As briefly noted in the introduction, a prepayment is a non contractual (partly) repayment of a loan. In this section we will discuss the different types of prepayment and variables influencing the prepayment behavior of clients. Also, we will briefly mention the characteristics of a basic model that can be used for modelling prepayment rates.

2.2.1 Classification of prepayments

There are several ways a client can prepay a mortgage. The following categories of prepayments can be distinguished [Kuijenhoven et al., 2016]:

1. Relocation: Full repayment of a loan part following the sale of the collateral.

2. Refinancing externally: Full repayment of a loan part without the collateral being sold.

3. Curtailment high: Partial repayment of the loan, where the additional repaid amount is more than 10.5% of the original principal[1].

4. Curtailment low: Partial repayment of the loan, where the additional repaid amount is less than or equal to 10.5% of the original principal.

5. Reconsider (exercise or, in Dutch, *rentebedenktijdoptie*): An early interest rate reset during an interest reconsider period.

6. Renteafkoop: Fixing a new interest rate and/or maturity of the contract.

7. No Prepayment: Only contractual repayments.

In order to quantify the risk for a bank that is caused by prepayments, it is necessary to predict the fraction of mortgages in a portfolio that belongs to each of the classes defined above. This fraction of prepayments that occurs on a monthly basis is called the *conditional prepayment rate* (CPR). The term 'prepayment rate' also refers to the CPR in this thesis. The prediction of a client's decision for a prepayment is assumed to depend on loan, client and macro-economic characteristics.
In the cases of refinancing and curtailment high, a penalty may be given to the client. The calculation of penalties is not treated in this thesis. It is discussed in detail in Pardoel [2015].

2.2.2 Explanatory variables

The prepayment behavior of clients depends on multiple variables. The most important ones can roughly be split into three groups: Loan specific drivers, client specific drivers and macroeconomic drivers [Perry et al., 2015].
Examples of explanatory variables that could be used in models to assess prepayment behavior are [Sirignano et al., 2018] [Kuijenhoven et al., 2016]:

- Market mortgage interest rate
- Contractual interest rate
- Amortization type
- Loan age
- House Price Index (HPI)
- Debt-to-income ratio

[1]The distinction in curtailments is made since prepayments less than 10% are penalty-free for most contracts. The margin of 0.5% prevents classifying low curtailments wrongly in the case of low data quality [Kuijenhoven et al., 2016].

- Loan-to-value ratio
- Client age

The variables that are taken into account may differ for different kinds of prepayments. In section 6.6, the predicted influence of variables on the CPR will be computed.

2.2.3 Multinomial logistic regression

A basic model that can be used for predicting behavior based on multiple variables is the so called *multinomial logistic regression model.* It assumes several explanatory variables that are combined to predict the probability that a prepayment of a certain type is made on a monthly basis.

The values of the explanatory variables are contained in row vector **x**. To compute a variable's relative influence, a vector $\boldsymbol{\beta}_j$ contains values that indicate importance for prepayment type j. Not every variable is necessarily used in the prediction of a specific type of prepayment. Entries of $\boldsymbol{\beta}_j$ corresponding to variables that are not taken into account for prepayment type j can be set to zero.
The prediction for the probability of a certain prepayment j is computed by:

$$\mathbb{P}_j(\mathbf{x}) = \frac{e^{\mathbf{x}\boldsymbol{\beta}_j}}{1 + \sum_{k=1}^{6} e^{\mathbf{x}\boldsymbol{\beta}_k}}, \tag{2.1}$$

for $j = 1, \ldots, 6$. The subscript j represents the prepayment types defined in section 2.2.1.
Also the index k of the sum in the denominator runs over the six types of prepayment. The elements of $\boldsymbol{\beta}_j$ are not necessarily constant but may depend on the input values contained in **x**. This way the relative contribution to the predicted CPR is not constant for different values of a variable.

The prediction for the probability that no prepayment event occurs in a certain month equals

$$\mathbb{P}_7(\mathbf{x}) = \frac{1}{1 + \sum_{k=1}^{6} e^{\mathbf{x}\boldsymbol{\beta}_k}}. \tag{2.2}$$

In fact, the Reconsider option can only be exercised at specific loan periods. This changes the expressions (2.1) and (2.2) slightly. A detailed explanation of the model can be found in Kuijenhoven et al. [2016].

Chapter 3

Feedforward artificial neural networks

Neural networks require an input that is processed by a network of connecting nodes (called *neurons*) to produce an output. The input is represented by the values of the input neurons and the output by the values of the output neurons. Input variables are often called *features*. The process of generating an output based on the presented features is done by the neurons in between. These are called the *hidden neurons*. Figure 3.1 shows a schematic example of a neural network[1]. The neurons are ordered in so-called *layers*.

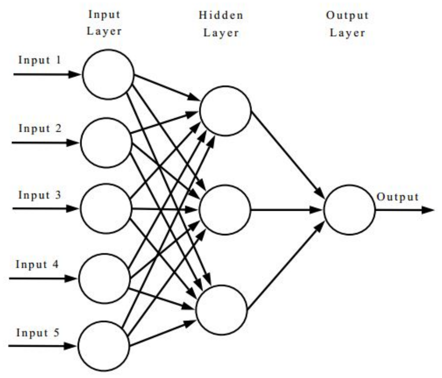


Figure 3.1: An example of a neural network with five input neurons, one hidden layer consisting of three neurons and one output neuron.

In this network, the neurons are influenced by neurons from previous layers only. This kind of network is called a *feedforward neural network* and contains no cycles. It is the most common type of network.

[1]Figure source: https://tex.stackexchange.com/questions/132444/diagram-of-an-artificial-neural-network.

The information entering the network represents the features (in this case five) influencing the output. These values act as an input for the hidden layer. Each of the neurons in the hidden layer transforms the information received as input. The values from the hidden layers are in turn passed to the output layer where their linear combination acts as model output.
It is possible for the network to consist of several hidden layers. The process is then repeated.

In this chapter we will first describe the computations that are made within the neurons. Then the process of training a network to generate useful outputs is explained.

3.1 Computations within a neural network

A neural network takes an input vector and produces an output. In our case, the input vector represents the variables that influence the prepayment behavior. Examples of input variables were given in section 2.2.2. The inputs that are selected depend on the prepayment type that is treated.

Figure 3.2 schematically shows the information flow within a neuron[2]. The input is processed by a weighted sum plus a bias term first. The weights and biases that are used are in general different for each neuron in the hidden layer. This way, each hidden neuron processes the input information differently.
In turn, these linear combinations act as an input for the *activation function* to compute the output of the neurons in the hidden layer.

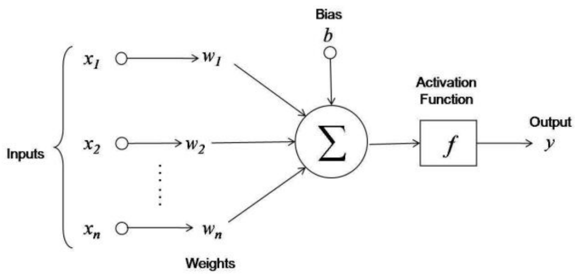


Figure 3.2: Schematic representation of the information flow within a neuron.

Let us denote by d_l the number of neurons in the l-th layer of the network. We denote the input layer, that contains the values for the model input, as layer 0. x_i for $i \in \{0, \ldots, d_0\}$ represents the value of the i-th input. Then the computation within the j-th neuron in the first hidden layer is given

[2]Figure source: https://tex.stackexchange.com/questions/132444/diagram-of-an-artificial-neural-network.

by:

$$z_j^{(1)} = \sum_{i=1}^{d_0} x_i w_{ij}^{(1)} + b_j^{(1)}, \tag{3.1}$$

where the value $w_{ij}^{(1)}$ for $i = 1, \dots, d_0$ and $j = 1, \dots, d_1$ denotes the weight of neuron j in layer 1 for neuron i from layer 0 (the input layer). $b_j^{(1)}$ for $j = 1, \dots, d_1$ denotes the bias for neuron j in layer 1.

The output of neuron j in layer 1 is

$$a_j^{(1)} = \sigma(z_j^{(1)}), \tag{3.2}$$

where $\sigma(\cdot)$ is the activation function. $a_j^{(1)}$ for $j = 1, \dots, d_1$ is called the activation of neuron j in layer 1. This activation function is usually nonlinear.

Three popular activation functions are the *Sigmoid* function, the *Rectified Linear Unit* (abbreviated: *ReLU*) and the hyperbolic tangent. Their activations are represented by the following functions:

$$\begin{aligned} \text{Sigmoid}(x) &= \frac{1}{1+e^{-x}}, \\ \text{ReLU}(x) &= \max(x, 0), \\ \tanh(x) &= \frac{e^x - e^{-x}}{e^x + e^{-x}}. \end{aligned} \tag{3.3}$$

These functions will be used in chapter 5.

We can generalize the expressions (3.1) and (3.2) for a neural network with L layers by writing values of the neurons in a layer together as row vectors. The input is a row vector **x** of size d_0, the activation of the first layer is a row vector $\mathbf{a}^{(1)}$ of size d_1, etc.
The output, which is assigned layer number L, is of size d_L.
The activation of layer $l \in 1, \dots, L-1$ can then be written as:

$$\mathbf{a}^{(l)} = \sigma(\mathbf{z}^{(l)}) = \sigma(\mathbf{a}^{(l-1)} W^{(l)} + \mathbf{b}^{(l)}), \tag{3.4}$$

where the $d_{l-1} \times d_l$ matrix $W^{(l)}$ contains the weights and $\mathbf{b}^{(l)} \in \mathbb{R}^{d_l}$ the biases of layer l. For simplicity the activation function is chosen to be the same for all layers. In this notation, the input vector **x** is denoted by $\mathbf{a}^{(0)}$.
The expression (3.4) for the output layer $l = L$ is the same up to the activation function $\sigma(\cdot)$. At the output layer, the likelihood is computed instead of applying the activation function. The shape of the likelihood is treated in section 3.2.1.

3.2 Training a neural network

When building a neural network model, initial values for the weights and biases are drawn randomly. The network is shaped during the training process by adjusting the weights and biases. To determine how to optimally change the weights and biases, available data pairs $(\mathbf{x}_i, \mathbf{y}_i)$ are used. The model features are represented by $\mathbf{x}_i \in \mathbb{R}^{d_0}$ and the observed output values, often called *targets*, by $\mathbf{y}_i \in \mathbb{R}^{d_L}$. A dataset (X, Y) consisting of pairs $(\mathbf{x}_i, \mathbf{y}_i)$ is typically split into three disjunct subsets: a *train* set, a *test* set and a *validation* set. The first two are respectively used for training the network and testing its performance. The validation set is used to monitor the performance during training (discussed further in section 3.2.4).

By training the network we wish to learn a function $\mathbf{f} : \mathbb{R}^{d_0} \rightarrow \mathbb{R}^{d_L}$ that is likely to have generated the instances in the train set. This function represents the relationship between input variables and corresponding targets. The computations of the network solely depend on the weights and biases that are used in the layers to pass the information. Therefore, to find such a function **f**, the weights and biases are updated iteratively to improve the performance of the model.

The most common and intuitive method for training an artificial neural network is the *error back propagation algorithm* [Rumelhart et al., 1986]. The structure of the back propagation algorithm consists of the following steps [Gardner and Dorling, 1998]:

1. Initialize network weights,
2. Present input vector(s) from the train set to the network[3],
3. Propagate the input vector(s) through the network to obtain the output(s),
4. Calculate the error signal by comparing model output(s) to the observed (target) output(s),
5. Propagate error signal back through the network,
6. Adjust weights and biases using an optimizer to reduce the overall error,
7. Repeat Steps $2 - 6$ until the overall error is satisfactorily small.

An iteration of Steps $2 - 6$ is called an *epoch*.
After training, the test set is used to evaluate the performance of the trained neural network.

The steps of the error back propagation algorithm will be discussed below.

[3]The features are often presented in groups, so-called *batches*.

3.2.1 Information processing (Steps 1-3)

We consider a network with L layers. Such a network consists of one input layer 0 with d_0 neurons, $L-1$ hidden layers $l \in \{1, \dots, L-1\}$ with d_l neurons each and an output layer L with d_L neurons. We already introduced the notation for the activation of a neuron in layer l:

$$\mathbf{a}^{(l)} = \sigma(\mathbf{z}^{(l)}) = \sigma(\mathbf{a}^{(l-1)} W^{(l)} + \mathbf{b}^{(l)}), \tag{3.5}$$

for $l \in \{1, \dots, L-1\}$. The elements $W^{(l)}$ and $\mathbf{b}^{(l)}$, for $l = 1, \dots, L$, are respectively a matrix and a vector of appropriate shape with initially randomly distributed elements. A normal distribution is often used to sample the initial weights and biases. Let us denote the collection of all weight matrices $W^{(l)}$ and bias vectors $\mathbf{b}^{(l)}$ by ω. The function $\mathbf{f}$ is fully determined by this collection of parameters ω.

Recall that no activation function is applied at the output layer. There, a weighted sum of activations in layer $L-1$ is taken and a bias is added:

$$\mathbf{f}(\mathbf{x}_i|\omega) = \mathbf{z}^{(L)} = \mathbf{a}^{(L-1)} W^{(L)} + \mathbf{b}^{(L)}. \tag{3.6}$$

This output in $\mathbb{R}^{d_L}$ is used to compute the likelihood of the prediction. How this likelihood is computed depends on whether a regression or classification problem is treated.

For a regression task the elements of $\mathbf{f}(\mathbf{x}_i|\omega)$ represent the prediction. The likelihood is often assumed to be a Gaussian:

$$p(\mathbf{y}_i|\mathbf{x}_i, \omega) = \mathcal{N}(\mathbf{f}(\mathbf{x}_i|\omega), \tau^{-1} I), \tag{3.7}$$

where τ^{-1} represents the observation noise.

When we consider a classification, the elements of the output have to add up to one. A popular output function that is used in classifications is the *softmax* function:

$$p(\mathbf{y}_i|\mathbf{x}_i, \omega) = \text{softmax}(\mathbf{f}(\mathbf{x}_i|\omega)) = \frac{e^{\mathbf{f}(\mathbf{x}_i|\omega)}}{\sum_{j=1}^{d_L} e^{\mathbf{f}(\mathbf{x}_i|\omega)_j}}. \tag{3.8}$$

The j-th element of $p(\mathbf{y}_i|\mathbf{x}_i, \omega)$ represents the predicted probability for corresponding event j.

Initially, a neural network does not generate useful information since the weights and biases are chosen at random. Using a dataset (X, Y) however, the network can be trained. Assume that we have a train set that consists of N vectors $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\} \subset X$ that provide the input of the model. Also it contains N target vectors $\mathcal{Y} = \{\mathbf{y}_1, \dots, \mathbf{y}_N\} \subset Y$ which represent the observations for the corresponding inputs $\mathbf{x}_i$. The input vectors are presented and processed by the neural network.

Evidently, $\mathbf{x}_i$ and $\mathbf{y}_i$ for $i = 1, \ldots, N$ are elements in $\mathbb{R}^{d_0}$ and $\mathbb{R}^{d_L}$ respectively. An input vector $\mathbf{x}_i$ is passed through the network to generate an output in $\mathbb{R}^{d_L}$. This output can be compared to the corresponding observation $\mathbf{y}_i$.

3.2.2 Loss function (Step 4)

To measure the performance of the model, we need to define a loss function that quantifies the error of the model. The loss function $E(\cdot)$ depends on the weights and biases of the network. Recall that by ω, we denote the collection of weights $W^{(l)}$ and biases $\mathbf{b}^{(l)}$ for $l = 1, \ldots, L$. A forward pass of an input vector through the network results in an output vector $\mathbf{f}(\mathbf{x}_i|\omega)$. The loss on the train set $(\mathcal{X}, \mathcal{Y})$ containing N observations is defined as:

$$E(\mathcal{X}, \mathcal{Y}|\omega) = \frac{1}{N}\sum_{i=1}^{N} E(\mathbf{x}_i, \mathbf{y}_i|\omega). \tag{3.9}$$

Here, $E(\mathbf{x}_i, \mathbf{y}_i|\omega)$ is the loss made on the i-th instance of the train set. Note that the loss on a set and the loss per instance are both referred to as $E(\cdot)$. The definition of the loss function for each instance depends on the type of problem that is dealt with.

For a regression problem, the *Euclidean loss* is often used:

$$E_{regr}(\mathbf{x}_i, \mathbf{y}_i|\omega) = \frac{1}{2}||\mathbf{y}_i - \mathbf{f}(\mathbf{x}_i|\omega)||^2, \tag{3.10}$$

where $|| \cdot ||$ is the Euclidean distance. Note that this loss function $E(\mathcal{X}, \mathcal{Y}|\omega)$ is equal to the mean squared error (MSE) of the instances scaled by $\frac{1}{2}$.

When a classification problem is treated, the target vector $\mathbf{y}$ representing the observed class has the form of a binary vector with a single 1 on the position corresponding to the observed class. A well known loss function for classification problems is the *cross entropy loss* (*CE loss*), i.e.:

$$E_{class}(\mathbf{x}_i, \mathbf{y}_i|\omega) = -\sum_{j=1}^{d_L} y_j \log \text{softmax}(\mathbf{f}(\mathbf{x}_i|\omega))_j. \tag{3.11}$$

Since it is closely related to the log-likelihood, the cross entropy loss is often used in rare event problems where events that occur with a low probability have to be predicted [Asmussen et al., 2003] [De Boer et al., 2005].

3.2.3 Adjusting the weights and biases (Steps 5-6)

The process of adjusting the weights and biases consists of two steps. Firstly, the directions in which the weights and biases are updated optimally are determined. Secondly, the weights and biases are

updated in the desired directions.
The loss function is used by a so-called *optimizer* to adjust the weights and biases of the network in such a way that the model is improved (i.e. the error as computed by the loss function decreases). Since, for $\mathcal{X}$ and $\mathcal{Y}$ known, the loss is a function of ω only. The weights and biases solely determine the loss.

The most popular technique to determine the directions used for updating the model parameters is the *gradient descent optimizer* [Rumelhart et al., 1986]. This method requires the activation function to be differentiable. It adjusts the weights and biases in the direction in which they cause a maximal decrease of the loss function. In order to achieve optimal adjustments, the gradient of the loss with respect to each model parameter is used.
This gives rise to the following update schemes:

$$\begin{aligned} W^{(l)} &\to W^{(l)} - \eta \nabla_{W^{(l)}} E, \\ \mathbf{b}^{(l)} &\to \mathbf{b}^{(l)} - \eta \nabla_{\mathbf{b}^{(l)}} E, \end{aligned} \tag{3.12}$$

for $l = 1, \ldots, L$. The value $\eta \in \mathbb{R}$ is a step size, called the *learning rate*, to scale the weight and bias adjustments. There are other kinds optimizers such as the popular *Adam optimizer* that can be used for updating the network parameters [Kingma and Ba, 2014].

3.2.4 Stop when error is sufficiently small (Step 7)

With each iteration, the network is adjusted such that its performance on the instances contained in the train set increases. When this process of learning relationships for mortgages with certain characteristics (those in the train set) continues, the neural network is vulnerable to *overfitting*. This is the phenomenon where the network represents the dataset by which it is trained more than can be justified. The output then captures too much noise in the train set. The performance of a model on unknown data drops when it is overfitting.

In order to prevent overfitting, performance of the model is often not only measured on the train set but also on the validation set which is disjunct from the train set. The instances in the validation set represent the unknown data which should be explained as good as possible by the trained neural network.
During training, the losses on both the training and validation set initially decrease. After a certain amount of epochs, the network starts focusing too much on observations in the train set which causes the error on the validation set to increase. The performance on the validation set is best just before the increase. This is usually the moment when training is stopped. Stopping the process of training the

network at this point is called *early stopping* [Bishop, 2006].

It may occur that the error on the validation set increases for certain epochs. It is not desirable to break the training process immediately in that case. As a solution, a *patience parameter* can be introduced [Fahlman and Lebiere, 1990].
If this parameter is set to a value $k \in \mathbb{N}$, the process is stopped at the n-th epoch if the error on the validation set has not decreased since epoch $n - k$. In that case, no significant increase in explanation of the validation set is attained during the last k epochs.

There are more methods to help prevent overfitting. Two of them, dropout and a regularized loss function, will be discussed in chapter 4.

Chapter 4

Dropout as a method to extract model uncertainty

Gal and Ghahramani [2015a] discuss the lack of ability of most machine learning methods to quantify the uncertainty of a prediction. A method called *dropout* is suggested to solve this problem. It can be shown that sampling from a neural network trained with dropout applied to its layers is equivalent to sampling from the posterior of a *Bayesian neural network*.

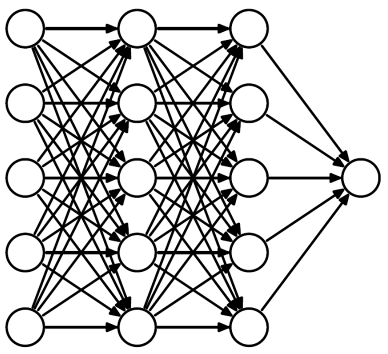
In this chapter, we will first introduce the concept of dropout. Then we will give a proof that connects a neural network trained with dropout to a Bayesian neural network. This relationship will be used later to allow us to generate uncertainty estimates for predictions. The chapter is concluded with a discussion of the necessary assumptions and the hyperparameters in the connection between dropout and Bayesian neural networks.

4.1 Dropout method description

As mentioned in section 3.2.4, when a network is trained it may interpret noise in the training data as existing relationships. This is called overfitting. Dropout is a method that has initially been suggested to prevent overfitting and has become very popular [Hinton et al., 2012] [Srivastava et al., 2014]. We will see that, in addition to reducing overfitting, it can be used to extract uncertainty information from a neural network.

Applying dropout to a neural network implies dropping out neurons at random in every layer l with a probability $1 - p_l$. The neurons are active with probability p_l which we will call the *neuron activity rate* of layer l. The term 'dropout' refers to the dropping out of neurons which results in a thinned

network.
To realize the dropout, independent Bernoulli samples are drawn with each iteration. By using these samples for dropping out neurons, a random subnetwork is trained every time. The collection of drawn Bernoulli variables that jointly represent which neurons are active and which are not is called a *dropout mask*.
In figure 4.1 the dropping out of neurons is shown schematically.


(a) A neural network with all of its neurons active.

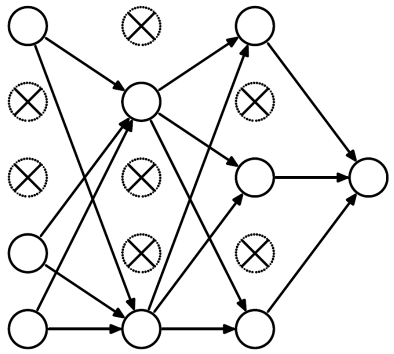

(b) A neural network with some of its neurons dropped out.

Figure 4.1: A schematic visualization of dropout [Srivastava et al., 2014].

Application of the dropout method changes the representation of computations within neurons as is stated in equation (3.4). The dropping out of neurons in layer $l-1$ can mathematically be represented by a binary vector $\mathbf{v}^{(l)}$ of length d_{l-1}.
The vector $\mathbf{v}^{(l)}$ drops out neurons from layer $l-1$ by the multiplication: $\text{diag}(\mathbf{v}^{(l)})W^{(l)}$. Here $\text{diag}(\cdot)$ is the operation that turns a vector in a square matrix of appropriate size with the vector's elements on the main diagonal.
The elements of the vector $\mathbf{v}^{(l)}$ are distributed according to a Bernoulli distribution with parameter p_{l-1}: $v_i^{(l)} = \text{Bernoulli}(p_{l-1})$ for $i = 1, \ldots, d_{l-1}$[1]. p_{l-1} is the fraction of neurons that is active in layer $l-1$. The activation of the neurons in layer l of the network can be written as:

$$\begin{aligned} \mathbf{a}^{(l)}_{dropout} = \sigma(\mathbf{z}^{(l)}_{dropout}) &= \sigma(\mathbf{a}^{(l-1)}\text{diag}(\mathbf{v}^{(l)})W^{(l)} + \mathbf{b}^{(l)}) \\ &= \sigma((\mathbf{a}^{(l-1)} \circ \mathbf{v}^{(l)})W^{(l)} + \mathbf{b}^{(l)}), \end{aligned} \tag{4.1}$$

for $l = 1, \ldots, L-1$. The symbol $\circ$ represents the element-wise multiplication of vectors (*Hadamard* product). Recall that, in this notation, $\mathbf{a}^{(0)}$ represents the input vector .

[1]Note that the vector $\mathbf{v}^{(l)}$ is an element in $\mathbb{R}^{d_{l-1}}$ and is generated using p_{l-1} since it concerns dropping out neurons in layer $l-1$. The superscript l is chosen since it matches the superscript of $W^{(l)}$, the $d_{l-1} \times d_l$ weight matrix with which it is multiplied.

For the output layer $l = L$, the activation function in expression (4.1) is substituted by a likelihood function.

The network is trained with, on average, a fraction p_l of its neurons active in the l-th layer every iteration. To use an approximating model of all the thinned networks that have been trained, the full network is used during testing. The weight matrices $W^{(l)}$ in the full network are multiplied by the appropriate factor $\frac{1}{p_l}$ to correct for the dropped out neurons during training [Srivastava et al., 2014].

Recall from section 3.2.2 that the average loss of a model on a dataset (X, Y) consisting of N observations can be denoted by

$$E(X, Y|\omega) = \frac{1}{N}\sum_{i=1}^{N} E(\mathbf{x}_i, \mathbf{y}_i|\omega), \tag{4.2}$$

for features $\mathbf{x}_i$ and observations $\mathbf{y}_i$.

A *regularization* term will be added to the optimization objective. The L_2 regularization $||\cdot||$ weighted by some coefficients $\lambda_W^{(l)}, \lambda_b^{(l)} \in \mathbb{R}_{\geq 0}$ for respectively the weights and biases of layer l can be used. This results in the following minimization objective [Gal and Ghahramani, 2016]:

$$\mathcal{L}_{dropout}(\omega) := E(X, Y|\omega) + \sum_{l=1}^{L}(\lambda_W^{(l)}||W^{(l)}||^2 + \lambda_b^{(l)}||\mathbf{b}^{(l)}||^2), \tag{4.3}$$

for a dataset (X, Y) containing N observations. The loss function can be chosen appropriately to the type of problem (regression or classification) as discussed in section 3.2.2. The objective $\mathcal{L}_{dropout}$ is sometimes called the *regularized Euclidean loss* or the *regularized cross entropy* [Goodfellow et al., 2016]. The regularization coefficients $\lambda_W^{(l)}$ and $\lambda_b^{(l)}$ determine the strictness of the regularization.
Optimization with respect to the regularized loss function (4.3) in general leads to smaller weights and biases than optimization with respect to (4.2). Higher values will only emerge if they significantly decrease the value of the first term. These small values lead to a model that tends to have less complexity and simpler but more powerfull explanations of the data [Nielsen, 2015]. Addition of the regularization terms to the loss function results in less risk of overfitting.

4.2 Dropout as an approximator for a Bayesian neural network

We will see that the optimization objective $\mathcal{L}_{dropout}$ in (4.3) for neural networks with dropout arises in the expression for optimization of the *variational distribution* for a Bayesian neural network as well. In order to prove this, we will use the concept of *variational inference* (*VI*).

4.2.1 Variational inference

Assume we wish to make a model represented by function $\mathbf{f} : \mathbb{R}^{d_0} \rightarrow \mathbb{R}^{d_L}$ that explains a dataset (X, Y). Let us condition the model on a set of random variables which we will denote by ω. Then we have that the model depends solely on ω such that it is a sufficient statistic for **f**. This is the model's interpretation of the relationship between the features X and the observations Y in the train set. Since ω completely determines the function **f**, it is sufficient to find the parameters ω such that the data is explained well.

In a Bayesian neural network, a prior distribution $p(\omega)$ is defined over the model parameters ω. From there, distributions over the outputs can be computed. One is able to reason about the confidence in the model's predictions using these distributions.
We are interested in the posterior distribution $p(\omega|X, Y)$ over the space of parameters ω. Using Bayes' theorem, the distribution that most likely captures the training data (X, Y) is

$$p(\omega|X, Y) \propto p(Y|X, \omega)p(\omega). \tag{4.4}$$

The likelihood $p(Y|X, \omega)$ is defined in section 3.2.1. Its form depends on whether a regression problem (expression (3.7)) or classification problem (expression (3.8)) is treated.
Using Bayes and our belief about the distribution of the parameters ω gives a predictive distribution for a test instance $\mathbf{x}_i$

$$p(\mathbf{y}_i|\mathbf{x}_i, X, Y) = \int p(\mathbf{y}_i|\mathbf{x}_i, \omega)p(\omega|X, Y)d\omega. \tag{4.5}$$

Usually, the posterior distribution $p(\omega|X, Y)$ can not be evaluated analytically. Therefore, an approximating variational distribution $q_\theta(\omega)$ is defined which is easier to evaluate.

We would like the variational distribution $q_\theta(\omega)$, parametrized by a deterministic set θ, to approximate the posterior $p(\omega|X, Y)$ as good as possible such that

$$p(\mathbf{y}_i|\mathbf{x}_i, X, Y) \approx \int p(\mathbf{y}_i|\mathbf{x}_i, \omega)q_\theta(\omega)d\omega. \tag{4.6}$$

This expression holds for variational distributions $q_\theta(\omega)$ that are close to the posterior distribution $p(\omega|X, Y)$.

To evaluate if this is indeed the case, we use the *Kullback-Leibler divergence* (KL divergence) [Kullback and Leibler, 1951]:

$$\text{KL}(q_\theta(\omega)||p(\omega|X, Y)), \tag{4.7}$$

which is a measure of similarity between the two distributions. Distributions that are very much alike have a low KL divergence.

The KL divergence between distributions P and Q defined on $\mathbb{R}$ is defined as:

$$\text{KL}(Q||P) := \int_{-\infty}^{\infty} Q(x)\log\Big(\frac{Q(x)}{P(x)}\Big)dx. \tag{4.8}$$

In our case, the integral does not run over $\mathbb{R}$ but over the model parameters ω.

Minimizing the KL divergence is equivalent to minimizing the negative *log evidence lower bound* with respect to $q_\theta(\omega)$ [Bishop, 2006] [Gal and Ghahramani, 2016]. The negative log evidence lower bound equals

$$\mathcal{L}_{VI}(\theta) := -\int q_\theta(\omega)\log p(\mathbf{y}_i|\mathbf{x}_i,\omega)d\omega + \text{KL}(q_\theta(\omega)||p(\omega)), \tag{4.9}$$

which we wish to minimize. The variational distribution $q_\theta(\omega)$ that minimizes $\mathcal{L}_{VI}$ is close to $p(\omega|X,Y)$ while still being close to the prior distribution $p(\omega)$ to prevent overfitting [Gal and Ghahramani, 2015b]. The first term in equation (4.9) ensures a good explanation of the data. The second term prevents the network from learning too complex solutions by preferring variational distributions close to the prior distribution. This procedure is known as variational inference.

In contrast to many machine learning methods, this optimization approach considers distributions instead of point estimates. This will become more clear in section 4.2.3 when we set the approximating distribution $q_\theta(\omega) := \delta(\omega - \theta)$, where $\delta(\cdot)$ is the Dirac delta function.

Consider an index set $S \subseteq \{1,\dots,N\}$ representing M feature-observation pairs drawn from a dataset (X,Y) of N observations. The minimization objective $\mathcal{L}_{VI}$ for this subset is:

$$\mathcal{L}_{VI}(\theta) := -\frac{N}{M}\sum_{i\in S}\int q_\theta(\omega)\log p(\mathbf{y}_i|\mathbf{x}_i,\omega)d\omega + \text{KL}(q_\theta(\omega)||p(\omega)). \tag{4.10}$$

Using a random subset of M observations is computationally more efficient and is called *data subsampling* [Hoffman et al., 2013]. The factor $\frac{N}{M}$ scales the first term of the expression for the index set S of size M to the total set size N.

Now we will introduce a method for sampling approximations of the elements of ω by defining a way to parametrize them using the deterministic set θ of model parameters and random variables $v_i^{(l)} \in \mathbb{R}$ for $i = 1,\dots,d_{l-1}$ and $l = 1,\dots,L$. The $v_i^{(l)}$ are random variables that can be sampled without the need for other parameters. In section 4.2.2, when we will treat optimizations of neural networks with dropout, the elements $v_i^{(l)}$ will be assumed to be independent Bernoulli samples.

For now we will discuss the distributions over the weights, the biases can be parametrized in a similar

way. By $\theta_i^{(l)}$ we will denote the deterministic row vector in $\mathbb{R}^{d_l}$ used to parametrize $\mathbf{w}_i^{(l)}$, the i-th row of weight matrix $W^{(l)}$. In section 4.2.2, the elements $\theta_i^{(l)}$ are obtained by optimization of the neural network.
Optimizing over the single weights leads to neglecting weight correlations [Gal, 2016]. In order to avoid this, Barber and Bishop [1998] suggested modelling weight correlations to capture strong posterior parameter correlations. We will follow this method and treat distributions for the rows of weight matrices. The parametrization of the variational distribution $q_{\theta_i^{(l)}}(\mathbf{w}_i^{(l)})$ will be written as $\mathbf{w}_i^{(l)} = g(\theta_i^{(l)}, v_i^{(l)})$ for $i = 1, \dots, d_{l-1}$ and $l = 1, \dots, L$ with parameters $\theta_i^{(l)} \in \mathbb{R}^{d_{l-1}}$ and $v_i^{(l)} \in \mathbb{R}$.

Collecting the parametrizations of random variables gives rise to the generalized notation $p(\mathbf{v}) = \prod_{l,i} p(v_i^{(l)})$ and $\omega = g(\theta, \mathbf{v})$.

This parametrization of ω in the variational distribution $q_\theta(\omega)$ can be used to rewrite expression (4.10) in terms of θ and $\mathbf{v}$:

$$\hat{\mathcal{L}}_{VI}(\theta) := -\frac{N}{M} \sum_{i \in S} \int p(\mathbf{v}) \log p(\mathbf{y}_i | \mathbf{x}_i, g(\theta, \mathbf{v})) d\mathbf{v} + \mathrm{KL}(q_\theta(\omega) || p(\omega)). \tag{4.11}$$

We introduced the parametrization $g(\theta, \mathbf{v})$ such that we can generate samples of ω solely by drawing samples[2] $\hat{\mathbf{v}}_i \sim p(\mathbf{v})$.
The expected log-likelihood terms can be replaced by their stochastic estimators to obtain an M-term Monte Carlo (MC) estimator for the minimization objective (4.11) for the variational distribution:

$$\hat{\mathcal{L}}_{MC}(\theta) := -\frac{N}{M} \sum_{i \in S} \log p(\mathbf{y}_i | \mathbf{x}_i, g(\theta, \hat{\mathbf{v}}_i)) + \mathrm{KL}(q_\theta(\omega) || p(\omega)). \tag{4.12}$$

The minimization objectives satisfy the relation $\mathbb{E}_{\mathbf{v}}[\hat{\mathcal{L}}_{MC}(\theta)] = \hat{\mathcal{L}}_{VI}(\theta)$ [Gal, 2016].

The derivative with respect to θ of the Monte Carlo estimator is

$$\frac{\partial}{\partial \theta} \hat{\mathcal{L}}_{MC}(\theta) = -\frac{N}{M} \sum_{i \in S} \frac{\partial}{\partial \theta} \log p(\mathbf{y}_i | \mathbf{x}_i, g(\theta, \hat{\mathbf{v}}_i)) + \frac{\partial}{\partial \theta} \mathrm{KL}(q_\theta(\omega) || p(\omega)). \tag{4.13}$$

It is important to note that minimization of $\hat{\mathcal{L}}_{MC}(\theta)$ with respect to θ is equivalent to minimizing $\mathcal{L}_{VI}(\theta)$. The minimization of $\hat{\mathcal{L}}_{MC}(\theta)$, and thus minimization of $\mathcal{L}_{VI}(\theta)$, is attained by following algorithm 1.

[2]The subscript i in $\hat{\mathbf{v}}_i$ should not be confused with the i in $v_i^{(l)}$. In $\hat{\mathbf{v}}_i$ it corresponds to the specific observation pair with which the sampled random variable is used. In $v_i^{(l)}$ it corresponds to the random variable for parametrizing the i-th row of weight matrix $W^{(l)}$.

Algorithm 1 Minimize KL divergence between variational distribution $q_\theta(\omega)$ and posterior $p(\omega|X, Y)$

```
1: Given dataset $(X, Y)$ and learning rate $\eta$
2: Randomly initialize parameters $\theta$
3: until $\theta$ has converged do
4:     Draw random subset $S$ of size $M$ from $\{1, \ldots, N\}$
5:     Sample $M$ realizations of the random variables $\hat{\mathbf{v}}_i \sim p(\mathbf{v})$
6:     Calculate derivative of minimization objective w.r.t. $\theta$:
           $\hat{\Delta\theta} \leftarrow -\frac{N}{M} \sum_{i \in S} \frac{\partial}{\partial \theta} \log p(\mathbf{y}_i|\mathbf{x}_i, g(\theta, \hat{\mathbf{v}}_i)) + \frac{\partial}{\partial \theta} \mathrm{KL}(q_\theta(\omega)||p(\omega))$    (4.14)
7:     Update $\theta$ with learning rate $\eta$:
           $\theta \leftarrow \theta + \eta \hat{\Delta\theta}$
```

4.2.2 Optimizing a neural network with dropout

To compare the variational inference to dropout, we will rewrite the optimization objective with regularization terms $\mathcal{L}_{dropout}$ defined in expression (4.3).
For notational simplicity we will focus on a neural network with one hidden layer on which dropout is applied. Extension to arbitrarily deep architectures is easily obtained by generalization to larger values L. Assume no bias is added in the output layer. The set of sufficient model parameters is $\omega = \{W^{(1)}, W^{(2)}, \mathbf{b}^{(1)}\}$.
With θ, we will denote the deterministic matrices and biases that are adjusted iteratively in the forward and backward passes through the network: $\theta = \{M^{(1)}, M^{(2)}, \mathbf{n}^{(1)}\}$. These are being optimized by training the network.

Recall equation (4.1) with the mathematical representation of dropout:

$$\begin{aligned} \mathbf{a}^{(l)}_{dropout} = \sigma(\mathbf{z}^{(l)}_{dropout}) &= \sigma(\mathbf{a}^{(l-1)} \mathrm{diag}(\mathbf{v}^{(l)}) W^{(l)} + \mathbf{b}^{(l)}) \\ &= \sigma((\mathbf{a}^{(l-1)} \circ \mathbf{v}^{(l)}) W^{(l)} + \mathbf{b}^{(l)}). \end{aligned} \tag{4.15}$$

The elements of $\mathbf{v}^{(l)} \in \mathbb{R}^{d_{l-1}}$ are sampled from a Bernoulli distribution with parameter p_{l-1}.
The output of our model with one hidden layer is computed by

$$\begin{aligned} \mathbf{f}(\mathbf{x}_i|\theta, \mathbf{v}) &= \Big(\sigma((\mathbf{x}_i \circ \mathbf{v}^{(1)}) M^{(1)} + \mathbf{n}^{(1)}) \circ \mathbf{v}^{(2)}\Big) M^{(2)} \\ &= \sigma(\mathbf{x}_i \ \mathrm{diag}(\mathbf{v}^{(1)}) M^{(1)} + \mathbf{n}^{(1)}) \ \mathrm{diag}(\mathbf{v}^{(2)}) M^{(2)}, \end{aligned} \tag{4.16}$$

where the element $\mathrm{diag}(\mathbf{v}^{(l)})$ represents the $d_{l-1} \times d_{l-1}$ diagonal matrix where the element $v^{(l)}_i$ is on position (i, i) for $l = 0, 1$.

diag($\mathbf{v}^{(2)}$) for example, is a square matrix of size $d_1 \times d_1$ where the zeros and ones on the diagonal indicate which of the d_1 neurons in the hidden layer are dropped out and which are active.
For notational simplicity we have denoted $\mathbf{v} = \{\text{diag}(\mathbf{v}^{(1)}), \text{diag}(\mathbf{v}^{(2)})\}$ in the parameters of $\mathbf{f}$ in expression (4.16).

Let us substitute $\hat{W}^{(1)} := \text{diag}(\hat{\mathbf{v}}^{(1)})M^{(1)}$ and $\hat{W}^{(2)} := \text{diag}(\hat{\mathbf{v}}^{(2)})M^{(2)}$ as the realizations of the weight matrix after the dropout mask is sampled. The set of realized parameters is denoted by $\hat{\omega} = \{\hat{W}^{(1)}, \hat{W}^{(2)}, \mathbf{n}^{(1)}\}$. The matrices $\hat{W}^{(1)}$ and $\hat{W}^{(2)}$ have shape $d_0 \times d_1$ and $d_1 \times d_2$ respectively. The vector $\mathbf{n}^{(1)} \in \mathbb{R}^{d_1}$ contains the biases of the first (and only) hidden layer.
A neural network with parameters $\hat{\omega}$ has a dropout mask sampled from $p(\mathbf{v})$. Later we will use this distribution as the parametrization that is used to sample the parameters ω in the approximate variational distribution.

Expression (4.16) for the output can be rewritten for realized dropout masks:

$$\mathbf{f}(\mathbf{x}_i|\theta, \hat{\mathbf{v}}) = \mathbf{f}(\mathbf{x}_i|\hat{\omega}) = \sigma(\mathbf{x}_i\hat{W}^{(1)} + \mathbf{n}^{(1)})\hat{W}^{(2)}. \tag{4.17}$$

The form of the likelihood $p(\mathbf{y}_i|\mathbf{f}(\mathbf{x}_i|\hat{\omega}))$ depends on whether we treat a regression or a classification problem. Both were introduced in section 3.2.1.

Recall that for a regression problem, the likelihood has the form of expression (3.7) where the term τ^{-1} represents the the observations noise. This parameter will be discussed in section 4.4.3.
For a neural network that concerns a classification problem we introduced the softmax classifier in expression (3.8).
These expressions for the output $\mathbf{y}_i$ depending on the realizations $\hat{\omega}$ allow us to rewrite the optimization objective (4.3) for a neural network with dropout in a more convenient form. For an index set S that consists of M observations we have:

$$\hat{\mathcal{L}}_{dropout}(\theta) := \frac{1}{M}\sum_{i\in S} E(\mathbf{x}_i, \mathbf{y}_i|\theta, \hat{\mathbf{v}}_i) + \lambda_W^{(1)}||M^{(1)}||^2 + \lambda_W^{(2)}||M^{(2)}||^2 + \lambda_b^{(1)}||\mathbf{n}^{(1)}||^2. \tag{4.18}$$

Here, $\hat{\mathbf{v}}_i$ in the loss term E denotes a collection of sampled vectors generating the dropout mask used with observation pair $(\mathbf{x}_i, \mathbf{y}_i)$[3].
The expression (4.18) consists of a term that represents the accuracy of the model and a regularization on the model parameters $\theta = \{M^{(1)}, M^{(2)}, \mathbf{n}^{(1)}\}$ based on coefficients $\lambda_W^{(1)}, \lambda_W^{(2)}, \lambda_b^{(1)} \in \mathbb{R}_{\geq 0}$.

[3]The subscript i in $\hat{\mathbf{v}}_i$ should not be confused with the i in $v_i^{(l)}$. In $\hat{\mathbf{v}}_i$ it corresponds to the specific observation pair with which the sampled dropout mask is used. In $v_i^{(l)}$ it corresponds to the i-th neuron in layer $l-1$.

The first term, which represents the accuracy of the model, depends on whether we are dealing with a regression or a classification problem. For a regression problem we use the Euclidean loss that was introduced in section 3.2.2:

$$E_{regr}(\mathbf{x}_i, \mathbf{y}_i|\hat{\omega}) = \frac{1}{2}||\mathbf{y}_i - \mathbf{f}(\mathbf{x}_i|\hat{\omega})||^2. \tag{4.19}$$

From Gal [2016] we know that this loss can be written as the negative log-likelihood plus a constant:

$$E_{regr}(\mathbf{x}_i, \mathbf{y}_i|\hat{\omega}) = -\tau^{-1}\log p(\mathbf{y}_i|\mathbf{f}(\mathbf{x}_i|\hat{\omega})) + \text{constant}. \tag{4.20}$$

Recall the Cross Entropy loss for classification:

$$\begin{aligned} E_{class}(\mathbf{x}_i, \mathbf{y}_i|\hat{\omega}) &= -\sum_{j=1}^{d_L} (y_i)_j \log \text{softmax}(\mathbf{f}(\mathbf{x}_i|\hat{\omega}))_j \\ &= -\sum_{j=1}^{d_L} (y_i)_j \log \frac{(e^{\mathbf{f}(\mathbf{x}_i|\hat{\omega})})_j}{\sum_{k=1}^{d_L} e^{\mathbf{f}_k(\mathbf{x}_i|\hat{\omega})}} \\ &= -\sum_{j=1}^{d_L} (y_i)_j \log p(\mathbf{y}_i|\mathbf{f}(\mathbf{x}_i|\hat{\omega}))_j. \end{aligned} \tag{4.21}$$

Here, $(y_i)_j$ denotes the j-th element of target vector $\mathbf{y}_i$. Note that $\mathbf{y}_i$ is a vector containing zeros except for a single one instance. When we denote the observed class by c, the expression can be simplified:

$$\begin{aligned} E_{class}(\mathbf{x}_i, \mathbf{y}_i|\hat{\omega}) &= -\log \text{softmax}(\mathbf{f}(\mathbf{x}_i|\hat{\omega}))_c \\ &= -\log p(\mathbf{y}_i|\mathbf{f}(\mathbf{x}_i|\hat{\omega}))_c + \text{constant}. \end{aligned} \tag{4.22}$$

Note that for $\tau = 1$, this expression is identical to the loss for regression tasks, E_{regr}. We will use these expressions to rewrite $\mathcal{L}_{dropout}$ for the regularized loss of a neural network with dropout.

Recall that $\mathbf{v}^{(l)}$ is a vector containing d_{l-1} Bernoulli elements with parameter p_{l-1} that represents which neurons are dropped. The trick is that samples can be drawn without the use of other parameters. The realizations of the dropout masks $\hat{\mathbf{v}}^{(l)}$ for $l = 1, \ldots, L$ can be collected in $\hat{\mathbf{v}}$. It contains binary vectors for the layers with elements drawn from the proper Bernoulli distribution.

Using the parametrization $\hat{\omega}_i = g(\theta, \hat{\mathbf{v}}_i)$, expression (4.18) for $\hat{\mathcal{L}}_{dropout}$ can be rewritten as:

$$\hat{\mathcal{L}}_{dropout}(\theta) = -\frac{1}{M\tau}\sum_{i\in S} \log p(\mathbf{y}_i|\mathbf{x}_i, g(\theta, \hat{\mathbf{v}}_i)) + \lambda_W^{(1)}||M^{(1)}||^2 + \lambda_W^{(2)}||M^{(2)}||^2 + \lambda_n^{(1)}||\mathbf{n}^{(1)}||^2 + C, \tag{4.23}$$

where $C \in \mathbb{R}$ is a constant. Note that the likelihood is denoted by $p(\mathbf{y}_i|\mathbf{x}_i, g(\theta, \hat{\mathbf{v}}_i))$. This form allows us to make a connection between the minimization objectives of dropout and variational inference. The derivative of objective (4.23) with respect to the model parameters $\theta = \{M^{(1)}, M^{(2)}, \mathbf{n}^{(1)}\}$ is given by

$$\begin{aligned}\frac{\partial}{\partial\theta}\hat{\mathcal{L}}_{dropout}(\theta) = &-\frac{1}{M\tau}\sum_{i\in S}\frac{\partial}{\partial\theta}\log p(\mathbf{y}_i|\mathbf{x}_i, g(\theta, \hat{\mathbf{v}}_i)) \\ &+ \frac{\partial}{\partial\theta}(\lambda_W^{(1)}||M^{(1)}||^2 + \lambda_W^{(2)}||M^{(2)}||^2 + \lambda_b^{(1)}||\mathbf{n}^{(1)}||^2).\end{aligned} \tag{4.24}$$

This derivative can be used to minimize the loss of a neural network trained with dropout.
In algorithm 2, a method is suggested to attain the minimization of $\hat{\mathcal{L}}_{dropout}(\theta)$ with respect to θ. This method can be generalized for networks consisting of L layers.

Algorithm 2 Optimize neural network with dropout applied

```
1: Given dataset $(X, Y)$ and learning rate $\eta$
2: Randomly initialize parameters $\theta$
3: until $\theta$ has converged do
4:     Draw random subset $S$ of size $M$ from $\{1, \dots, N\}$
5:     Sample $M$ realizations of the random variables $\hat{\mathbf{v}}_i \sim p(\mathbf{v})$
6:     Calculate derivative of minimization objective w.r.t. $\theta$:
       $\hat{\Delta\theta} \leftarrow -\frac{1}{M\tau}\sum_{i\in S}\frac{\partial}{\partial\theta}\log p(\mathbf{y}_i|\mathbf{x}_i, g(\theta, \hat{\mathbf{v}}_i)) + \frac{\partial}{\partial\theta}(\lambda_W^{(1)}||M^{(1)}||^2 + \lambda_W^{(2)}||M^{(2)}||^2 + \lambda_b^{(1)}||\mathbf{n}^{(1)}||^2)$   (4.25)
7:     Update $\theta$ with learning rate $\eta$:
       $\theta \leftarrow \theta + \eta\hat{\Delta\theta}$
```

4.2.3 Connection between variational distribution and dropout

The algorithms 1 and 2, for optimizing approximate inference in a Bayesian neural network and the neural network with dropout respectively, are similar. The first terms in the expressions (4.14) and (4.25) containing the likelihood are equal up to a factor $N\tau$. Deriving the relationship between the second terms of the minimization objective with the KL divergence (expression (4.14)) and with the regularization terms (expression (4.25)) requires deeper investigation.
This section focuses on deriving equivalence of algorithms 1 and 2 under certain definition of the prior parameter distribution in (4.14) and the regularization coefficients in (4.25). This equivalence allows us to reason about confidence regarding predictions from neural networks trained with dropout.

If we define a prior distribution $p(\omega)$ over the parameters ω such that the condition

$$\frac{\partial}{\partial\theta}\mathrm{KL}(q_\theta(\omega)||p(\omega)) = N\tau\frac{\partial}{\partial\theta}(\lambda_W^{(1)}||M^{(1)}||^2 + \lambda_W^{(2)}||M^{(2)}||^2 + \lambda_b^{(1)}||\mathbf{n}^{(1)}||^2), \tag{4.26}$$

holds, then the derivatives of the optimization objectives $\hat{\mathcal{L}}_{MC}$ and $\hat{\mathcal{L}}_{dropout}$ are identical up to a scale $N\tau$, i.e.:

$$\begin{aligned}\frac{\partial}{\partial\theta}\hat{\mathcal{L}}_{MC}(\theta) &= -\frac{\partial}{\partial\theta}\frac{N}{M}\sum_{i\in S}\log p(\mathbf{y}_i|\mathbf{x}_i, g(\theta,\mathbf{v})) + \mathrm{KL}(q_\theta(\omega)|p(\omega))\\ &= -\frac{\partial}{\partial\theta}\frac{N}{M}\sum_{i\in S}\log p(\mathbf{y}_i|\mathbf{x}_i, g(\theta,\mathbf{v}))\\ &\quad + N\tau\frac{\partial}{\partial\theta}(\lambda_W^{(1)}||M^{(1)}||^2 + \lambda_W^{(2)}||M^{(2)}||^2 + \lambda_b^{(1)}||\mathbf{n}^{(1)}||^2)\\ &= N\tau\frac{\partial}{\partial\theta}\hat{\mathcal{L}}_{dropout}(\theta).\end{aligned} \tag{4.27}$$

As a result, the algorithms 1 and 2 give rise to the same optimal model parameters $theta$ under this prior $p(\omega)$.
This means that the optimization of a neural network with dropout with respect to a loss function with regularization terms is equivalent to approximate inference in a probabilistic interpretation of the neural network.

We will show that the condition (4.26) holds when the prior distribution $p(\omega)$ will be a product of uncorrelated Gaussian distributions over each weight and bias:

$$p(\omega) = \prod_{l=1}^{L} p(W^{(l)})p(\mathbf{b}^{(l)}), \tag{4.28}$$

with, for $l = 1, \dots, L$,

$$p(W^{(l)}) = \mathcal{MN}(W^{(l)}; 0, I/(\gamma_W^{(l)})^2, I), \tag{4.29}$$

a $d_{l-1} \times d_l$ matrix Gaussian distribution and

$$p(\mathbf{b}^{(l)}) = \mathcal{N}(\mathbf{b}^{(l)}; 0, I/(\gamma_b^{(l)})^2), \tag{4.30}$$

a vector in $\mathbb{R}^{d_l}$ with Gaussian elements. The terms $\gamma_W^{(l)}, \gamma_b^{(l)} \in \mathbb{R}_{\geq 0}$ scale the width of the distributions. As we will see, these terms are connected with the scalars $\lambda_W^{(l)}$ and $\lambda_b^{(l)}$ in the regularization terms.

The approximating distribution is a dropout variational distribution as discussed in section 4.2.1. The network is considered to consist of L layers. Recall from section 3.2.1 that such a network contains an input layer, $L - 1$ hidden layers and an output layer. We will follow the proof proposed by Gal

[2016].

In this proof, we use a parametrization for the variational distribution:

$$q_\theta(\omega) = \int q_\theta(\omega|\mathbf{v})p(\mathbf{v})d\mathbf{v}, \tag{4.31}$$

where $q_\theta(\omega|\mathbf{v}) = \delta(\omega - g(\theta, \mathbf{v}))$. $\delta(\cdot)$ represents the Dirac delta function that we will later approximate with a very narrow Gaussian distribution. The random variable $\mathbf{v}$ is a factorization over the random variables $\mathbf{v}^{(l)}$ used in the parametrization. Recall that, for dropout, the elements of $\mathbf{v}^{(l)} \in \mathbb{R}^{d_{l-1}}$ are drawn from the Bernoulli distribution with parameter p_{l-1}.

As before, $g(\theta, \mathbf{v}) = \{\text{diag}(\mathbf{v}^{(1)})M^{(1)}, \ldots, \text{diag}(\mathbf{v}^{(L)})M^{(L)}, \mathbf{n}^{(1)}, \ldots, \mathbf{n}^{(L)}\}$ are the weights and biases of the trained network after the dropout mask is applied. $\theta = \{M^{(1)}, \ldots, M^{(L)}, \mathbf{n}^{(1)}, \ldots, \mathbf{n}^{(L)}\}$ represents the optimized model parameters and $\mathbf{v}$ represents the random vectors to realize dropout.

We will treat the KL divergence for rows of the weight matrices. The relation for the biases is obtained similarly. Since $q_\theta(\omega)$ is a factorization over the rows of the weight matrices, the KL divergence can be written as

$$\text{KL}(q_\theta(\omega)||p(\omega)) = \sum_{l,k} \text{KL}\big(q_{\theta_k^{(l)}}(\mathbf{w}_k^{(l)})||p(\mathbf{w}_k^{(l)})\big), \tag{4.32}$$

where l sums over the layers and k over the rows of the weight matrices.

The distribution $q_{\theta_k^{(l)}}(\mathbf{w}_k^{(l)}|\mathbf{v}) = \delta\big(\mathbf{w}_k^{(l)} - g(\theta_k^{(l)}, \mathbf{v}_k^{(l)})\big)$ for each weight row can be approximated by a mixture of Gaussians with small variance $\lambda^2 I = \Lambda$ and one component fixed at zero.

Derivation of $\text{KL}(t(\mathbf{x})||s(\mathbf{x}))$ for general distributions $s(\mathbf{x})$ and $t(\mathbf{x})$

To avoid getting lost in complex notation we will consider two general distributions to rewrite the relationship (4.32). The distributions s and t over the same domain will replace the prior distributions and variational distribution respectively for the moment.

Define a vector $\mathbf{a} = (a_0, \ldots, a_{L-1})$. Also define the number of hidden neurons per layer $d_l \in \mathbb{N}$ for $l = 1, \ldots, L$, the amount of layers $L \in \mathbb{N}$ and a diagonal positive-definite matrix $\Lambda_l \in \mathbb{R}^{d_l \times d_l}$ for $l = 1, \ldots, L$. The elements of Λ_l are independent of d_l. We assume the prior parameter distribution to be $s(\mathbf{x}) \sim \mathcal{N}(0, I)$. Later we will substitute this by a Gaussian with scaled variance.

Let

$$t(\mathbf{x}) = \sum_{l=1}^{L} a_{l-1}\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_l, \Lambda_l), \tag{4.33}$$

with $\boldsymbol{\mu}_l \in \mathbb{R}^{d_l}$ be a mixture of Gaussian distributions. Assume that $\boldsymbol{\mu}_i - \boldsymbol{\mu}_j \sim \mathcal{N}(0, I)$ for all i, j.

The KL divergence between $s(\mathbf{x})$ and $t(\mathbf{x})$ can be rewritten as

$$\begin{aligned}\mathrm{KL}(t(\mathbf{x})||s(\mathbf{x})) &= \int t(\mathbf{x})\log\Big(\frac{t(\mathbf{x})}{s(\mathbf{x})}\Big)d\mathbf{x} \\ &= \int t(\mathbf{x})\log t(\mathbf{x})d\mathbf{x} - \int t(\mathbf{x})\log s(\mathbf{x})d\mathbf{x} \\ &= -\mathcal{H}(t(\mathbf{x})) - \int t(\mathbf{x})\log s(\mathbf{x})d\mathbf{x},\end{aligned} \tag{4.34}$$

where in the last step the substitution $\mathcal{H}(t(\mathbf{x})) = -\int t(\mathbf{x})\log t(\mathbf{x})d\mathbf{x}$ is performed for simplicity. Note the relationship between $\mathcal{H}(\cdot)$ and the cross entropy introduced in section 3.2.2. The function $\mathcal{H}(t(\mathbf{x}))$ is also called the *entropy* of $t(\mathbf{x})$ [Shannon, 1948].
We will rewrite the terms of (4.34) separately to obtain a more useful expression of $\mathrm{KL}(t(\mathbf{x})||s(\mathbf{x}))$.

$\mathcal{H}(t(\mathbf{x}))$ can be rewritten by using a change of variables $\mathbf{x} = \boldsymbol{\mu}_l + Q_l\boldsymbol{\epsilon}_l$ within the logarithm. Here, Q_l is a $d_l \times d_l$ matrix such that $Q_lQ_l^T = \Lambda_l$ which exists since Λ_l is a positive definite diagonal matrix for $l = 1, \dots, L$. $\boldsymbol{\epsilon}_l$ is a standard normal random variable. This change of variables leads to a favorable approximator of $t(\mathbf{x})$ as we will see.
Using the probability density function of a normally distributed random variable we have

$$\begin{aligned}t(\boldsymbol{\mu}_l + Q_l\boldsymbol{\epsilon}_l) &= \sum_{j=1}^{L} a_{j-1}\mathcal{N}(\boldsymbol{\mu}_l + Q_l\boldsymbol{\epsilon}_l; \boldsymbol{\mu}_j, \Lambda_j) \\ &= \sum_{j=1}^{L} a_{j-1}\frac{1}{(2\pi)^{d_j/2}|\Lambda_j|^{1/2}}\exp(-\frac{1}{2}||\boldsymbol{\mu}_l + Q_l\boldsymbol{\epsilon}_l - \boldsymbol{\mu}_j||^2_{\Lambda_j}).\end{aligned} \tag{4.35}$$

Here, $||\cdot||_{\Lambda_j}$ is the *generalized squared interpoint distance* (or *Mahalanobis* distance) [Xiang et al., 2008]. The term $\boldsymbol{\mu}_l + Q_l\boldsymbol{\epsilon}_l - \boldsymbol{\mu}_j$ within this distance operator consists of independent normally distributed elements so it is a Gaussian on itself. Computing the generalized squared interpoint distance gives rise to a Chi-squared distribution with d_l degrees of freedom.

From this, we know that for $j \neq l$, $||\boldsymbol{\mu}_l + Q_l\boldsymbol{\epsilon}_l - \boldsymbol{\mu}_j||^2_{\Lambda_j} \gg 0$ if $d_l \gg 0$. If our layers are sufficiently large, the summation disappears. In that case, expression (4.35) can be simplified to[4]

$$\begin{aligned}t(\boldsymbol{\mu}_l + Q_l\boldsymbol{\epsilon}_l) &\approx a_{l-1}\frac{1}{(2\pi)^{d_l/2}|\Lambda_l|^{1/2}}\exp(-\frac{1}{2}||\boldsymbol{\mu}_l + Q_l\boldsymbol{\epsilon}_l - \boldsymbol{\mu}_l||^2_{\Lambda_l}) \\ &= a_{l-1}\frac{1}{(2\pi)^{d_l/2}|\Lambda_l|^{1/2}}\exp(-\frac{1}{2}\boldsymbol{\epsilon}_l^TQ_l^TQ_l^{-T}Q_l^{-1}Q_l\boldsymbol{\epsilon}_l) \\ &= a_{l-1}\frac{1}{(2\pi)^{d_l/2}|\Lambda_l|^{1/2}}\exp(-\frac{1}{2}\boldsymbol{\epsilon}_l^T\boldsymbol{\epsilon}_l).\end{aligned} \tag{4.36}$$

[4]In the second step we use that $||\boldsymbol{\mu}_l + Q_l\boldsymbol{\epsilon}_l - \boldsymbol{\mu}_l||^2_{\Lambda_l} = ||Q_l\boldsymbol{\epsilon}_l||^2_{\Lambda_l} := \sqrt{(Q_l\boldsymbol{\epsilon}_l)^T\Lambda_l^{-1}(Q_l\boldsymbol{\epsilon}_l)}^2 = \boldsymbol{\epsilon}_l^TQ_l^TQ_l^{-T}Q_l^{-1}Q_l\boldsymbol{\epsilon}_l$.

We will use expressions (4.33) and (4.36) to substitute the instances of $t(\mathbf{x})$ in $\mathcal{H}(t(\mathbf{x}))$. This leads to:

$$\begin{aligned}
\mathcal{H}(t(\mathbf{x})) &= -\int t(\mathbf{x})\log t(\mathbf{x})d\mathbf{x} \\
&= -\int \sum_{l=1}^{L} a_{l-1}\mathcal{N}(\mathbf{x};\boldsymbol{\mu}_l,\Lambda_l)\log t(\mathbf{x})d\mathbf{x} \\
&= -\sum_{l=1}^{L} a_{l-1}\int \mathcal{N}(\boldsymbol{\epsilon}_l;0,I)\log t(\boldsymbol{\mu}_l + Q_l\boldsymbol{\epsilon}_l)d\boldsymbol{\epsilon}_l \\
&\approx -\sum_{l=1}^{L} a_{l-1}\int \mathcal{N}(\boldsymbol{\epsilon}_l;0,I)\log\Big(a_{l-1}\frac{1}{(2\pi)^{d_l/2}|\Lambda_l|^{1/2}}\exp(-\frac{1}{2}\boldsymbol{\epsilon}_l^T\boldsymbol{\epsilon}_l)\Big)d\boldsymbol{\epsilon}_l \qquad (4.37)\\
&= \sum_{l=1}^{L}\frac{a_{l-1}}{2}\Big(\int \mathcal{N}(\boldsymbol{\epsilon}_l;0,I)\boldsymbol{\epsilon}_l^T\boldsymbol{\epsilon}_l d\boldsymbol{\epsilon}_l + \log(|\Lambda_l|) + d_l\log(2\pi)\Big) + \mathcal{H}(\mathbf{a}) \\
&= \sum_{l=1}^{L}\frac{a_{l-1}}{2}\Big(d_l + \log(|\Lambda_l|) + d_l\log(2\pi)\Big) + \mathcal{H}(\mathbf{a}) \\
&= \sum_{l=1}^{L}\frac{a_{l-1}}{2}\Big(\log(|\Lambda_l|) + d_l(\log(2\pi)+1)\Big) + \mathcal{H}(\mathbf{a}).
\end{aligned}$$

Since $\boldsymbol{\epsilon}_l^T\boldsymbol{\epsilon}_l$ is a Chi-squared distributed with d_l degrees of freedom, it has expectation d_l. Therefore the integral vanishes in the second to last step.
The entropy term $\mathcal{H}(\mathbf{a}) = -\sum_{l=1}^{L} a_{l-1}\log(a_{l-1})$ is introduced for notational simplicity.

To rewrite the second term in expression (4.34) for the KL divergence, we will use $s(\mathbf{x}) \sim \mathcal{N}(0, I)$. This derivation is

$$\begin{aligned}
\int t(\mathbf{x})\log s(\mathbf{x})d\mathbf{x} &= \sum_{l=1}^{L} a_{l-1}\int \mathcal{N}(\mathbf{x};\boldsymbol{\mu}_l,\Lambda_l)\log s(\mathbf{x})d\mathbf{x} \\
&= -\sum_{l=1}^{L}\frac{a_{l-1}}{2}\Big(\boldsymbol{\mu}_l^T\boldsymbol{\mu}_l + \mathrm{tr}(\Lambda_l)\Big),
\end{aligned} \qquad (4.38)$$

where $\mathrm{tr}(\Lambda_l)$ is the trace of matrix Λ_l which is defined as the sum of the elements on the main diagonal of the matrix [Gal, 2016][Davis and Dhillon, 2007].

Finally, an approximation of the KL divergence in (4.34) can be rewritten by substituting both terms by the expressions (4.37) and (4.38). This leads to:

$$\mathrm{KL}(t(\mathbf{x})||s(\mathbf{x})) \approx \sum_{l=1}^{L}\frac{a_{l-1}}{2}\Big(\boldsymbol{\mu}_l^T\boldsymbol{\mu}_l + \mathrm{tr}(\Lambda_l) - \log(|\Lambda_l|) - d_l(\log(2\pi)+1)\Big) - \mathcal{H}(\mathbf{a}), \qquad (4.39)$$

for d_l large enough. Earlier we noticed that algorithms 1 and 2 are equivalent if condition (4.26) is satisfied. Then we stated that the condition holds when the prior distribution $p(\omega)$ is a product of uncorrelated Gaussian distributions over each weight. Now we have the tools to prove that statement.

Equivalence of optimization algorithms for specific prior distribution

The result (4.39) can be used to connect both sides of condition (4.26). In order to do this, the distributions $s(\mathbf{x})$ and $t(\mathbf{x})$ that were used above are substituted by the prior distribution and the variational distribution respectively.
We define the prior distribution $p(\omega)$ as in (4.28), (4.29) and (4.30). This is a scaled version of the $s(\mathbf{x})$ that was considered above.

In the steps above, we used $t(\mathbf{x}) = \sum_{l=1}^{L} a_{l-1}\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_l, \Lambda_l)$. Now we will use that the variational distribution, $q_{\theta_k^{(l)}}(\mathbf{w}_k^{(l)}|\mathbf{v}) = \delta\big(\mathbf{w}_k^{(l)} - g(\theta_k^{(l)}, \mathbf{v}_k^{(l)})\big)$, is a mixture of narrow Gaussians. Note that $\mathbf{w}_k^{(l)}$ is the k-th row of $W^{(l)}$ which represents the influence of the k-th neuron in layer $l-1$ on the neurons of layer l.
Since a neuron in layer $l-1$ is active with probability p_{l-1}, we have that $g(\theta_k^{(l)}, \mathbf{v}_k^{(l)})$ is equal to $\theta_k^{(l)}$ with probability p_{l-1}. Then $q_{\theta_k^{(l)}}(\mathbf{w}_k^{(l)}|\mathbf{v})$ is approximated by a Dirac delta function centered at $\theta_k^{(l)}$.
On the other hand, a neuron in layer $l-1$ is dropped out with probability $1-p_{l-1}$. Therefore, $g(\theta_k^{(l)}, \mathbf{v}_k^{(l)})$ is equal to 0 with probability $1-p_{l-1}$. Then $q_{\theta_k^{(l)}}(\mathbf{w}_k^{(l)}|\mathbf{v})$ is approximated by a narrow Gaussian centered at zero.

We will see that this form of the variational distribution leads to the desired relationship between the two algorithms when it substitutes $t(\mathbf{x})$ in expression (4.39).

By $\mathbf{m}_k^{(l)}$, we will denote the k-th row of the optimized weight matrix in layer l of the network. The vector $\mathbf{n}^{(l)}$ represents the optimized biases in the l-th layer.
Using (4.39) and the specific distributions of the prior and posterior, we have for every partial derivative with respect to $\mathbf{m}_k^{(l)}$:

$$\begin{aligned}\frac{\partial}{\partial \mathbf{m}_k^{(l)}}\mathrm{KL}(q_\theta(\omega)||p(\omega)) &= \frac{\partial}{\partial \mathbf{m}_k^{(l)}}\mathrm{KL}(q_{\theta_k^{(l)}}(\mathbf{w}_k^{(l)})||p(\mathbf{w}_k^{(l)})) \\ &\approx \frac{p_{l-1}}{2}\frac{\partial}{\partial \mathbf{m}_k^{(l)}}\gamma_W^{(l)2}\mathbf{m}_k^{(l)T}\mathbf{m}_k^{(l)}.\end{aligned} \tag{4.40}$$

for $l = 1, \ldots, L$.
The biases of layer l are added separately from the dropping out of neurons in layer $l-1$. Their influence is dependent on the neuron activity rate of layer l. This gives rise to the partial derivative

with respect to $\mathbf{n}^{(l)}$:

$$\begin{aligned}\frac{\partial}{\partial \mathbf{n}^{(l)}}\text{KL}(q_\theta(\omega)||p(\omega)) &= \frac{\partial}{\partial \mathbf{n}_k^{(l)}}\text{KL}(q_{\theta^{(l)}}(\mathbf{b}_k^{(l)})||p(\mathbf{b}^{(l)})) \\ &\approx \frac{p_l}{2}\frac{\partial}{\partial \mathbf{n}^{(l)}}\gamma_b^{(l)2}\mathbf{n}^{(l)T}\mathbf{n}^{(l)},\end{aligned} \tag{4.41}$$

for $l = 1, \ldots, L$. Note that this expression requires a definition of p_L which we set equal to 1 since biases are added to the output layer with probability 1.
Combining expressions (4.40) and (4.41) leads to the desired expression:

$$\begin{aligned}\frac{\partial}{\partial\theta}\text{KL}(q_\theta(\omega)||p(\omega)) &\approx \frac{\partial}{\partial\theta}\Big(\sum_{l=1}^{L}(\frac{p_{l-1}\gamma_W^{(l)2}}{2}||M^{(l)}||^2 + \frac{p_l\gamma_b^{(l)2}}{2}||\mathbf{n}^{(l)}||^2)\Big) \\ &= N\tau\frac{\partial}{\partial\theta}\Big(\sum_{l=1}^{L}(\lambda_W^{(l)}||M^{(l)}||^2 + \lambda_b^{(l)}||\mathbf{n}^{(l)}||^2)\Big),\end{aligned} \tag{4.42}$$

for $\lambda_W^{(l)} = \frac{p_{l-1}\gamma_W^{(l)2}}{2N\tau}$ and $\lambda_b^{(l)} = \frac{p_l\gamma_b^{(l)2}}{2N\tau}$. As a result, (4.27) holds and thus the optimization algorithms are equivalent. It can be concluded that a stochastic pass through a trained neural network with dropout applied is equivalent to an approximation of the posterior of a (similar) optimized Bayesian neural network. We can generate approximations of the Bayesian posterior using this probabilistic interpretation of the model.

Some models define regularization just on the weights since the weights connect the input to the output [Goodfellow et al., 2016]. The biases merely shift the output to the right order of magnitude. When one wishes to smoothen sensitivity, weight regularization is sufficient.
In the remainder of this thesis we will use equally wide priors for the weights and biases in all layers l. This width will be denoted by $\gamma^{(l)} = \gamma_W^{(l)} = \gamma_b^{(l)}$ for $l = 1, \ldots, L$.

4.3 Uncertainty assumptions

The uncertainty that can be computed using the method described in section 4.2 holds under two assumptions. These assumptions will be discussed in this section. Both rely on a sufficient amount of neurons in the layers of the neural network.
The width of a neural network was mentioned earlier to connect expressions (4.35) and (4.36).

4.3.1 Location of the variational distributions

The variational distribution $q_{\theta_k^{(l)}}(\mathbf{w}_k^{(l)}|\mathbf{v}) = \delta\big(\mathbf{w}_k^{(l)} - g(\theta_k^{(l)}, \mathbf{v}_k^{(l)})\big)$ is used such that it can be approximated by a mixture of two very narrow Gaussians. The distribution is centered at zero with probability

$1 - p_{l-1}$ and centered at $\theta_k^{(l)}$ with probability p_{l-1}.
Note that the variational distribution can not be chosen as a Bernoulli random variable taking zero and $\theta_k^{(l)}$ with probability $1-p_{l-1}$ and p_{l-1} respectively since that would lead to $\text{KL}(q_{\theta_k^{(l)}}(\mathbf{w}_k^{(l)})||p(\mathbf{w}_k^{(l)})) = \infty$. By the introduction of the Dirac delta function, which is approximated by a mixture of Gaussians, the KL divergence is finite.

The minimization objective (4.14) consists of two components, one for good explanation of the data and one that ensures a well generalized solution. These two components keep each other in balance. The term $\text{KL}(q_{\theta_k^{(l)}}(\mathbf{w}_k^{(l)})||p(\mathbf{w}_k^{(l)}))$ forces generalization by preferring a small KL divergence between the variational distribution and the prior distribution. In order for the objective (4.14) to be small, it is assumed that the data can be explained well with model parameters close to their prior such that $\text{KL}(q_{\theta_k^{(l)}}(\mathbf{w}_k^{(l)})||p(\mathbf{w}_k^{(l)}))$ is small.
This implies that the optimized parameters $\theta_k^{(l)}$ cannot be too large compared to the width of the prior $p(\mathbf{w}_k^{(l)})$ since it would cause the KL divergence to be large. Therefore we assume that the data can be explained with parameters $\theta_k^{(l)}$ being not large compared to the width of the prior $p(\mathbf{w}_k^{(l)})$.
In order to achieve this, a sufficiently wide network is needed. Information can then be jointly computed by neurons such that there is no demand for high values of the model parameters in order to explain the data.

4.3.2 Shape of the variational distributions

The variational distributions based on the optimal parameters $\theta_k^{(l)}$ for $k = 1, \ldots d_l$ and $l = 1, \ldots, L$ determine the distributions over the predictions. The optimal posterior weight distributions which are generated by the optimized Bayesian neural network are not necessarily Gaussians.

In order to explain the data well, the variational distributions should be close to the optimal posterior distributions. It may require many neurons for the data to be explained well by Gaussian weight distributions. Only in that case can the resulting model have reasonable performance using the variational distribution.

4.4 Hyperparameters related to uncertainty

Recall that the equivalence between a neural network trained with dropout and Bayesian neural networks is based on a relationship between the regularization coefficients and some hyperparameters.

Equality (4.27) holds for

$$\lambda_W^{(l)} = \frac{p_{l-1}\gamma_W^{(l)2}}{2N\tau}, \text{ and } \lambda_b^{(l)} = \frac{p_l\gamma_b^{(l)2}}{2N\tau}, \tag{4.43}$$

for $l = 1, \dots, L$. Now we will briefly justify the relationships between the parameters and the regularization coefficients.

4.4.1 Neuron activity rate

The probability p_{l-1} of a neuron in layer $l-1$ being active is positively related to the regularization coefficients of layer l. The lower p_{l-1}, the lower the regularization coefficients. A low value for p_{l-1} stimulates a well generalized solution since it implies dropping out relatively many neurons on average. The low regularization coefficients imply less strict regularization of the weights and biases. The lower regularization coefficients are justified by the strong regularization that is caused by the low value of p_{l-1}.
On the other hand, a high probability p_{l-1} implies that relatively few neurons are dropped out. This encourages a good explanation of the training data. The protection against overfitting is ensured by the higher values of the regularization coefficients.

Hence, the dropout probability and the regularization terms keep each other in balance when it comes to generalizing and fitting the training data.
Optimal values for p_l can be obtained by performing a grid search. This is a state of the art method which aims to obtain optimal hyperparameters for a model by measuring the model performance under different values of the hyperparameters and choosing the best performing model [Bergstra and Bengio, 2012].

4.4.2 Dependence on length-scale

The term $\gamma_W^{(l)}$ is also called the prior length-scale of layer l [Gal, 2016]. It represents the width of the prior distribution over the weights.
Assume we define very strict prior distributions by choosing $\gamma^{(l)}$ large. Then, the model parameters are close to zero and the regularization term is relatively strong. A small value of $\gamma^{(l)}$, which implies a wide prior, gives rise to a relatively weak regularization term. Hence, the strictness of the regularization coefficients is scaled with the width of the prior distribution.
Usually, an optimal value for the prior length-scale is obtained by performing a grid search.

4.4.3 Model precision and number of observations

The noise τ^{-1} arises in the predictive likelihood of a neural network. The parameter τ is also called the *model precision*.
The more noisy a dataset, the smaller the value τ. This gives rise to higher regularization coefficients than an accurate dataset with a high value of τ. A model is more vulnerable to overfitting when it is trained on a noisy dataset. This justifies the higher protection for overfitting by a strong regularization terms in the case of high values of τ^{-1}.

A large train set has a high value N and therefore leads to lower regularization coefficients. In general, large train sets cover the feature domain better. Since such sets are a wider representation of the relationship between the features and observations, they generally give rise to more general interpretations of the train data.
It is therefore justified that a larger train set leads to less strict regularization terms.

Chapter 5

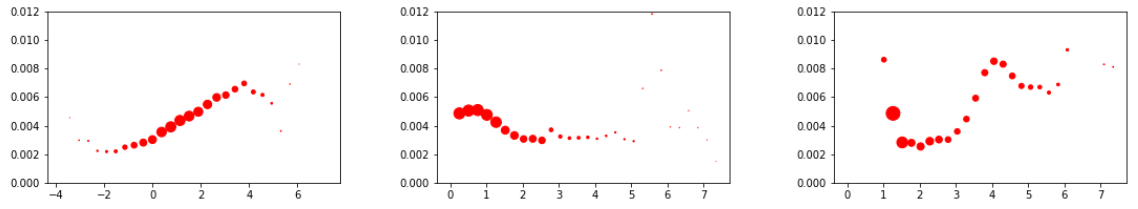
Performance of neural networks under different dataset structures

A neural network is trained iteratively to capture relationships between input variables and observations. In order to train the network, a train set is selected from a dataset containing (historical) feature-observation pairs. The accuracy of the resulting model is dependent on the data which may be noisy or distributed disadvantageously.

We will start this chapter with a discussion of the capability of networks to capture three simple functions under different structures of the train set. The performance of trained neural networks is evaluated under different setups of the train set. Then neural networks with dropout are used to generate uncertainty estimates for predictions. These estimates will be evaluated under the setups to identify regions of the domain where predictions are accurate or not. It will be illustrated that the uncertainty estimates are useful in certain situations. Also, we will discuss a scenario where the uncertainty estimates are unreliable.

5.1 Prepayment feature influence

In section 2.2.2, examples of features that drive prepayment behavior were given. The different features have a specific influence on the prepayment behavior of clients. In figure 5.1, examples of relationships between features and the observed prepayment rate of a specific type are shown. The observations are divided into buckets. Each bucket represents a small interval on the domain of the feature in question.



(a) Interest Rate Incentive (b) Remaining Interest Maturity (c) House Price Index Ratio

Figure 5.1: Different relationships between features and client prepayment behavior.

On the horizontal axis in figure 5.1, values on the domain for the features are shown. The vertical axis represents the observed conditional prepayment rate (CPR) of that bucket. The size of the dots indicates the number of prepayments within the bucket.
The interest rate incentive (figure 5.1a) appears to be positively correlated with the prepayment rate. The effect flattens at both tails. This means that increase (or decrease) of the interest rate incentive does not significantly influence the prepayment rate in the tails.
The remaining interest rate maturity (figure 5.1b) is positively correlated with the prepayment rate for low values. Then it reaches a maximum and decreases before flattening at the right tail.
Figure 5.1c shows the relationship between the house price index (HPI) ratio and the observed prepayment rate. This relationship is more complex and shows multiple aperiodic curves.

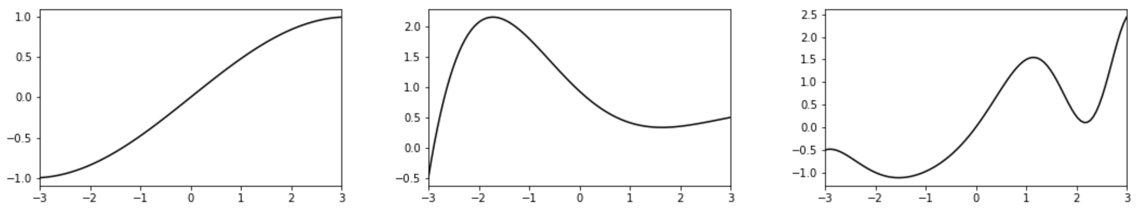
As Rumelhart et al. [1986] and Cybenko [1989] proved, neural networks with as much as one single hidden layer are able to represent all possible relationships. Whether or not a trained neural network will actually capture the true underlying function depends on the structure of the train set. To illustrate the dependence of the model performance on the train set distribution, we define three functions that have similar characteristics as the relationships in figure 5.1:

a) Function 1: $f_1(x) = \sin(\frac{x}{2})$,

b) Function 2: $f_2(x) = \text{Beta}(\frac{x+3}{6}) + \frac{x}{6}$,

c) Function 3: $f_3(x) = \sin(x(\frac{x}{2} + 1)) + \frac{x}{2}$.

Functions $1-3$ are evaluated on the domain $x \in [-3, 3]$. In figure 5.2, plots of the functions are shown.



(a) Function 1 (b) Function 2 (c) Function 3

Figure 5.2: Functions $1-3$ on which performance of a neural network will be investigated.

To illustrate how the structure of the train set is related to the performance of a trained neural network, we will discuss three types of train set distributions.

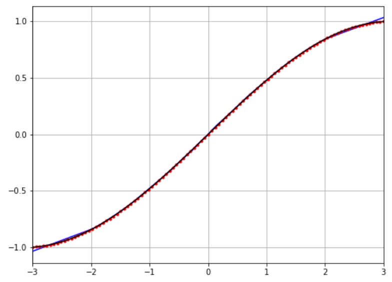
5.2 Performance on different train set distributions

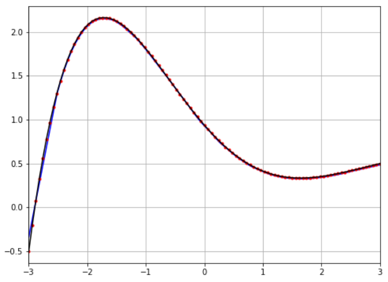
A neural network is trained on a dataset. Evidently, the performance of the network therefore depends on the distribution of that dataset.
In this section, predictions by trained neural networks will be compared to functions $1-3$ that have generated the data. In all cases, a neural network with two hidden layers consisting of 100 neurons each is defined such that there are sufficient neurons to generate functions of the complexity of f_1, f_2 and f_3. The ReLU activation function is used and the weights and biases are initialized by a normal distribution with mean zero and standard deviation $\frac{1}{5}$ (or the length-scale: $\gamma^{(l)}=5$ for $l=1,\ldots,L$). In all cases, 100 observations were generated by the functions to be used for training. The patience parameter is fixed at 100 epochs. The regularized mean squared error is used to measure the loss of the model.

In the following figures, red dots represent the data points provided for training. These observations are generated by functions $1-3$ which are plotted as black lines. The model predictions are plotted as blue lines. The plots are best viewed on a computer screen.
As the validation set, 100 evenly distributed observations over the function domain $[-3,3]$ are used.

5.2.1 Evenly distributed observations

In the simplest case, observations in the train set are evenly distributed over the domain. The train set covers the entire domain in this case. Therefore, neural networks are provided with enough information to capture the relationships of functions $1-3$. This can be observed in figure 5.3 where the predictions of three trained neural networks almost exactly coincide with the functions that generated the observations.


(a) Function 1


(b) Function 2

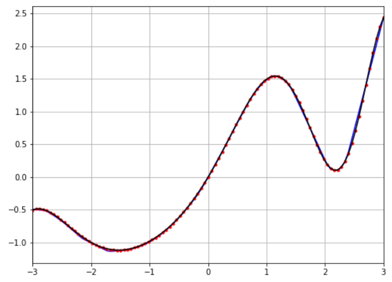

(c) Function 3

Figure 5.3: Even distribution of observations.

This is in accordance with the property that neural networks are universal approximators. The network contains a sufficient number of neurons to simulate the outcomes of the functions on the domain. In this case, where the observations are evenly placed over the entire domain, the function is replicated very well.
In table 5.1, the average mean squared errors of the predictions on the train sets and validation sets of 20 trained neural networks are shown.

	Number of epochs	**MSE train set**	**MSE validation set**
Function 1	1210	$8.2 \cdot 10^{-5}$	$8.2 \cdot 10^{-5}$
Function 2	960	$8.4 \cdot 10^{-4}$	$8.4 \cdot 10^{-4}$
Function 3	1200	$6.6 \cdot 10^{-4}$	$6.6 \cdot 10^{-4}$

Table 5.1: Average number of epochs, MSE on the test set and MSE on the validation set for 20 trained neural networks on the three functions with evenly distributed training observations.

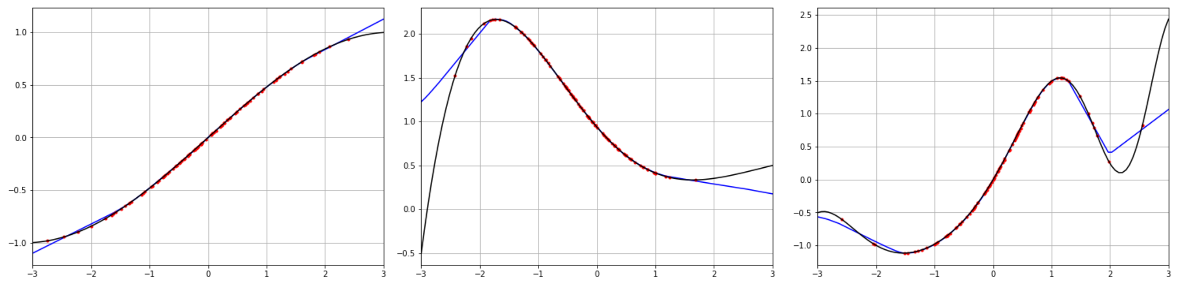
Since the train set and the validation set are the same, the errors on both sets are equal. The neural network learns relationships between the input and function values until the error is very small. The process of training is stopped when the error does not significantly decrease further.
In figure A.1 in appendix A.1, the decay of the error on the train set and validation set during training can be observed for the trained neural networks shown in figure 5.3.

This setup of evenly distributed observations assumes that available information is evenly distributed over the entire domain of the feature(s). Unfortunately this is rarely the case.

5.2.2 Standard normally distributed observations

Often, observations are not available over the whole domain, especially when one deals with many dimensions. The amount of data that is needed to cover the entire domain grows exponentially. This is called the *curse of dimensionality* [Donoho, 2000].
This is also the case for modelling prepayment behavior where multiple variables are used. In figure 5.1a, the sizes of the dots represent the amount of observations available in that region. Towards the outside of the domain, observations are scarce. Also in figures 5.1b and 5.1c regions with high and low observation density can be observed.
Note that the dots in figure 5.1 represent the amount of observations for a specific feature across all other feature values. The observations within an interval for a specific feature are spread over the domain of the other feautures.

Like in figure 5.2, it is often the case that most information is available around some mean in the middle of the domain and the more scarce regions are towards the domain boundaries.
To illustrate how the performance of trained neural networks are influenced by train sets with low observation density towards the boundaries of the domain, we generate 100 observations where $x \sim \mathcal{N}(0, 1)$. The predictions of trained networks can be seen in figure 5.4.



(a) Function 1 (b) Function 2 (c) Function 3

Figure 5.4: Standard normally generated observations.

When these kind of train sets are provided, the model has to extrapolate predictions or has to interpolate between points that are further from each other. For f_1, which is close to linear until the end of the domain, the prediction is not far from the function values.
For the other functions the prediction becomes much worse towards the boundaries where fewer observations are available. This is caused by the derivatives of f_2 and f_3, which are less stable than for f_1. The faster the derivative changes of the function are to each other, the less accurate the predictions of the neural network become.

In table 5.2, the average mean squared errors of the predictions on the train sets and validation sets of 20 trained neural networks are shown.

	Number of epochs	**MSE train set**	**MSE validation set**
Function 1	1650	$3.8 \cdot 10^{-5}$	$1.2 \cdot 10^{-3}$
Function 2	1680	$2.1 \cdot 10^{-4}$	$3.1 \cdot 10^{-1}$
Function 3	1170	$7.2 \cdot 10^{-4}$	$2.8 \cdot 10^{-1}$

Table 5.2: Average number of epochs, MSE on the test set and MSE on the validation set for 20 trained neural networks on the three functions with standard normally distributed training observations.

The error on the train set still decreases to zero but the error of the validation set is significantly higher. The observations do not contain enough information to reduce the loss on the validation set further. Fitting the neural network better on the train set does not improve the performance of capturing the underlying function. As a consequence, the MSE on the validation set is higher than for the evenly distributed observations where the characteristics of the function over the whole domain were represented in the train set.
In figure A.2 in appendix A.2, the decay of the error on the train set and validation set during training can be observed for the trained neural networks shown in figure 5.4.

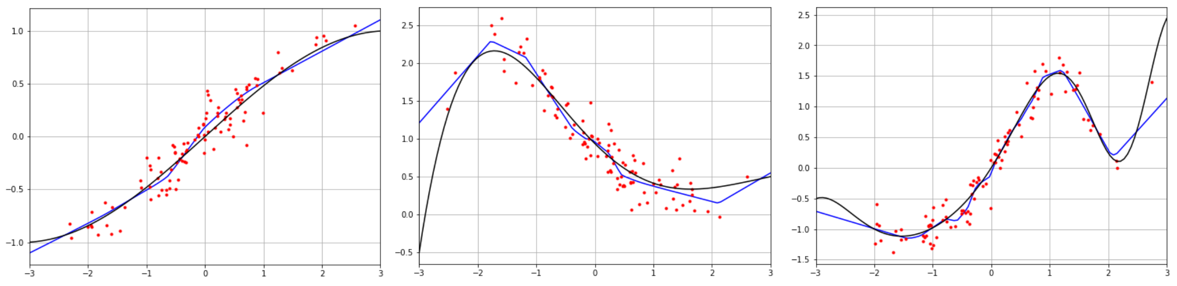
For these examples, the instances of test set could be coincided by a relatively smooth function since they exactly equal values on functions $1-3$. In many problems this is not the case.

5.2.3 Standard normally distributed observations with white noise

When working with a dataset of observations, especially when it concerns human behavior, there is no precise function that is followed. The observations for similar features may be different. A model is meant to capture some trend that explains the behavior in general.
In other words, observations in a dataset are not necessarily points on the function that we seek to explain the data. The dataset includes white noise on top of the 'average' behavior.
As an example, consider two batches that contain mortgages with identical characteristics. The observed CPR for the batches is not equal in general. Some clients may prepay their mortgage while others do not, even if they have identical mortgages. The observed CPRs for the batches are merely interpreted as observations around some average CPR for that type of batch.

To illustrate how the performance of neural networks depends on noise in the train set, white noise

$\epsilon \sim \mathcal{N}(0, \frac{1}{5})$ is added to the train instances to simulate observation noise. We still consider normally distributed x-coordinates of the observations. Predictions generated by trained networks are shown in figure 5.5.



(a) Function 1 (b) Function 2 (c) Function 3

Figure 5.5: Standard normally generated observations with white noise added.

The noise in the train set could mislead the neural network. The trained model then captures a relationship that is rather caused by noise than by actual driving properties (overfitting). The neural networks are protected in two ways against capturing too much noise.
Firstly, as mentioned in section 4.1, the regularization coefficients in the loss function stimulate less complex solutions of the model. This prevents the neural network from capturing too much noise in the train set.
As a second protection, the training is stopped when the performance on the validation set does not improve further. Early stopping was introduced in section 3.2.4.

In table 5.3, the average mean squared errors of the predictions on the train sets and validation sets of 20 trained neural networks are shown.

	Number of epochs	**MSE train set**	**MSE validation set**
Function 1	570	$3.6 \cdot 10^{-2}$	$5.0 \cdot 10^{-2}$
Function 2	1110	$4.2 \cdot 10^{-2}$	$3.5 \cdot 10^{-1}$
Function 3	1120	$4.4 \cdot 10^{-2}$	$3.2 \cdot 10^{-1}$

Table 5.3: Average number of epochs, MSE on the test set and MSE on the validation set for 20 trained neural networks on the three functions with standard normally distributed training observations with white noise.

In this scenario, the MSE on the train set can not be reduced further without increasing the error on

the validation set. This is due to the noise in the dataset. Reducing the error further causes overfitting from which the error on the validation set increases.
In figure A.3 in appendix A.3, the decay of the error on the train set and validation set during training can be observed for the trained neural networks shown in figure 5.5.

We have seen that predictions towards the boundaries of the domain appear to be less accurate. There is often a lack of information in certain (extreme) regions of the domain which causes the neural network to generate inaccurate predictions. The regular neural neural networks that were trained above do not provide information to identify those inaccurate predictions. We will now discuss methods to quantify a measure of uncertainty in order to get insight in the accuracy of a prediction.

5.3 Evaluation of model uncertainty estimates

To understand the uncertainty of a prediction, one should understand the different types of uncertainty of which it may exist first. We will make a distinction between *epistemic* and *aleatoric* uncertainty [Kendall and Gal, 2017].
The first type, epistemic uncertainty, is also known as model uncertainty. It represents the lack of knowledge about interpretation of the function that generated our data. This includes uncertainty about which parameters should be used. It is uncertainty that is caused because data is hidden or the model neglects certain effects. The name epistemic comes from the Greek word $\epsilon\pi\iota\sigma\tau\eta\mu\eta$ (*epistèmè*) which means knowledge [Der Kiureghian and Ditlevsen, 2009].
Aleatoric uncertainty is also known a statistical uncertainty. It represents the error that is inevitably made in every prediction, even if the function generating the observations is completely understood. Aleatoric uncertainty is caused by noise that can not be reduced. In section 5.2.3, aleatoric uncertainty was generated by adding white noise $\mathcal{N}(0, \frac{1}{5})$ to the data. The name aleatoric comes from the Latin word *alea* which means dice or gambling [Der Kiureghian and Ditlevsen, 2009].
Epistemic uncertainty can be reduced by gathering more data that provides knowledge about the underlying function while aleatoric uncertainty cannot be reduced by improving the model. This thesis focuses on the epistemic uncertainty of predictions to get insight in when the prediction of a neural network is unreliable due to a lack of information.

In this section, we will first illustrate that the point estimates generated by a regular neural network do not provide information about uncertainty. Then we will generate and visualize samples from a neural network trained with dropout. These samples can be used to compute uncertainty estimates that indicate how accurate a prediction is expected to be. Then a scenario is discussed where the method using dropout does gives rise to unreliable uncertainty estimates and a solution is suggested. The section is concluded by an evaluation of how the uncertainty estimates depend upon the activation function.

5.3.1 The problem with point estimates

The characteristics of a trained neural network generally cannot be captured by a regular function. Since input as well as model parameters are deterministic, just a point estimate based on the many nonlinear functions is provided. There is no direct indication of how the model's prediction is driven and uncertain the output is expected to be. A regular neural network does not warn the user if the output is highly uncertain, even if it is based on unreliably little data. This poses a problem for situations where the uncertainty of the output is relevant.

In section 5.2, we illustrated that a neural network is trained by a dataset and therefore the quality of its prediction depends on the data. In regions where there are few observations, the confidence in the prediction drops. Also sharp curves appear to negatively influence the capability of converging to the function that generated the data.
To illustrate how one can be mislead by the prediction of a single neural network, we trained twenty networks independently on the same dataset. In figure 5.6, the predictions of the twenty neural networks trained by data generated by f_3 are shown.

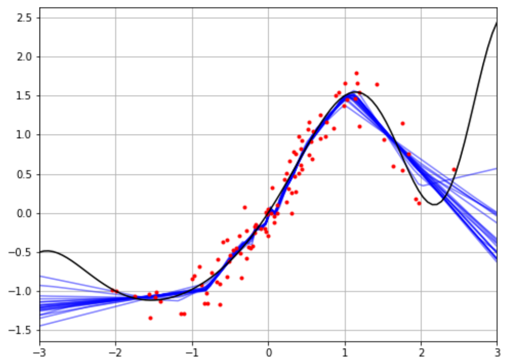


Figure 5.6: Disagreement towards the boundaries of domain between twenty neural networks that are trained independently on the same data.

Due to random initialization, the models are not identical. This is especially the case at the boundaries of the domain where observations are more rare and the curves steeper. Here, there is not enough information to adjust the network. Nevertheless, the prediction of a single network is a point estimate and it is presented with the same certainty as predictions where the neural networks seem to agree.

When only one neural network is used, it is easy to be mislead by the data without knowing that the proposed prediction is unreliable. The problem is that uncertainty for a prediction can only be quantified using the losses with respect to the test data. Often there is not enough data available to assess the uncertainty at specific regions of the domain. Therefore the user only knows an average error. In the situation of figure 5.6 the average error may be rather small while predictions close to $x = 3$ are highly inaccurate.

A possible solution is an *ensemble* method [Hansen and Salamon, 1990] [Dietterich, 2000]. This implies training multiple networks like is done in figure 5.6. A combination of those networks is used as a final model. The combination may be a simple average or a more complex function of the individual networks. This method reduces the risk of being extremely wrong since it takes into account multiple 'votes'. The uncertainty of the ensemble could be quantified by the disagreement between the models from which it is built.
Using such an uncertainty measure in figure 5.6, the uncertainty estimate would be small in the center of the domain and large towards the boundaries. This is due to the lack of information at the boundaries to adjust the neural networks. In section 5.3.4, ensembles will be used to identify uncertain regions of the domain.

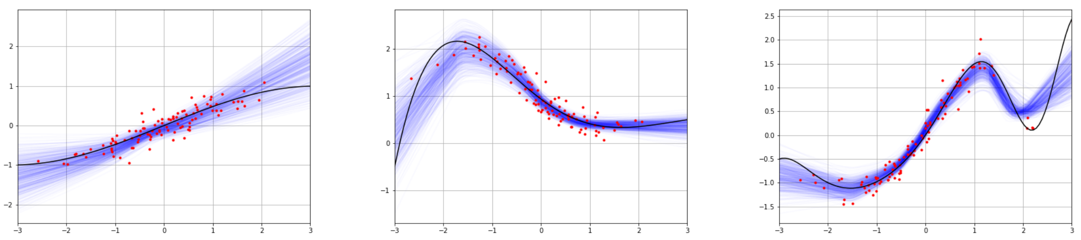
With the ensemble method, all interpretations of the data require training a separate neural network. This is not the case for the method using dropout as suggested in chapter 4. The interpretations can be obtained by generating stochastic forward passes through a single neural network trained with dropout. This method is computationally much more efficient than an ensemble since it does not require training multiple neural networks.

5.3.2 Generating stochastic forward passes through a neural network with dropout

From a single neural network trained with dropout, multiple predicting networks can be sampled. Now we will dicuss stochastic forward passes obtained using neural networks with dropout.
We will see that, in addition to identifying regions with few observations, this method appears to capture regions where the behavior is highly variable. In the case of noisy input data, these regions are important since their predictions are significantly influenced by the noise of the input.

The neural networks that are used have the same structure as in section 5.2. In addition, we define a neuron activity rate $p_l = 0.8$ for $l = 1, \ldots, L-1$ which implies that neurons in the hidden layers are active with a probability 0.8. Also we define $p_0 = 1$ such that the input neuron is always active since, in the case of dropping out the input neuron, the output would be based on biases only and thus be constant.

Once a network is trained with dropout, samples can be drawn by generating predictions with different dropout masks (see section 4.1). As we concluded, samples generated by different dropout masks approximate the mathematically equivalent Bayesian neural network. In figure 5.7, forward passes through the network are plotted.



(a) Function 1 (b) Function 2 (c) Function 3

Figure 5.7: 500 stochastic forward passes through neural networks with dropout applied. The observations are standard normally distributed over the domain and white noise $\epsilon \sim \mathcal{N}(0, \frac{1}{5})$ is added to the function values.

Visualizing the predictions of the samples gives an insight in the agreement between the posteriors. The further samples are from each other, the more they disagree on how the data should be interpreted.
It can be observed that the samples disagree more in some regions than in others. In particular the results at the boundaries of the domain and regions where the slope of the underlying function changes quickly appear to give rise to more diverse predictions. These regions with highly variable underlying functions and extreme (rare) situations are the ones we wish to identify since we noticed in section 5.2.3 that the underlying function was captured poorly in these regions.

The level of disagreement between the forward passes allows us to reason about the uncertainty of predictions provided by a neural network trained with dropout. Now we will focus on quantifying that uncertainty.

5.3.3 Quantifying model uncertainty estimates for different setups

We will discuss a measure of uncertainty represented by the standard deviation of generated stochastic forward passes. The model variance can be derived from a trained neural network since sampling from the posterior allows us to estimate the first two raw moments of the predictive distribution $p(\mathbf{y}|\mathbf{x}, \theta)$ empirically [Gal and Ghahramani, 2016]. The optimized parameters for the neural network will be denoted by θ. Let $\hat{\mathbf{y}}(\hat{\mathbf{x}}, \theta)$ be the prediction for input $\hat{\mathbf{x}}$.

Let us denote by $\hat{\omega}_t = g(\theta, \mathbf{v}_t)$ for $t = 1, \dots T$, the model parameters θ (weights and biases) multiplied by the dropout masks. The dropout masks are represented by the collections $\mathbf{v}_t$ containing binary vectors to perform the dropout (see section 4.2.2). The first two moments can be estimated by using a Monte Carlo estimation with T stochastic forward passes:

$$\begin{aligned} \hat{\mathbb{E}}_{p(\hat{\mathbf{y}}|\hat{\mathbf{x}},\theta)}(\hat{\mathbf{y}}) &= \frac{1}{T}\sum_{t=1}^{T} \hat{\mathbf{y}}(\hat{\mathbf{x}}, \hat{\omega}_t), \\ \hat{\mathbb{E}}_{p(\hat{\mathbf{y}}|\hat{\mathbf{x}},\theta)}(\hat{\mathbf{y}}^T\hat{\mathbf{y}}) &= \tau^{-1}I + \frac{1}{T}\sum_{t=1}^{T} \hat{\mathbf{y}}(\hat{\mathbf{x}}, \hat{\omega}_t)^T \hat{\mathbf{y}}(\hat{\mathbf{x}}, \hat{\omega}_t), \end{aligned} \tag{5.1}$$

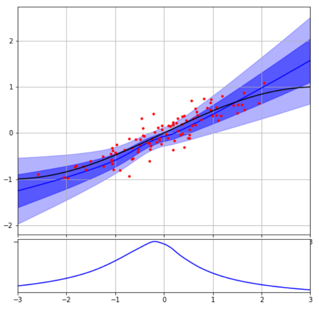
where τ is a hyperparameter that represents the observation noise (aleatoric uncertainty). In Gal [2016], a distinction is made between *homoscedastic* uncertainty, where τ is assumed constant, and *heteroscedastic* uncertainty where τ may vary over the domain and has to be modelled seperately.

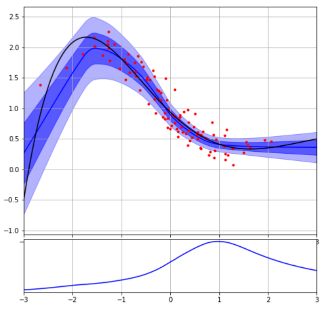
The variance of the predictions can be estimated by combining the two expressions in (5.1):

$$\hat{\text{Var}}_{p(\hat{\mathbf{y}}|\hat{\mathbf{x}},\theta)}(\hat{\mathbf{y}}) = \tau^{-1}I + \frac{1}{T}\sum_{t=1}^{T} \hat{\mathbf{y}}(\hat{\mathbf{x}}, \hat{\omega}_t)^T \hat{\mathbf{y}}(\hat{\mathbf{x}}, \hat{\omega}_t) - \hat{\mathbb{E}}_{p(\hat{\mathbf{y}}|\hat{\mathbf{x}},\theta)}(\hat{\mathbf{y}})^T \hat{\mathbb{E}}_{p(\hat{\mathbf{y}}|\hat{\mathbf{x}},\theta)}(\hat{\mathbf{y}}). \tag{5.2}$$

We will focus on the relative differences in uncertainty estimates for the predictions. Our interest lies in identifying those observations that are expected to be worst, i.e. those observations with relatively the highest estimated (epistemic) uncertainty.
Since we will not evaluate the absolute standard deviation, the first term $\tau^{-1}I$ representing the aleatoric uncertainty in the expression for the variance is ignored. This leaves an expression for the estimated epistemic uncertainty. To evaluate differences in estimated confidence in the predictions, we define the estimated confidence to be one divided by the square root of the estimated epistemic uncertainty. Throughout this thesis, the terms estimated uncertainty and estimated confidence will be used both.

Note that this expression of uncertainty can be extracted without changing the model itself. It is obtained by estimating the moments by generating random samples from the posterior. Therefore it is computationally efficient. In figure 5.8, the confidence estimates generated by the stochastic forward passes from figure 5.7 are visualized. The dark en light blue bands represent respectively one and two times the square root of the estimated epistemic uncertainty from the mean prediction. The blue line below the graph represents the estimated confidence in the prediction represented by one divided by the square root of the epistemic uncertainty.


(a) Function 1


(b) Function 2

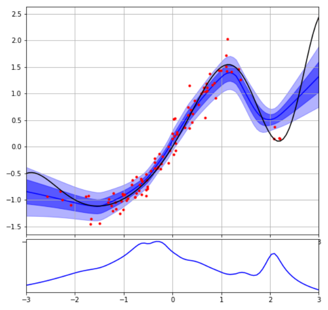
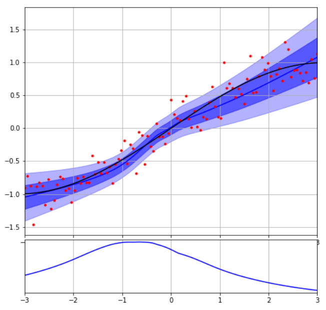
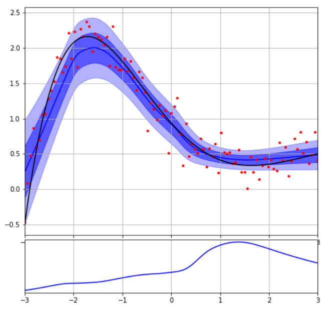

(c) Function 3

Figure 5.8: Confidence estimates obtained by 500 stochastic forward passes through dropout neural network with a train set that consists of observations with standard normally distributed x-coordinates.

The disagreement between networks with different dropout masks can be observed more clearly than in figure 5.7. Regions with relatively few observations are paired with a low estimated prediction confidence. Towards the boundaries of the domain, confidence in the prediction is less than in the center where more information is available.
In figure 5.9, confidence estimates are generated for a train set that consists of evenly distributed observations with white noise.


(a) Function 1


(b) Function 2

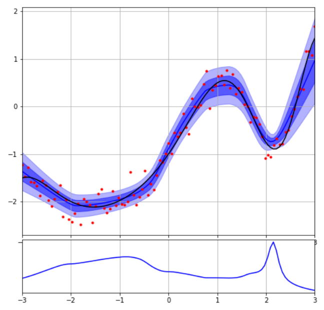

(c) Function 3

Figure 5.9: Confidence estimates obtained by 500 stochastic forward passes through dropout neural network with a train set that consists of observations with evenly distributed x-coordinates.

It can be observed that the estimated confidence in the predictions decreases towards the boundaries too. This is justified since there are no observations outside the $[-3, 3]$ which implies that there

is relatively less information towards the boundaries. As could be expected, the confidence overall is slightly more stable than for the normally distributed x-coordinates.
Let us study the relationship between the confidence estimates and the errors under both setups further. In table 5.4, the mean squared errors and confidence estimates can be observed.

	Normally distributed		**Evenly distributed**	
	Confidence	**MSE**	**Confidence**	**MSE**
Function 1	6.8	$5.0 \cdot 10^{-2}$	7.6	$4.6 \cdot 10^{-2}$
Function 2	7.8	$3.5 \cdot 10^{-1}$	11.1	$8.3 \cdot 10^{-2}$
Function 3	7.0	$3.3 \cdot 10^{-1}$	7.7	$9.2 \cdot 10^{-2}$

Table 5.4: Average confidence estimate and MSE on the validation set for 20 trained neural networks on the three functions with evenly and normally distributed observations.

In table 5.4, a clear difference can be seen between the averages under both setups. For the evenly distributed observations, estimated confidence is higher and the mean squared errors are lower than for the train set with normally distributed observations. This could be expected since the evenly distributed observations contain more information about the function over the entire domain.
We concluded in section 5.2 that predictions in regions with few observations should be treated carefully. These regions appear to be identified by a low estimated confidence (high epistemic uncertainty). In section 6.3, a neural network for predicting prepayment behavior will be trained with dropout. Then we will observe that confidence estimates are a good indicator for model performance.

The seam side from confidence estimates using dropout is that confidence may be overestimated. The stochastic passes are generated around a single interpretation of the data. We will see that it may be wise to take multiple interpretations of the data into account to be more conservative when it comes to confidence in the prediction.

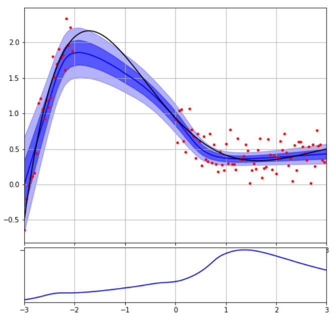
5.3.4 Shortcoming of uncertainty estimates using dropout

The (un)certainty about the correct interpretation of data is a difficult topic. It gives rise to a more philosophical discussion about how conservative one should be in trusting model predictions. This is especially the case when predictions are made in regions where no data is available at all.
Take for example the estimated confidence close to $x = 3$ in figure 5.8c. It could be stated that it should be much smaller since the function values are far away from the predictions. Therefore the network should not provide its predictions with such high confidence. On the other hand, increasing

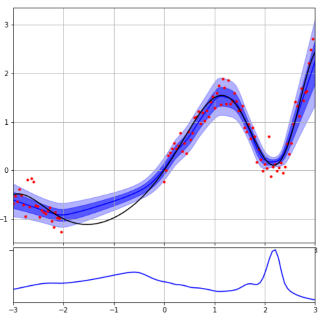
confidence bounds may lead to unnecessary conservatism. Here, a trade-off has to be made between safe predictions and predictive power.
The impact of missing improbable events like at $x = 3$ in figure 5.8c that lie outside of the collection of available data (sometimes called *Black Swans*) is discussed rigorously in Taleb [2007].

The modelling of this impact falls outside the scope of this thesis. However, we would like to briefly discuss how a neural network may be prevented frome naively underestimating the uncertainty paired with its prediction. When dealing with models within a bank, it is adviced to be conservative rather than to be too opportunistic. Scenarios identified as uncertain can be passed to be evaluated by another model or an expert. As we noted, estimated uncertainty close to $x = 3$ in figure 5.8c is too small. Although the uncertainty may not be appropriately large, the region is identified as uncertain. These observations are identified sufficiently risky to be passed to an expert.

The computed uncertainty estimates are unreliable when there is a lack of information centrally in the domain. Using an ensemble to identify the critical region is more useful under these circumstances. To illustrate this we consider two scenarios with 'data gaps' in the domain.
In the case of a data gap, the model has to interpolate between regions with observations. In figure 5.10, confidence bounds for the functions f_2 and f_3 are plotted for datasets that have a gap between $x = -2$ and $x = 0$. The 100 observations in the train set are evenly distributed over the remainder of the domain: $[-3, -2] \cup [0, 3]$.



(a) f_2: $[-3, -2] \cup [0, 3]$

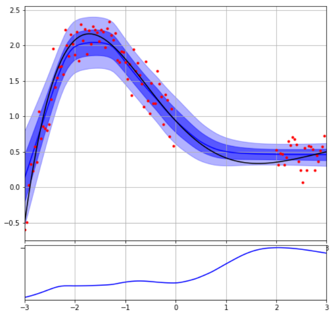


(b) f_3: $[-3, -2] \cup [0, 3]$

Figure 5.10: Confidence estimates obtained by stochastic forward passes through a dropout neural network trained on a set with a gap at $(-2, 0)$.

As can be observed, the confidence estimates within these blind spots appear to be unjustifiably high. According to the computed estimates, a relatively confident prediction is given at $x = -1$ in

figure 5.10b. Evidently, underestimation of uncertainty is undesirable even though the functions are accurately approximated in the gap.
A problem arises when a lack of information is in a region where the model's interpolation leads to inaccurate predictions. In figure 5.11, confidence bounds for the functions f_2 and f_3 are plotted for datasets that have a gap between $x = 0$ and $x = 2$. The 100 observations in the train set are evenly distributed over the remainder of the domain: $[-3, 0] \cup [2, 3]$.


(a) f_2: $[-3, 0] \cup [2, 3]$

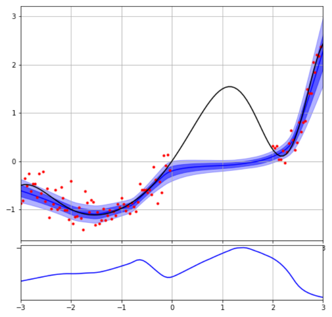

(b) f_3: $[-3, 0] \cup [2, 3]$

Figure 5.11: Confidence estimates obtained by stochastic forward passes through a dropout neural network trained on a set with a gap at $(0, 2)$.

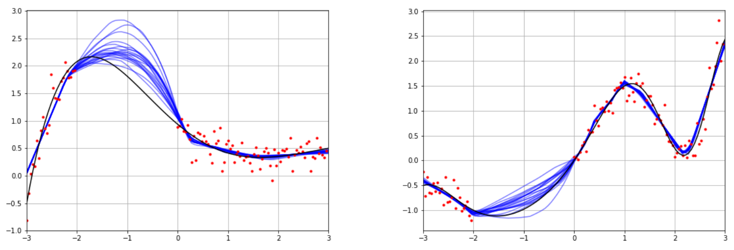
In this scenario, the predictions around $x = 1$ in figure 5.11b are inaccurate. This however is not reflected in the estimated confidence.
In table 5.5, the average estimated confidence and the MSE on the validation sets can be observed for 20 trained neural networks under both scenarios. The results are compared with the situation where observations are evenly distributed over the domain.

	$[-3, -2] \cup [0, 3]$		$[-3, 0] \cup [2, 3]$		**Evenly distributed**	
	Confidence	**MSE**	**Confidence**	**MSE**	**Confidence**	**MSE**
Function 2	11.3	$8.6 \cdot 10^{-2}$	10.8	$8.1 \cdot 10^{-2}$	11.1	$8.3 \cdot 10^{-2}$
Function 3	8.7	$1.1 \cdot 10^{-1}$	10.7	$5.3 \cdot 10^{-1}$	7.7	$9.2 \cdot 10^{-2}$

Table 5.5: Average confidence estimates and MSE on the validation set for 20 trained neural networks on functions 2 and 3 in case of both setups with a gap. The results are compared to the averages of evenly distributed observations over the entire domain.

It can be observed that for f_2, the estimated confidence is fairly stable under the three scenarios.

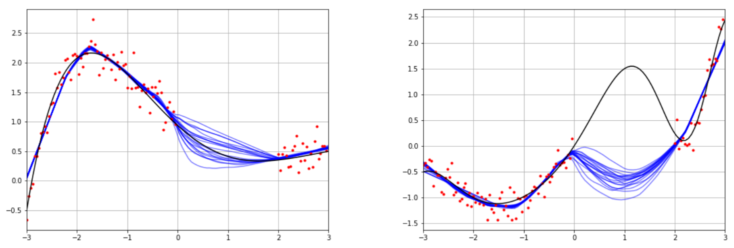
The MSE on the validation set is stable since function 2 is approximated well by the interpolation. For f_3 with the gap at $(-2, 0)$, we see that the confidence is slightly higher while the prediction measure by the MSE is larger. This is undesirable. The problem is even larger for neural networks predicting f_3 with a gap at $(0, 2)$. The average confidence is higher than for neural networks trained on a train set with evenly distributed observations while the MSE is much larger.
This scenario clearly illustrates that it can be dangerous to be overconfident in regions that lack information. The method using dropout to generate estimated confidence appears to be unable to identify those regions. As a solution, an ensemble can be trained on a train set with a data gap. In figure 5.12, predictions of 20 trained neural networks are shown for the situations where data is available on $[-3, -2] \cup [0, 3]$.



(a) f_2: $[-3, -2] \cup [0, 3]$ (b) f_3: $[-3, -2] \cup [0, 3]$

Figure 5.12: Different interpretations in the region without observations.

In contrast to the method using dropout, the independently trained neural networks disagree more in the region that lacks observations.
This can also be observed in figure 5.13.



(a) f_2: $[-3, 0] \cup [2, 3]$ (b) f_3: $[-3, 0] \cup [2, 3]$

Figure 5.13: Different interpretations in the region without observations.

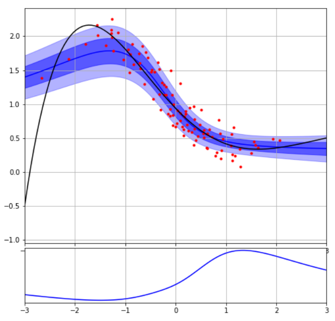
Although predictions around $x = 1$ are not accurate in figure 5.13b, they are paired with relatively high uncertainty. The predictions in this region are identified as uncertain and can be re-evaluated by

another model or an expert.
It can be observed that the networks are close together in the regions where observations are available. In general holds: The further away from observations while interpolating, the further predictions of the different neural networks are from each other. As we observed in figure 5.6, networks also tend to disagree more for further extrapolation.
Multiple neural networks can be used this way to identify regions where observations are scarce. An expert judgment can be requested in those regions where the different neural networks disagree. This reduces the risk of naively following model predictions.

Fortunately, database structures like these do not often occur naturally. More often, a lack of data is observed at the boundaries of the domain. This is also the case in our dataset for prepayment behavior. Therefore we will not treat the case of large gaps within the data further but rather focus on the boundaries of the domain.

5.3.5 Uncertainty estimates under Sigmoid and hyperbolic tangent activation functions

So far, the ReLU activation function has been used in this thesis. In section 3.1, we also introduced the Sigmoid function and the hyperbolic tangent as common activation functions. In this section, we will briefly discuss how the activation function influences the shape of the prediction and its confidence estimate.
In figure 5.14, confidence estimates are computed for neural networks using the Sigmoid activation function. The networks are trained using exactly the same train data as for figure 5.8.


(a) f_2 using sigmoid

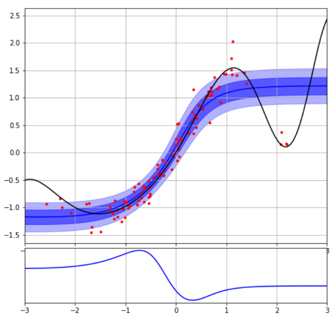
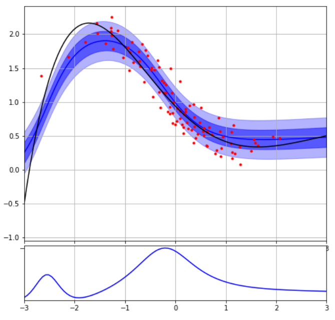

(b) f_3 using sigmoid

Figure 5.14: Evaluation of the confidence estimates for f_2 and f_3 trained by standard normally distributed observations using the Sigmoid activation function.

The Sigmoid activation functions gives rise to a smoother prediction. Also the estimated confidence is smoother than in figure 5.8 where the ReLU activation function is used. As a result, the outside of the domain, which we would like to be identified by a higher estimated uncertainty, is not clearly paired with a higher estimated uncertainty. The estimated uncertainty grows faster towards the boundaries when the ReLU activation function is used. The confidence computed using the Sigmoid activation function are therefore less useful for us.

In figure 5.15, confidence estimates are computed for neural networks using the hyperbolic tangent activation function.


(a) f_2 using hyperbolic tangent

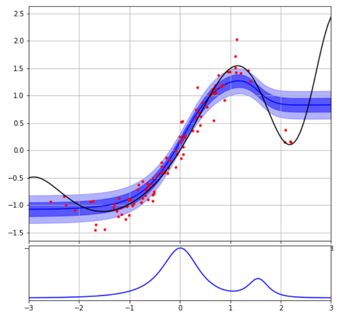

(b) f_3 using hyperbolic tangent

Figure 5.15: Evaluation of the confidence estimates for f_2 and f_3 trained by standard normally distributed observations using the hyperbolic tangent activation function.

For the hyperbolic tangent, the estimated confidence appears to be high in the regions where the slope of the function is steep. There exists no clear relationship between scarceness of observations and high uncertainty. Therefore, the hyperbolic tangent is not preferable over the ReLU or Sigmoid activation function for identifying uncertain predictions.

The computed uncertainty for the functions f_2 and f_3 using sigmoid and hyperbolic tangent activation functions are smaller while extrapolating than when the ReLU is used. The ReLU activation function therefore provides more conservative confidence estimates. Since we would like to identify observations in regions with few observations, this activation function is most useful for us. The ReLU activation will be used in the remainder of this thesis when uncertainty in predicting prepayment behavior is discussed.

Chapter 6

Predicting prepayment behavior using neural networks

In this chapter we will discuss how a neural network can be trained to predict prepayment behavior. In order to do so, a dataset on mortgage characteristics with corresponding prepayments is used.

This chapter starts with a discussion about the structure of the dataset. Some adjustments to the dataset are made before a neural network can be trained well and efficiently. These adjustments will be justified mathematically. To model Relocation behavior, several features that drive Relocation prepayments are introduced. Then the performance of a trained neural network for predicting conditional prepayment rates is evaluated.
A neural network will be trained with dropout such that we are able extract uncertainty estimates for the CPR predictions. We will discuss the performance of the resulting neural network. Then the relationship between the performance of the network and the uncertainty estimates is evaluated. To identify the contracts whose predictions are expected to be uncertain, a characterization of the most uncertain regions of the domain is made. Then some attention is paid to the sensitivity of the predicted CPR with respect to the different features. The chapter is concluded with a discussion of the distribution of values for the uncertainty estimates.

6.1 Dataset structure

As discussed in section 3.2, a train set containing pairs $(\mathbf{x}_i, \mathbf{y}_i)$ for $i \in \{1, \dots, N\}$ of features $\mathbf{x}_i$ and targets $\mathbf{y}_i$ is needed in order to train a neural network. We are considering a classification of prepayments and no prepayment. The observations consist of values for the driving variables (see section 6.2) at different points in time and a target that represents whether or not a (specific type of) prepay-

ment occurred that month. In general, prepayment events are rare. For an outstanding mortgage, the probability that a prepayment event occurs is very small on a monthly basis. This implies that most of the observations in the dataset belong to the class 'No Prepayment'.
The dominance of this class can be observed in table 6.1. The dataset is called *imbalanced*.

Class	**Fraction of observations**
Relocation	0.48%
Refinancing externally	0.36%
Curtailment high	0.04%
Curtailment low	0.31%
Reconsider	0.09%
Renteafkoop	0.13%
No Prepayment	98.60%

Table 6.1: Fraction of observations within each class is highly imbalanced.

In the introduction of this section, it was stated that we would be focusing on Relocation prepayments. There are three main reasons for this choice.
The first is that Relocation prepayments are considered an important risk for a bank. Prepayments of this type are often of high volume (in contrast to for example Curtailment low). Secondly, the Relocation class has the most instances in the dataset which is advantageous when training a network. The third reason is that Relocations are hard to predict. A neural network is able to capture highly nonlinear relationships [Lek et al., 1994] which makes it interesting to evaluate the performance on predicting the complex Relocation behavior.

6.1.1 Class imbalance methods

There is a problem when one tries to train a neural network on an imbalanced train set. It appears that, when classifying, a learning algorithm tends to be biased to the overrepresented classes, in our case the No Prepayment class [Saez et al., 2016]. To solve this issue, several methods are suggested. Examples are [Japkowicz and Stephen, 2002] [Saez et al., 2016]:

- Undersampling: Downsizing the majority class(es),
- Oversampling: Upsizing the minority class(es),
- Adjusting the loss function for misclassifying minority and/or majority class(es),

- Hybrid methods containing combinations of the above.

We will focus on undersampling. This method is based on removing observations from majority classes until the train set is balanced. There are two reasons for this choice.
Firstly because removing observations is intuitively a reasonable approach since observations of the same contract where no prepayment occurs are almost equal. They represent the same contract at different points in time. Therefore, removing instances does not directly lead to ignoring mortgages but merely to removing copies.
The second reason is that undersampling has the important advantage that the size of the train set is reduced. Since all the instances representing a prepayment will be contained in the undersampled train set, this can be seen as an increase in information density. The process of training the network therefore is faster for the smaller, undersampled set than for the large, original train set.
The neural network will be trained on the undersampled set. The predicted probabilities generated by the neural network are then transformed back to the right scale (see section 6.1.2) such that the performance can be computed using the imbalanced validation and test set.

Every type of prepayment has its own driving features. Also each type has another level of complexity which requires a deeper or shallower network. For this reason, a separate neural network has to be constructed for each prepayment type.
Undersampling has to be performed on No Prepayment in combination with one prepayment type. An advantage is that prepayment types that are observed relatively often do not have to be undersampled to match the amount of observations of the least occurring event. This would drastically reduce the amount of information about the more common prepayment types.

6.1.2 Re-scaling class probabilities

Assume we wish to predict the monthly probability that No Prepayment (NP) or a prepayment of type P occurs for a mortgage contract. We have a train set containing N observations of which N_{NP} instances of No Prepayment and N_P instances of the prepayment type that is considered. In accordance with table 6.1 we take $N_P < N_{NP}$.
The data can be undersampled by scaling the number of No Prepayment events with a factor $\frac{N_P}{N_{NP}}$. This implies removing $N_{NP} - N_P$ instances of No Prepayment. In the resampled set there are equally many observations of both events. The resampled probabilities that are computed by the neural network do, on average, not tend to favor one of the events.
Following the method of Pozzolo et al. [2015] and Saito [2018], the resampled probabilities can be adjusted to the prior probability predictions.

Let us denote the original imbalanced train set of observations by (X, Y) where X represents the network inputs and Y the observed behavior (targets). We will assume the elements of Y to be binary values. A prepayment event is denoted by $y = 1$ and No Prepayment by $y = 0$. The full set then consists of N couples $(\mathbf{x}_i, y_i)$ of which N_{NP} instances of No Prepayment and N_P instances of the prepayment type that is modelled.

Undersampling is the operation where the number of No Prepayment observations is reduced randomly in order to balance the dataset. This implies taking a random sample of N_P instances of the set of observations where no prepayment occurred. The balanced dataset with N_P instances of both events is denoted by (X', Y'). The probability of selecting a certain No Prepayment events for the undersampled train set equals $\beta = \frac{N_P}{N_{NP}}$.

Define a binary variable $s \in \{0, 1\}$ for the instances in the original train set (X, Y) that takes the value 1 if an instance is also contained in (X', Y') and taking the value 0 if it is not. Then we have $\mathbb{P}(s = 1|y = 0) = \beta$. Note that $\mathbb{P}(s = 1|y = 1) = 1$ since all instances representing a prepayment event are contained in the undersampled set.

We assume that the distribution p_s of variable s is independent from the input $\mathbf{x} \in X$ given the class $y \in Y$ [Pozzolo et al., 2015]: $p_s(s|y, \mathbf{x}) = p_s(s|y)$. This assumption implies that the distribution of input within classes does not change by removing observations: $p_x(\mathbf{x}|y, s) = p_x(\mathbf{x}|y)$. This assumption is based on the insight that a lot of observations in the majority class are redundant. Therefore they can be removed without changing the data distribution significantly.

Undersampling does change the probability distribution $p_y(y|\mathbf{x}, s = 1)$, which does not equal $p_y(y|\mathbf{x})$. Using the Bayes' rule, the following equation can be found:

$$\mathbb{P}(y = 1|\mathbf{x}, s = 1) = \frac{\mathbb{P}(s = 1|y = 1)\mathbb{P}(y = 1|\mathbf{x})}{\mathbb{P}(s = 1|y = 1)\mathbb{P}(y = 1|\mathbf{x}) + \mathbb{P}(s = 1|y = 0)\mathbb{P}(y = 0|\mathbf{x})}. \tag{6.1}$$

Recall that $\mathbb{P}(s = 1|y = 1) = 1$ since all prepayment instances are contained in the undersampled set. The probability of selecting a No Prepayment observation for the undersampled set is β. Hence equation (6.1) can be simplified to

$$\mathbb{P}(y = 1|\mathbf{x}, s = 1) = \frac{\mathbb{P}(y = 1|\mathbf{x})}{\mathbb{P}(y = 1|\mathbf{x}) + \beta\mathbb{P}(y = 0|\mathbf{x})}. \tag{6.2}$$

By the law of total probability we also know that $\mathbb{P}(y = 0|\mathbf{x}) = 1 - \mathbb{P}(y = 1|\mathbf{x})$. Substituting this term leads to

$$\mathbb{P}(y = 1|\mathbf{x}, s = 1) = \frac{\mathbb{P}(y = 1|\mathbf{x})}{\mathbb{P}(y = 1|\mathbf{x}) + \beta(1 - \mathbb{P}(y = 1|\mathbf{x}))}. \tag{6.3}$$

The class probabilities $\mathbb{P}(y = 1|\mathbf{x}, s = 1)$ conditioned on undersampling can be computed by a neural network trained using the undersampled train set (X', Y'). To transform these to prior probabilities $\mathbb{P}(y = 1|\mathbf{x})$, equation (6.3) can be rewritten as

$$\mathbb{P}(y = 1|\mathbf{x}) = \frac{\beta \mathbb{P}(y = 1|\mathbf{x}, s = 1)}{1 + \mathbb{P}(y = 1|\mathbf{x}, s = 1)(\beta - 1)}. \tag{6.4}$$

Probability predictions $\mathbb{P}(y = 1|\mathbf{x}, s = 1)$ obtained using the undersampled set can be transformed to predicted probabilities for the original dataset using expression (6.4).

Accuracy of this probability adjustment will be discussed for a neural network trained with dropout in section 6.3.2.

6.2 Neural network for predicting prepayment behavior

An important decision while making the model is the architecture of the network. Although one hidden layer networks are capable of approximating every general mapping it may require many neurons [Eldan and Shamir, 2015]. This increases the risk of overfitting the network. Also, shallow networks seem to have more trouble capturing complex relationships than deeper networks (consisting of more hidden layers) [Eldan and Shamir, 2015]. On the contrary, these deeper networks require more computing power and have training difficulties [Hunter et al., 2012].

For each prepayment type, a separate network has to be trained. We will only discuss results for Relocation prepayments. Several neural network structures have been tested. The network structure discussed in this section consists of three hidden layers with 300 neurons each. The prior length-scale is 5 and the regularization coefficients are computed as in expressions (4.43).
Five different features are used:

- HPI ratio change[1],
- Current month,
- Interest rate incentive[2],
- Age of loan in months,
- Remaining interest rate maturity in months[3].

[1]The ratio between the HPI ratio at the beginning of the contract and the current HPI ratio.
[2]The difference between the contractual interest rate and the current market interest rate.
[3]The number of remaining months until the end of the interest period.

During training, the way in which the features above influence the output is adjusted iteratively such that the loss as defined in equation (4.3) on the train set decreases. The loss on the validation set is monitored simultaneously. The training is stopped when the loss on the validation set increases. The epoch-wise decrease of the model loss on both the train and validation set can be observed in figure 6.1.

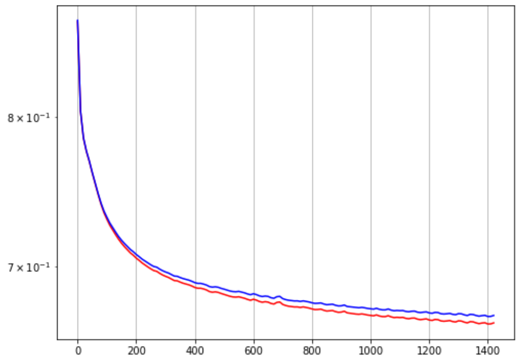


Figure 6.1: Decreasing model loss on the train set (red line) and validation set (blue line) during training.

It can be observed that the loss on the train set becomes smaller than the loss on the validation set. When the loss on the validation set increases, the network starts capturing the instances in the train set more than can be justified. Early stopping prevents the neural network from overfitting (see section 3.2.4).

6.2.1 Performance of the network

To quantify the performance of a model, the predictions are compared to corresponding targets. We will illustrate that one can be easily mislead by performance measures for the model in the case of an imbalanced dataset.
To visualize performance information, a confusion matrix is shown in table 6.2. Here, the symbols T and F stand for True prediction and False prediction, respectively.

	Actual prepayment	**No actual prepayment**
Prepayment predicted	T_P	F_P
No Prepayment predicted	F_{NP}	T_{NP}

Table 6.2: The confusion matrix for classifying prepayments.

Multiple performance measures can be extracted from the confusion matrix. The *accuracy* for example equals:

$$\text{accuracy} = \frac{T_P + T_{NP}}{N}. \tag{6.5}$$

Most performance measures are designed to measure the fraction of misclassifications [Bradley, 1997]. However, when evaluating a dataset which has classes that are represented unequally, these measures such as the accuracy may not be useful.
In the dataset represented by table 6.1 for example, a model classifying every possible input as 'No Prepayment' will have a high accuracy[4] while it does not represent any predictive power.

As a solution, the *ROC-curve* (*Receiver Operating Characteristic*) can be examined. The ROC-curve is generated by the values of $\mathbb{P}(T_P)$ and $\mathbb{P}(F_P)$ as the decision threshold is varied [Bradley, 1997]. The *AUC* (area under the ROC-curve) is a well known performance measure that is not biased towards majority classes since it is based on a ranking of probabilities [Kotsiantis et al., 2006].
An AUC value of 0.5 represents the predictive value equal to random guessing while a value of 1 implies the perfect ordering of prepayment probabilities of the observations in the test set.
The best achieved AUC for predicting Relocation events using a trained neural network is 0.663. This is comparable with performance of models that are currently used for modelling prepayment behavior.

6.3 Neural network with dropout for predicting prepayment behavior

It is important to have a measure to quantify the uncertainty regarding the predictions of prepayment rates since naively following bad predictions could lead to unexpected changes in the outstanding mortgage loan volume of a bank.
In this section we will discuss a trained neural network with dropout for predicting Relocations. The network consists of three hidden layers with 400 neurons each. An active neuron rate of 0.75 is used for the hidden layers such that, on average, equally many neurons are active per layer during each epoch as for the regular neural network (300 neurons). The neurons of the input layer are active with probability 1. The same features are used as in the construction of the neural network without dropout (see section 6.2).

6.3.1 Performance of the network

For a regular neural network without dropout, training is stopped when the model loss on the validation set starts to increase. This prevents the model from overfitting. During the training of a network with

[4]When modelling Relocation events, the accuracy equals: $\frac{T_P+T_{NP}}{N} \approx \frac{0+98.60\%}{0.48\%+98.60\%} = 99.52\%$.

dropout, the path of loss decrease is not smooth since a random selection of neurons is active each epoch. This causes the error to fluctuate around some mean model performance. A stopping criterion based on the loss in a single epoch can therefore not be applied. Since dropout is a method to prevent overfitting itself, no such stopping criterion is needed. This can be observed in figure 6.2.

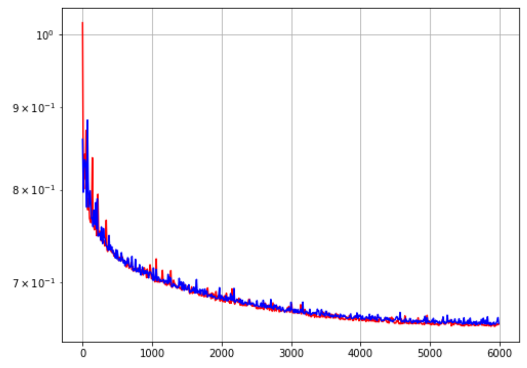


Figure 6.2: Decreasing model loss on the train set (red line) and validation set (blue line) during training.

Notice that the model loss on the validation set does not get structurally larger than on the loss on the train set, as it does for the neural network without dropout (see figure 6.1). It is therefore not necessary to stop the training process in order to prevent the network from overfitting the train data. The stopping criterion for training is defined as a fixed number of epochs. This number is empirically determined and set to 6000 epochs. It can be observed that the loss on both the test and validation set does not decrease significantly further.

The network with dropout has an AUC of 0.657 which is comparable to the performance of the regular network for which an AUC score of 0.663 is achieved.
Training a neural network with dropout typically takes about $2 - 3$ times longer than for a regular neural network. The major cause of the slower training is that the parameter updates are very noisy [Hinton et al., 2012]. Effectively, a different network is trained every epoch which causes the gradients, that represent the directions in which the parameters are updated, to be instable.

6.3.2 Re-scaling accuracy

As has been mentioned in section 6.1.2, re-scaling the probability relies on the assumption that the distribution within classes is not changed by removing observations. If this assumption does not hold, the undersampled dataset does favor some input variable settings. As a result, for example, certain

feature combinations that typically correspond to a 'No Prepayment' instance will barely be represented in the undersampled set. This could deteriorate the CPR predictions of the neural network in that region of the domain.
Since the AUC is based on accuracy of ordering, it does not inform whether the predictions are close to the observed probability. However, how close the average predicted CPR of a pool of mortgages is to the actual CPR is essential for a bank since large differences cause large unexpected cash-flows. Even if the order of predicted probability is accurate according to the AUC, it is important to get an accurate prediction for the fraction of prepayments within a group of mortgages.

For this reason, 5 million mortgages from the test set are ranked on their predicted prepayment rate. Subsequently the group is split in fifty pools of 100000 individual mortgages such that every pool contains enough Relocation events to calculate a useful average. The predicted CPR of a pool is calculated by the mean of the neural network's CPR predictions for the individual contracts in the pool. Each representing the predicted probability that a prepayment is observed in a certain month. The observed CPR is the fraction of observed Relocations in a pool. A comparison between the two can be seen in figure 6.3.

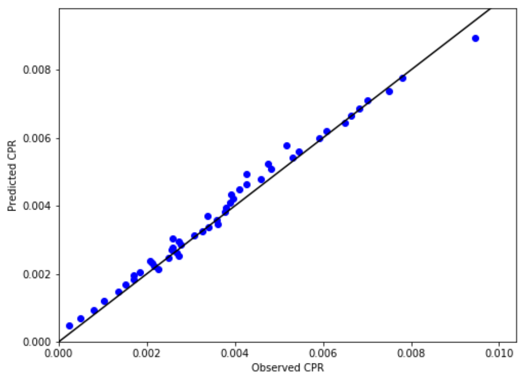


Figure 6.3: Accuracy of re-scaling predictions. The average predicted prepayment rates (vertical axis) for the pools are close to the observed prepayment rates (horizontal axis).

The plot shows that the re-scaling of predictions is accurate since the predictions are centered around the ideal line where the predicted prepayment rate equals the observed prepayment rate for each pool. This implies that the probabilities are re-scaled to the right order of magnitude which suggests the distributions within classes are not significantly changed by undersampling.

6.4 Relationship between estimated uncertainty and model performance

The samples from the trained neural network can be used to compute the mean prediction of an equivalent Bayesian neural network as well as an estimate for the uncertainty of that prediction. In this section we will study the relationship between the performance of a neural network trained with dropout and the uncertainty estimates computed by 100 samples generated by forward stochastic passes through the network.

In order to do this, we have split 5 million test set instances in fifty pools of equal size. Each pool represents a subset of the test set ranked on the estimated uncertainty obtained from the prediction samples. The first pool contains the 2% of the observations with the lowest estimated uncertainty (or, equivalently, the highest estimated confidence). The fiftieth pool contains the 2% of the observations with the highest estimated uncertainty. Splitting the set in fifty disjunct subsets provides enough pools to extract useful relationships but also ensures that enough observations (100000) are present within each pool in order to compute performance measures such as the AUC.

In the following subsections, we will discuss the differences in observed prepayment rate, AUC score and cross entropy loss for the pools. The plots are generated using data that can be found in table B.1 in appendix B.

Table 6.3 below shows a shorter version of table B.1 where the test instances are split into ten pools of 500000 contracts.

Pool	Uncertainty	Observed CPR	Predicted CPR	AUC	CE
1.0	0.000172	0.001512	0.00170	0.665	0.0110
2.0	0.000213	0.002134	0.00211	0.622	0.0150
3.0	0.000250	0.002450	0.00237	0.617	0.0170
4.0	0.000298	0.002518	0.00276	0.590	0.0175
5.0	0.000349	0.003190	0.00343	0.610	0.0213
6.0	0.000397	0.004224	0.00437	0.612	0.0270
7.0	0.000446	0.004532	0.00476	0.591	0.0288
8.0	0.000506	0.004890	0.00509	0.583	0.0307
9.0	0.000596	0.005482	0.00564	0.578	0.0338
10.0	0.000960	0.006334	0.00647	0.589	0.0381

Table 6.3: Differences in observed and predicted prepayment rates, AUC scores and cross entropy losses for ten pools arranged by their estimated uncertainty.

A noticeable detail is that there is a large difference between the value that represents the uncertainty estimates for pools 9 and 10. This increase will be discussed in section 6.7 where we will study the distribution of the uncertainty estimates for the test set instances.

6.4.1 Estimated uncertainty and prepayment rate

In table B.1 in appendix B, it can be observed that pools that are classified as relatively uncertain in general appear to have a higher observed prepayment rate than pools with predictions that are paired with lower estimated uncertainty. In figure 6.4, the observed and predicted prepayment rates are shown for the fifty pools defined above. The darker the color of the dot, the higher the average estimated uncertainty of the predictions in the corresponding pool.

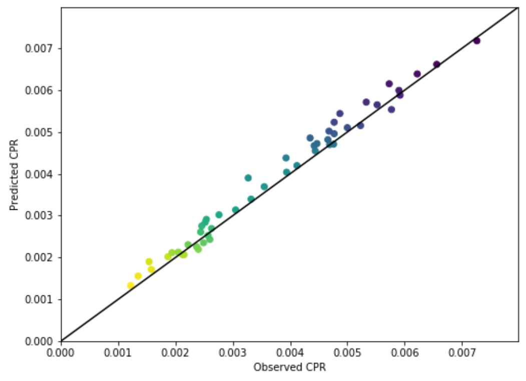


Figure 6.4: Observed prepayment rates compared to predictions for the fifty pools of test instances. The average predicted prepayment rates (vertical axis) for the pools are close to the observed prepayment rates (horizontal axis). The darkness of dots increases with the average estimated uncertainty of the pool which it represents.

It can be observed that the model predicts prepayment rates well for the pools of contracts. We saw this performance already in figure 6.3 where after ranking the instances on the predicted CPR. In figure 6.4 the pools are ranked on the estimated uncertainty. There we see that the accuracy is slightly lower for the darker dots that represent the pools with a high average estimated uncertainty. Note that these pools with high estimated uncertainty and less accurate CPR predictions correspond to the pools that have a high observed and predicted CPR.

6.4.2 Estimated uncertainty and AUC

We wish to evaluate how the uncertainty measure relates to the AUC (see section 6.2.1). Therefore, the AUC score is computed on the fifty pools. The values can be found in B.1 in appendix B. In figure 6.5, the AUC scores can be observed.

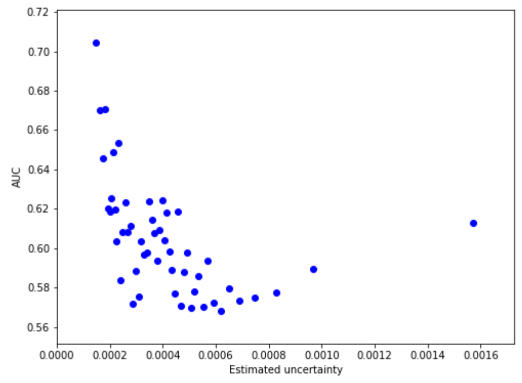


Figure 6.5: AUC for the fifty pools of test instances ordered on their uncertainty value. The horizontal axis shows the average standard deviation of 100 dropout samples for the instances contained in the pool. The vertical axis represents the AUC score on that particular pool.

A negative trend can be observed in the graph. This indicates that, in general, predicted CPRs for pools classified as uncertain are ordered less accurately than pools with predictions that are relatively certain.

We would like to stress that the AUC score on the full test set does not equal the average AUC score of each pool. This is a result from the definition of the AUC which is based on the relative ordering of probabilities. The model may have trouble ordering a pool of similar mortgages while it performs better on the full set which contains a more diverse contracts.

This reasoning also explains the relatively higher AUC for the two most uncertain pools. The observations in these pools are diverse since they are in extreme regions of the domain (see section 6.7). This makes it easier to order them.

6.4.3 Estimated uncertainty and cross entropy

Another way to investigate how well prepayment rates are predicted for the fifty pools, is to calculate the average cross entropy loss (see section 3.2.2) of the pool. In figure 6.6, the average cross entropy loss on the fifty pools is shown.

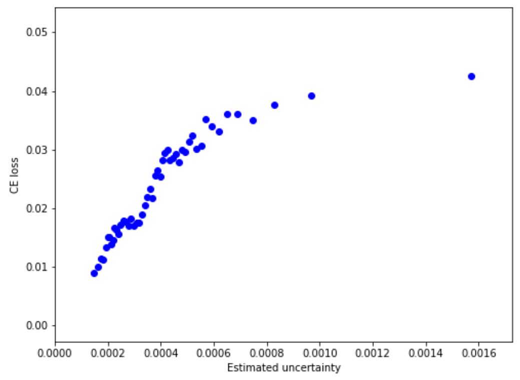


Figure 6.6: Cross entropy loss on the fifty pools of test instances ordered on their uncertainty value. The horizontal axis shows the average standard deviation of 100 dropout samples for the instances contained in the pool. The vertical axis represents the cross entropy loss on that particular pool.

It can be observed that the pools that have a higher estimated uncertainty in general have a higher cross entropy loss. Recall that the cross entropy loss is related to the log-likelihood and that a low cross entropy loss indicates a high similarity between the observed and predicted distribution. From figure 6.6, we deduce that the predictive distributions are more similar to the observed prepayment behavior for pools that have a low estimated uncertainty than pools with predictions that are classified as relatively uncertain.

6.5 Dealing with the most uncertain uncertain contracts

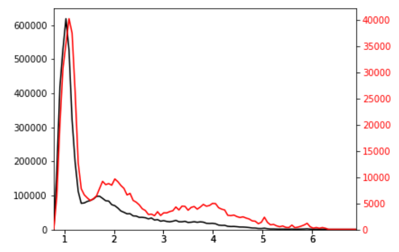
Now that the measure of uncertainty appears to be a good indicator for the quality of the prediction, we will focus on how regions with relatively high uncertainty can be identified. These regions are interesting since they are expected to be predicted worst. This knowledge could be useful for a bank to determine which types of mortgages are desired and which are not.
A bank may decide, for example, to avoid new contracts that have a high estimated uncertainty since they are expected to be predicted inaccurately. These contracts may require holding larger capital buffers to anticipate on the poorer CPR predictions. In that case it may be favorable to focus on new mortgages that are expected to be predicted more accurately.

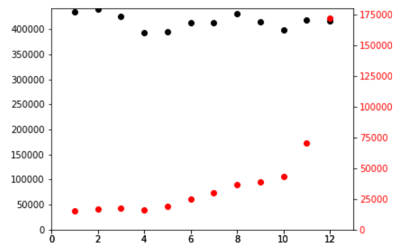
In this section, we will briefly make a characterization of the contracts that are paired with a high estimated uncertainty. In table 6.4, the average feature values of the full test set are compared to the 10% most uncertain contracts. This second set corresponds to the tenth pool considered in table 6.3.

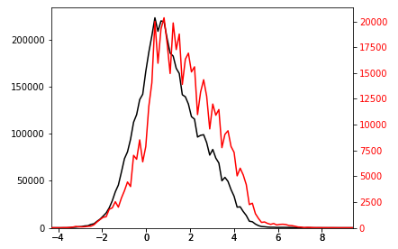
Feature	**Full test set**	10% **most uncertain**
HPI ratio	1.58	2.10
Current month	6.5	9.1
Interest rate incentive	1.01	1.60
Age of loan	106	168
Rem. interest rate maturity	70.7	24.3
AUC score	0.657	0.589

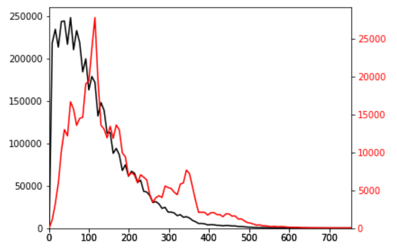
Table 6.4: Average feature value for the full test set compared to the 10% of mortgages with the highest estimated uncertainty.

It can be observed that there are clear differences between the contracts of both sets. As discussed in section 6.4, the pool of predictions classified as most uncertain has a lower AUC score than the full test set. The difference in mortgage characteristics between the sets can be observed better in figure 6.7 where the distributions of the five features are shown for both the full test set and the 10% most uncertain contracts.


(a) HPI ratio


(b) Month


(c) Interest rate incentive


(d) Age of loan (in months)

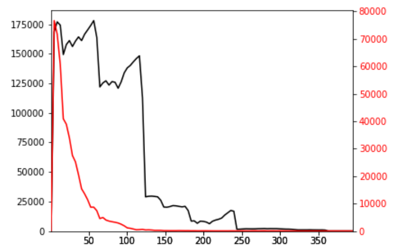

(e) Remaining interest rate maturity

Figure 6.7: Visualization of feature characteristics of 10% mortgages with highest estimated uncertainty. The number of mortgages in the most uncertain subset is represented by the red lines/dots. The number of mortgages in the full test set is represented by the black lines/dots.

Characteristics of the total test set are compared to characteristics of the 10% observations with highest expected uncertainty. Some interesting differences can be observed.
First of all, the instances with a relatively high HPI ratio change (in particular 3 and higher) are overrepresented in the uncertain category.
Towards the end of the year, predictions are estimated to be less certain. The month December in particular is highly overrepresented in the most uncertain set.
The subset containing contracts with predictions that are relatively uncertain appear to have, on average, a higher interest rate incentive than the full test set.
For the age of the loan, contracts with an age of more than 20 years are clearly paired with high estimated uncertainty.
Mortgages that have a short remaining interest rate maturity are likely to be prepayed. Also these mortgages are overrepresented in the most uncertain class.

In section 6.4, it was illustrated that predictions of the neural network are, on average, less accurate for the mortgages discussed above. It is reasonable to assume that a second model can be built that performs better on the uncertain subset. By defining a hybrid model that uses the neural network for the certain contracts the second model for the most uncertain contracts, a better performance on the full set would be attained.

6.5.1 Hybrid model

In this section we will, in addition to a neural network, use a second model that performs similarly on a test set. Then we will illustrate that a combination of those models can perform better than both individual models. The uncertainty estimates prove to be sufficient to identify which instances should be predicted by which model to construct the improved model.
A neural network and a logistic regression model are trained on the same data and made sure that their performance measured by the AUC on the test set is similar. Both models have an AUC score around 0.651.
Subsequently, both models generate CPR predictions for all 5 million contracts in the test set. Uncertainty estimates are computed by the neural network with dropout applied. The instances in the test set are ranked on their estimated uncertainty like in section 6.4.

Assume a hybrid model that uses predictions from both individual models. It uses the fraction $a \in [0, 1]$ of most certain predictions from the neural network with dropout. The remaining contracts are predicted by the logistic regression model. We have computed the AUC scores on the full test set for this hybrid model for different values of $a \in [0, 1]$. The results can be found in figure 6.8.

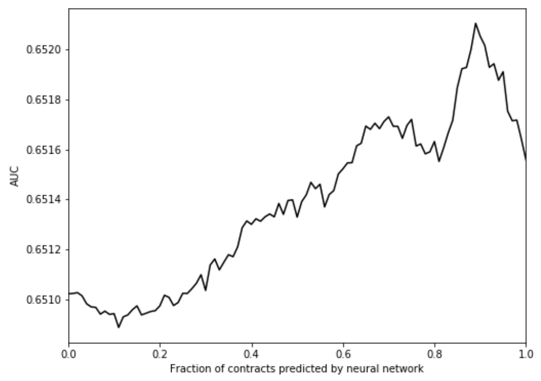


Figure 6.8: The AUC score on the test set of 5 million observations for different constructions of the hybrid model. The horizontal axis shows the fraction $a \in [0, 1]$ of most certain predictions that is used from the neural network. The vertical axis represents the AUC score on the full test set.

The figure shows that, measured by the AUC, a combination of models performs better than one of the individual models. Assume the model with $a = 0.9$ which means that we use the 90% most certain predictions from the neural network with dropout. For the remaining 10% of contracts, the predictions from the logistic regression are used. This model performs, measured by the AUC, better than both individual models. The neural network's predictions for the contracts visualized in figure 6.7 are substituted.

It can be concluded that the uncertainty estimates indicate which contracts are predicted inaccurately by the neural network. The performance of the suggested hybrid model is better compared to the individual models when the split is simply defined by the estimated uncertainty. This interesting result indicates that the weakness of a neural network in extrapolations can be partly covered by another model.

It is worth noticing that this hybrid model requires models that have similar performance since the improvement of performance is small. The improvement may however be valuable.

6.6 Contribution of variables

A neural network for predicting prepayment behavior is trained to recognize the influence of the different input variables. The trained network can be helpful to gain insight in how the features drive prepayment behavior. This section covers the contribution of those features to the (predicted) prepayment behavior.

6.6.1 Methodology

Sirignano et al. [2018] studied the sensitivity of prepayment behavior with respect to different variables. In order to extract the sensitivities, an 'average' contract is defined. This is a contract where the feature values equal the average of all the contracts in a set. The average contract for the test set used in this thesis can be found in table 6.4. From that contract, the CPR is calculated while keeping all but one feature at their average. The CPR is then predicted while the non-constant feature runs over its domain. Sensitivities with respect to the feature that is varied can be generated this way.
This method is not followed in this thesis since it gives rise to impossible feature combinations under our setup. For example, a new loan (with Loan Age 0) is impossible to be observed in combination with an average HPI ratio change of 1.58 since the HPI can not increase with 58% in 0 months time. The prediction for this contract is not a good reflection of the average CPR for contracts of age 0.

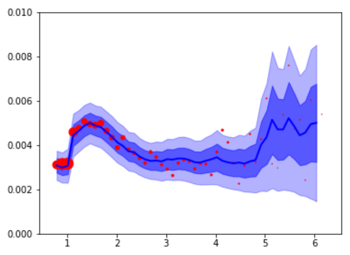
We therefore generate the sensitivies with respect to the five features by another method. For each feature, a partition of the domain is defined. The set of contracts is split into buckets using this partition. The first bucket contains contracts that have feature values in the first interval, second in the second interval, etc. The CPR is predicted for each contract in the buckets and their average represents the prediction for the feature value corresponding to the buckets. This can be repeated for each feature. The j-th bucket of feature i is defined as

$$X_j^i = \{\mathbf{x} \in X : z_{j-1} \leq x_i < z_j\}, \tag{6.6}$$

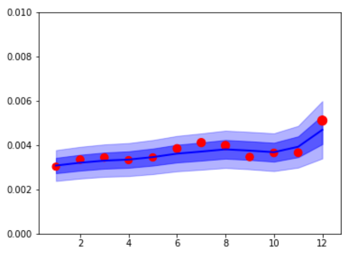
for $j = 1, \dots, J$ and $i = 1, \dots, d_0$. The element $J \in \mathbb{N}$ denotes the number of buckets constructed by the partition $\{z_0, \dots, z_J\}$ of the domain of the i-th feature, where $z_{j-1} < z_j$ for $j = 1, \dots, J$.
This method prevents predicting unrealistic contracts such as described above. The sensitivies are calculated by using only test set instances that have actually been observed. Every bucket contains mortgages for which the feature in question has a value within the specified interval. The other four features are spread over their domain. This method works best when each bucket X_n^i contains enough observations to cover the characteristics of contracts with the i-th feature in the interval $[z_{n-1}, z_n)$.
In addition to calculating the average predicted CPR for the buckets, the average estimated uncertainty is computed. This value gives an indication of the neural network's confidence in the predicted relationship.

6.6.2 Sensitivity of predicted CPR with respect to the features

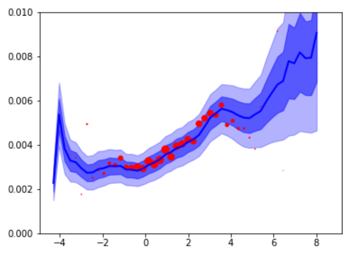
Figure 6.9 shows the relationship between features and the relocation CPR calculated by the method described above. The blue line is the prepayment rate predicted by the neural network. The red dots represent the observed fraction of relocations in the set of 5 million observations conditioned that the variable in question lies within a specified interval. The size of the dots represents the number of prepayments corresponding to the dot. The averaged estimated uncertainty regarding the predictions is visualized by the blue bands.



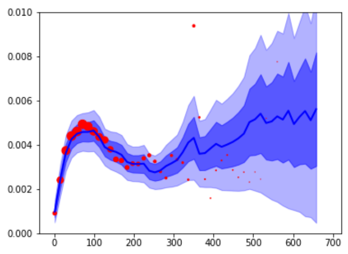
(a) HPI ratio



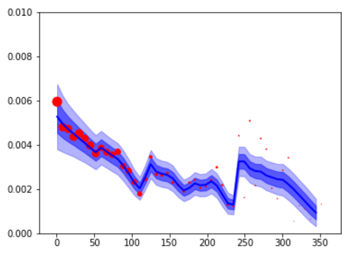
(b) Month



(c) Interest rate incentive



(d) Age of loan (in months)



(e) Remaining interest rate maturity

Figure 6.9: Predictions and uncertainty estimates generated by 100 posteriors for the Relocation rate. The red dots represent the observed fraction of Relocations.

The prepayment behavior appears to be captured well. Especially in the regions with relatively many observations. These regions are characterized by large red dots.
These predictions can be used to determine the sensitivity of prepayment behavior with respect to the five features. The sensitivities are useful since they give insight into how the value of outstanding mortgages is expected to change under different market scenarios.
As an example, if the market interest rate goes up, it is important for a bank to anticipate on how this is expected affect the prepayment behavior of the clients. Figure 6.9c visualizes how interest rate changes are expected to affect the CPR.
Also, this method allows the user to observe easily under what circumstances uncertainty regarding predictions is expected to increase of decrease. This could be useful in quantification of long-term expectations. In figures 6.9a, 6.9c and 6.9d, it can be clearly seen that the estimated uncertainty is larger in regions where relatively few observations are present. This is in accordance with the results

from chapter 5 where we noticed that predictions in those regions are expected to be less accurate than where more observations are present.

6.7 Distribution of uncertainty estimates

Section 6.4 illustrated that the estimated uncertainty is an indicator for the performance of the model. We would like to conclude this chapter with a discussion of how the uncertainty estimates are distributed for the test instances. In figure 6.10, the values of the uncertainty estimates can be observed for the test instances.

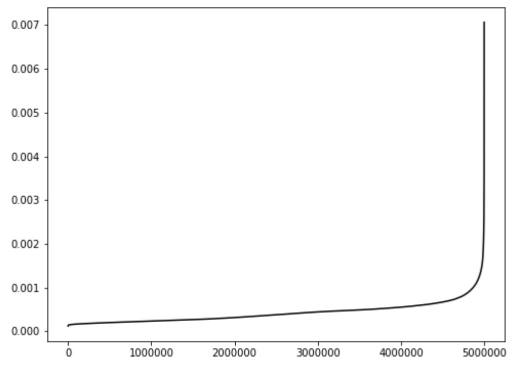


Figure 6.10: The distribution of the uncertainty estimates for the 5 million observations in the test set. The horizontal axis shows the observation number ranked on the estimated uncertainty for the predictions. The vertical axis represents the estimated uncertainty.

It can be observed that uncertainty estimates are close to each other for most instances of the test set. For the contracts with highest uncertainty, there are large differences. This is caused by the distribution of the dataset. This is clearly (though one-dimensionally) visualized in figures 6.9a, 6.9c and 6.9d.

The majority of the observations are close to each other in a certain region of the domain. The neural network has learned the relationships between the features and the CPR well in this region. The estimated uncertainty is therefore low. The further an observation is from this known region, the less reliable the prediction for that observation is expected to be. For these observations, there are fewer instances from the train set in the neighbourhood on which a prediction can be based. This is reflected by a higher estimated uncertainty. The number of highly unreliable predictions is small since, by the property of the uncertainty estimates, they are rare. Therefore only a small fraction of predictions for observations in the test set is identified as highly uncertain.

Chapter 7

Conclusion and future research

In this thesis, the possibility to generate reliable uncertainty estimates for neural networks has been studied. First for data generated by known functions and later for predicting conditional prepayment rates. In this chapter the results are summarized. Recommendations for future research can be found at the end of the section.

Conclusion

In section 5.2, it was shown that the distribution of the observations in the train set influences the performance of a neural network. We illustrated that it is essential for the user of a model to acknowledge that some predictions are more accurate than others. By generating an indication of the uncertainty regarding predictions, it is possible to anticipate on inaccurate predictions. Bayesian neural networks do, in contrast to most machine learning methods, allow the user to obtain uncertainty estimates for predictions. In a Bayesian neural network, distributions over the model parameters are considered. This way, distributions over the outputs can be obtained. A drawback of this method is that it is computationally very expensive.
In chapter 4, the proof in Gal [2016] was followed to connect a neural network trained with dropout to a Bayesian neural network. In addition to the original proof, the optimization objectives were evaluated including biases. Moreover, we studied the influence of hyperparameters and underlying assumptions of the relationship. The proof justified obtaining uncertainty estimates by computing stochastic forward passes through a neural network trained with dropout. In contrast to Bayesian neural networks, the uncertainty estimates obtained using this method were obtained computationally efficient. Like Sirignano et al. [2018] and Saito [2018], we were able to model prepayment behaviour using artificial neural networks, but, in addition to the predictions, we could extract information about the model uncertainty from the model itself.

In section 5.3, three simple but relevant functions were studied. It could be concluded that uncertainty estimates computed using dropout were accurate indicators for the knowledge about the underlying function in a certain region. Especially predictions towards the domain boundaries, where the neighbourhood was barely covered by the train set, were classified as uncertain.
In section 5.3.4 however, a setup was discovered where the method for obtaining uncertainty estimates did not work well. Model uncertainty was highly underestimated within a region of the domain. The uncertainty estimates computed by the method using dropout did not capture the information gap on the domain. As a solution, an ensemble was used in these situations. These kinds of scenarios were not studied further since they do not arise often naturally.
The results for the three simple functions in section 5.3 were extended in section 6.4 where uncertainty estimates obtained from a neural network trained with dropout to predict Relocation behavior were studied. Predictions classified as relatively uncertain were on average worse than predictions that were classified as certain. This result was confirmed by measuring performance using the AUC score, the cross entropy loss and the difference between the observed and predicted CPR.
In accordance with our earlier results from section 5.3, the predictions that were classified as most uncertain were located in regions with fewer observations. It appeared that the train set did not contain enough similar contracts to generate a confident prediction. We concluded that a neural network trained with dropout to predict CPRs was able to indicate which contracts were predicted poorly. These predictions were substituted by the predictions from a model that was more robust to outliers. Using this substitution, we managed in section 6.5.1 to improve the performance of a neural network by discarding the most uncertain predictions.
Sirignano et al. [2018] and Saito [2018] studied the sensitivity of the predicted CPR with respect to different features. In section 6.6, we were able to extract the sensitivity of the uncertainty regarding predictions as well.

Future research

The conclusions about the usefulness of uncertainty estimates in section 6.4 are based on a relative difference between the uncertainty estimates. The confidence in this thesis is meant to indicate which contracts are expected to be predicted relatively well and which are not. No information is provided about how confident, in an absolute sense, the model is about a prediction.
We are aware that it is more useful when measures can be extracted from a model that give insight in the expected error that is made. This would require a clear distinction between epistemic and aleatoric uncertainty. Both may be modelled separately to compute a measure for the total uncertainty which can, in turn, be transformed to actual confidence intervals for the predictions.

A setup where uncertainty regarding predictions was underestimated is treated in section 5.3.4. This problem arose when a there was a lack of data within the one-dimensional domain. It could be interesting to study if this problem also arises in other scenarios. This may lead to a better understanding of the uncertainty estimates.

In section 6.5, we discussed the characteristics of contracts whose predictions are paired with high uncertainty estimates. It appears that these mortgages tend to have rare characteristics such as a relatively high age of the loan. In a one-dimensional sense, this conclusion was confirmed in figure 6.9. It would be interesting to get a deeper understanding of where observations that are classified as uncertain are located compared to the the other observations in the dataset. This requires an examination of the neighbourhood of those observations in their (in this case five-dimensional) domain.

Bibliography

S. Asmussen, D. Kroese, and R. Rubinstein. Heavy tails, importance sampling and cross–entropy. 2003.

D. Barber and C. Bishop. Ensemble learning in bayesian neural networks. *Nato ASI Series F Computer and Systems Sciences*, 168:215–238, 1998.

J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

C. Bishop. Pattern recognition and machine learning. *Springer*, 2006.

A. Bradley. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern Recognition, Vol. 30, No. 7, pp. 1145-1159*, 1997.

D. Ciresan, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classification. *CoRR*, abs/1202.2745, 2012. URL `http://arxiv.org/abs/1202.2745`.

G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 1989.

J. Davis and I. Dhillon. Differential entropic clustering of multivariate gaussians. In *Advances in Neural Information Processing Systems*, pages 337–344, 2007.

P. De Boer, D. Kroese, S. Mannor, and R. Rubinstein. A tutorial on the cross-entropy method. *Annals of operations research*, 134(1):19–67, 2005.

A. Der Kiureghian and O. Ditlevsen. Aleatory or epistemic? does it matter? *Structural Safety*, 31(2): 105–112, 2009.

T. Dietterich. Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, pages 1–15. Springer, 2000.

D. Donoho. Aide-memoire. high-dimensional data analysis: The curses and blessings of dimensionality. 2000.

R. Eldan and R. Shamir. The power of depth for feedforward neural networks. *Weizmann Institute of Science*, 2015.

S. Fahlman and C. Lebiere. The cascade-correlation learning architecture. In *Advances in neural information processing systems*, pages 524–532, 1990.

Y. Gal. *Uncertainty in Deep Learning*. PhD thesis, University of Cambridge, 2016.

Y. Gal and Z. Ghahramani. Dropout as a Bayesian approximation: Insights and applications. In *Deep Learning Workshop, ICML*, 2015a.

Y. Gal and Z. Ghahramani. On modern deep learning and variational inference. In *Advances in Approximate Bayesian Inference workshop, NIPS*, 2015b.

Y. Gal and Z. Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *Proceedings of the 33rd International Conference on Machine Learning (ICML-16)*, 2016.

M. Gardner and S. Dorling. Artificial neural networks (the multilayer perceptron) - a review of applications in the atmospheric sciences. *School of Environmental Sciences, University of East Anglia, Norwich*, 1998.

M. Gevrey, I. Dimopoulos, and S. Lek. Review and comparison of methods to study the contribution of variables in artificial neural network models. *Ecological Modelling 160 (2003), 249-264*, 2002.

M. Gevrey, I. Dimopoulos, and S. Lek. Two-way interaction of input variables in the sensitivity analysis of neural network models. *Ecological Modelling 195 (2006) 43–50*, 2006.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*, volume 1. MIT press Cambridge, 2016.

D. Graupe. *Principles of artificial neural networks*, volume 7. World Scientific, 2013.

L. Hansen and P. Salamon. Neural network ensembles. *IEEE transactions on pattern analysis and machine intelligence*, 12(10):993–1001, 1990.

G. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

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

K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Technische Universität Wien*, 1989.

D. Hunter, H. Yu, M. Pukish, J. Kolbusz, and B. Wilamowski. Selection of proper neural network sizes and architectures—a comparative study. *IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS*, page 11, 2012.

N. Japkowicz and S. Stephen. The class imbalance problem: A systematic study. *School of Information Technology and Engineering, University of Ottawa*, 2002.

A. Kendall and Y. Gal. What uncertainties do we need in bayesian deep learning for computer vision? *Advances in neural information processing systems*, 2017.

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

S. Kotsiantis, D. Kanellopoulos, and P. Pintelas. Handling imbalanced datasets: A review. *GESTS International Transactions on Computer Science and Engineering, Vol.30*, 2006.

B. Kuijenhoven, B. Maarse, and E. Driessen. Dutch residential mortgage interest-typical prepayment 7.0 model. *ABN AMRO*, 2016.

S. Kullback and R. Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22 (1):79–86, 1951.

S. Lek, M. Delacoste, P. Baran, I. Dimopoulos, J. Lauga, and S. Aulagnier. Application of neural networks to modelling nonlinear relationships in ecology. *Ecological Modelling 90 (1996) 39-52*, 1994.

J. Leonard, M. Kramer, and L. Ungar. A neural network architecture that computes its own reliability. *Computers chem. Engng, Vol. 16, No. 9, pp. 819-835, 1992*, 1992.

D. MacKay. A practical bayesian framework for backprop networks. *Neural computation 4.3*, 1992.

M. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.

F. Pardoel. Penalty model. *ABN AMRO*, 2015.

S. Perry, S. Robinson, and J. Rowland. A study of mortgage prepayment risk. *ABN AMRO*, 2015.

A. Dal Pozzolo, O. Caeleny, R. Johnsonz, and G. Bontempi. Calibrating probability with undersampling for unbalanced classification. *Machine Learning Group, Computer Science Department, Universite Libre de Bruxelles, Brussels, Belgium.*, 2015.

D. Rumelhart, G. Hinton, and R. Williams. Learning representations by backpropagation error. *Nature 323, 533-536*, 1986.

J. Saez, B. Krawczyk, and M. Wozniak. Analyzing the oversampling of different classes and types of examples in multi-class imbalanced datasets. *PatternRecognition57(2016)164–178*, 2016.

T. Saito. Mortgage prepayment rate estimation with machine learning. *Technische Universiteit Delft*, 2018.

C. Shannon. A mathematical theory of communication. *Bell system technical journal*, 27(3):379–423, 1948.

J. Sirignano, A. Sadhwani, and K. Giesecke. Deep learning in mortgage risk. *Cornell University*, 2018.

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1): 1929–1958, 2014.

N. Taleb. *The black swan: The impact of the highly improbable*, volume 2. Random house, 2007.

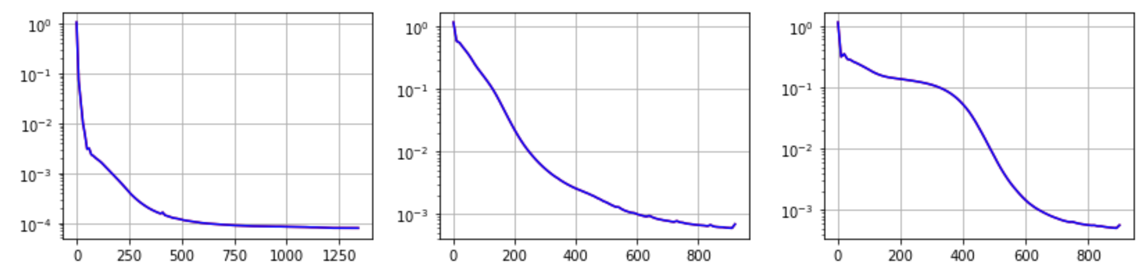
S. Xiang, F. Nie, and C. Zhang. Learning a mahalanobis distance metric for data clustering and classification. *Pattern Recognition*, 41(12):3600–3612, 2008.

Appendix A

Decrease of error on test set and validation set

In the following figures, the decrease of the error on the train and validation sets can be observed under the different setups discussed in section 5.2. Note that the errors on both sets strongly depend on how the observations in the train set are distributed.

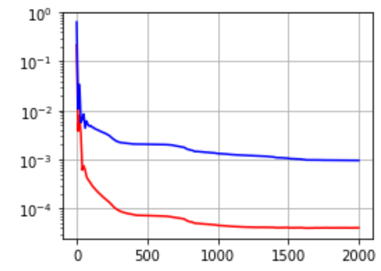
A.1 Evenly distributed observations



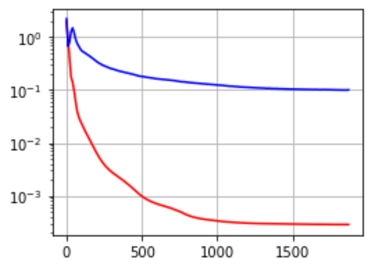
(a) Function 1 (b) Function 2 (c) Function 3

Figure A.1: Decrease of the Mean Squared Error (MSE) on the train set while training the three functions. The epochs are on the horizontal axis and the MSE on the vertical axis. Since the train set is evenly distributed on the interval, the error on the train set equals the error on the validation set.

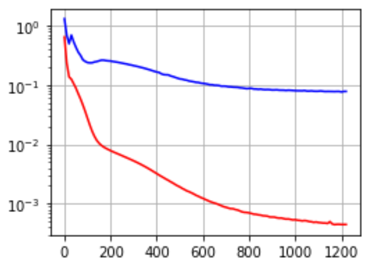
A.2 Standard normally distributed observations



(a) Function 1



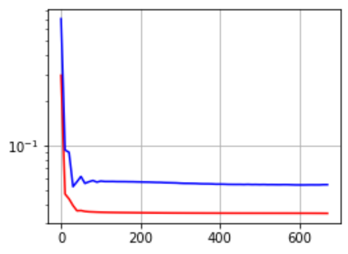
(b) Function 2



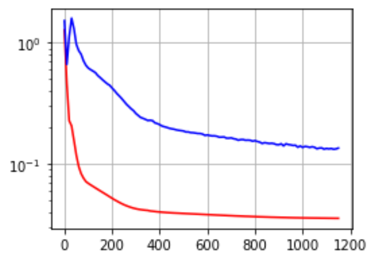
(c) Function 3

Figure A.2: Decrease of the Mean Squared Error (MSE) on the train set (red line) and validation set (blue line) while training the three functions. The epochs are on the horizontal axis and the MSE on the vertical axis. The training is stopped when the error on the validation set does not decrease further.

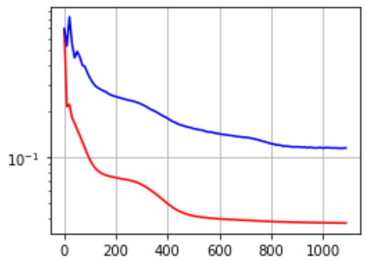
A.3 Standard normally distributed observations with white noise



(a) Function 1



(b) Function 2



(c) Function 3

Figure A.3: Decrease of the Mean Squared Error (MSE) on the train set (red line) and validation set (blue line) while training the three functions. The epochs are on the horizontal axis and the MSE on the vertical axis. The training is stopped when the error on the validation set does not decrease further.

Appendix B

Uncertainty and model performance

Relationships between the prediction of a neural network trained with dropout and the uncertainty estimates computed by 100 samples generated by forwards stochastic passes through the network. 5 million test set instances are split in fifty pools of equal size. Each pool represents a subset of the test set ranked on the estimated uncertainty obtained from the posteriors.
Table B.1 below shows the differences in observed prepayment rate, AUC score and cross entropy loss for the pools. The data is used in section 6.4.

Pool	Uncertainty	Observed CPR	Predicted CPR	AUC	CE
1.0	0.000147	0.00122	0.00133	0.704	0.0090
2.0	0.000163	0.00135	0.00156	0.670	0.0100
3.0	0.000173	0.00158	0.00171	0.646	0.0115
4.0	0.000183	0.00154	0.00190	0.671	0.0113
5.0	0.000192	0.00187	0.00202	0.620	0.0134
6.0	0.000199	0.00215	0.00207	0.619	0.0151
7.0	0.000206	0.00213	0.00207	0.625	0.0150
8.0	0.000213	0.00194	0.00211	0.649	0.0138
9.0	0.000219	0.00205	0.00213	0.620	0.0145
10.0	0.000226	0.00240	0.00220	0.603	0.0167
11.0	0.000234	0.00237	0.00226	0.653	0.0164
12.0	0.000241	0.00222	0.00231	0.584	0.0157
13.0	0.000250	0.00249	0.00235	0.608	0.0172
14.0	0.000258	0.00260	0.00243	0.623	0.0178
15.0	0.000268	0.00257	0.00253	0.608	0.0177
16.0	0.000277	0.00244	0.00261	0.611	0.0169
17.0	0.000288	0.00263	0.00269	0.572	0.0182

Continued on next page

Pool	Uncertainty	Observed CPR	Predicted CPR	AUC	CE
18.0	0.000298	0.00246	0.00276	0.588	0.0171
19.0	0.000308	0.00252	0.00284	0.575	0.0175
20.0	0.000318	0.00254	0.00291	0.604	0.0176
21.0	0.000329	0.00276	0.00302	0.597	0.0189
22.0	0.000339	0.00305	0.00314	0.598	0.0205
23.0	0.000350	0.00332	0.00340	0.624	0.0219
24.0	0.000360	0.00355	0.00369	0.614	0.0233
25.0	0.000369	0.00327	0.00390	0.608	0.0218
26.0	0.000379	0.00394	0.00404	0.594	0.0255
27.0	0.000388	0.00412	0.00420	0.609	0.0264
28.0	0.000397	0.00393	0.00438	0.625	0.0253
29.0	0.000407	0.00444	0.00455	0.604	0.0282
30.0	0.000416	0.00469	0.00470	0.618	0.0294
31.0	0.000425	0.00476	0.00471	0.598	0.0299
32.0	0.000435	0.00442	0.00467	0.589	0.0282
33.0	0.000445	0.00447	0.00472	0.577	0.0285
34.0	0.000456	0.00466	0.00482	0.618	0.0293
35.0	0.000468	0.00435	0.00486	0.571	0.0279
36.0	0.000480	0.00477	0.00496	0.588	0.0300
37.0	0.000492	0.00468	0.00502	0.598	0.0295
38.0	0.000506	0.00500	0.00510	0.570	0.0313
39.0	0.000520	0.00523	0.00515	0.578	0.0325
40.0	0.000535	0.00477	0.00523	0.586	0.0301
41.0	0.000552	0.00487	0.00544	0.570	0.0307
42.0	0.000571	0.00577	0.00554	0.594	0.0352
43.0	0.000592	0.00552	0.00565	0.572	0.0341
44.0	0.000618	0.00533	0.00571	0.568	0.0331
45.0	0.000649	0.00592	0.00588	0.579	0.0361
46.0	0.000690	0.00590	0.00599	0.573	0.0360
47.0	0.000746	0.00573	0.00615	0.575	0.0351
48.0	0.000829	0.00622	0.00639	0.577	0.0376
49.0	0.000966	0.00656	0.00662	0.589	0.0392
50.0	0.001571	0.00726	0.00718	0.613	0.0425

Table B.1: Differences in observed and predicted prepayment rates, AUC scores and cross entropy losses for fifty pools arranged by their estimated uncertainty.