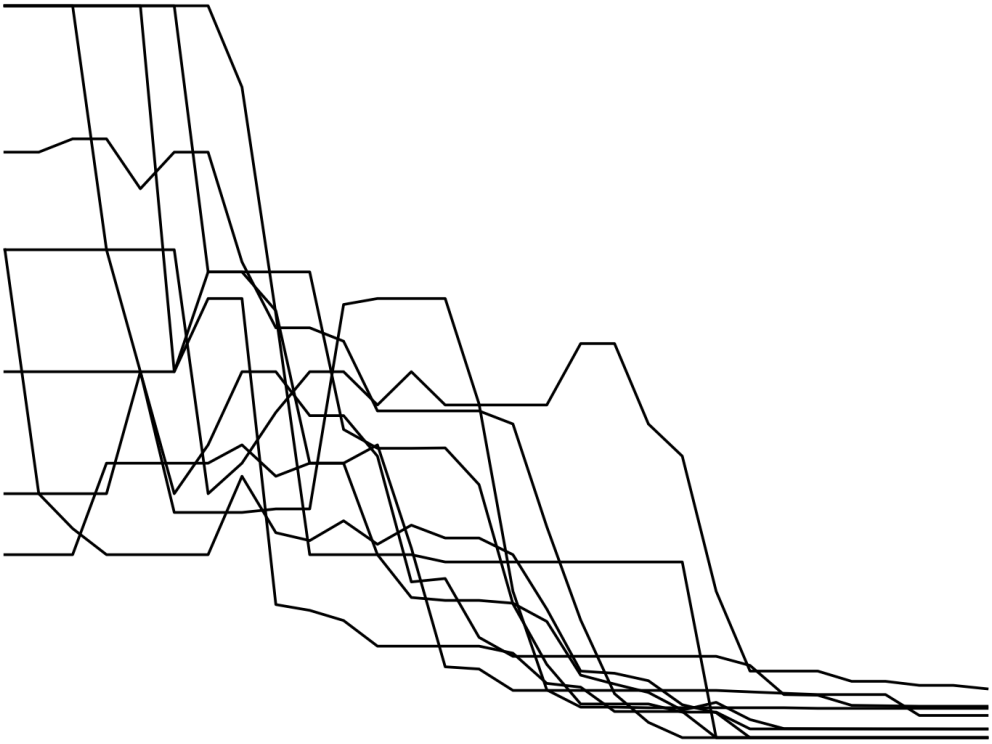

Forecasting Booking Behaviour in the Freight Ferry Industry

An Artificial Neural Network Approach

Master's thesis in Engineering Mathematics and Computational Science

ARVID RÄNKESKOG

DEPARTMENT OF MATHEMATICAL SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2023
www.chalmers.se

Master's thesis 2023

Forecasting Booking Behaviour in the Freight Ferry Industry

An Artificial Neural Network Approach

ARVID RÄNKESKOG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mathematical Sciences
Chalmers University of Technology
Gothenburg, Sweden 2023

Forecasting Booking Behaviour in the Freight Ferry Industry
An Artificial Neural Network Approach
Arvid Ränkeskog

Supervisors:
Alfred Olsson and Christian Söyland, Stena Line
Johan Jonasson, Mathematical Sciences
Examiner: Stefan Lemurell, Mathematical Sciences

Master's Thesis 2023
Department of Mathematical Sciences
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Visualisation of 10 time series with the percentage of active bookings that were cancelled, transferred or did not show up before the departure of the corresponding sailing. The timeline is decreasing from left to right in the figure.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2023

Forecasting Booking Behaviour in the Freight Ferry Industry
An Artificial Neural Network Approach
Arvid Ränkeskog
Department of Mathematical Sciences
Chalmers University of Technology

Abstract

The freight ferry industry operates in a stochastic environment where the arrival of vehicles on the departure day is uncertain. This thesis aimed to explore the application of artificial neural networks (ANNs) to predict whether active bookings at a given time before departure are going to become transferred, cancelled, or no-show. The sum of these bookings at a given time before departure was assumed to follow a Poisson binomial random variable. A comparative analysis was conducted between ANNs and Gradient Boosting Trees (LightGBM) trained on structured booking data, such as customer number and check-in status, and recurrent neural networks (RNNs) trained on a sequence of previous cumulative transfer and cancellation rates. The findings revealed that LightGBM was the better option to predict the mean of the distribution, while an ensemble of ANNs demonstrated promising potential in predicting the distribution's variance and error estimates.

Keywords: artificial neural networks, machine learning, poisson-binomial probability distribution, entity embedding, recurrent neural networks, revenue management, overbooking.

Acknowledgements

Firstly, I would like to express my gratitude to my supervisors, Alfred Olsson and Christian Söyland at Stena Line, and Johan Jonasson at Chalmers University of Technology, for their guidance, insightful discussions, and expertise. I would also like to thank Ken Ryrbo at Stena Line for providing me with the opportunity to work on this thesis project. Last but not least, I would like to express my sincere gratitude to my friends and family for all the support.

Arvid Ränkeskog, Flogned, May 2023

Contents

List of Figures

List of Tables

1
Introduction

This chapter starts with a background to the problem addressed in the thesis. Subsequently, the aim, scope and ethical considerations are presented.

1.1 Background

Revenue management, also known as yield management, is the practice of maximising revenue in a stochastic, or unpredictable, environment [1]. The term originates from the airline industry and has been extensively studied for the past 50 years, particularly in the context of airlines [1], [2]. Initially, the majority of research focused on overbooking, which involves accepting more bookings than the actual capacity of the airline due to uncertainty in customer show-up rates. The objective is to maximise revenue and profits while minimising customer dissatisfaction.

Despite the importance of sea freight for global trade, revenue management and overbooking are not as well-established in the shipping industry as in the airline industry [3]. In 2021, the total gross weight of goods handled in EU ports was estimated at 3.5 billion tonnes, making sea freight the most common mode of transportation for long-distance transport of goods to and from the EU in terms of tonnage [4]. However, uncertainty around no-shows and cancellations in the shipping industry remains a common problem. Shippers are not typically penalised for booking a certain volume of cargo and then failing to send it, and carriers are not penalised for failing to ship cargo that they have promised to ship [3]. In 2018, the New York Shipping Exchange and Boston Consulting Group estimated the worldwide cost of no-shows for container liner shippers to be $\$30 - 40$ billion per year [3]. One potential solution to this problem is to introduce monetary penalty fees for both parties. However, when Maersk Line, one of the world's largest container shipping companies, implemented this concept in 2011, it was deemed unsuccessful shortly thereafter [3].

In recent years, research papers on the prediction of cancellation rates in the container liner shipping industry by using various methods of statistical learning have been presented [5]–[7]. Wang and Meng [7] proposed a stacked model of three models to forecast the realised aggregated cargo of a departure: a piece-wise linear regression model, an autoregressive model and an artificial neural network. The resulting forecasting model was therefore a weighted model and the artificial neural network was the most influential of the three models. The data consisted of intercontinental

container liner shipping routes, including routes between Asia and the west coast of North America, Asia and the east coast of North America, as well as between Asia and Europe. Zhao, Meng, and Wang [6] compared linear regression and artificial neural networks for predicting cancellation rates for intercontinental container liner shipping between Asia and the west coast of the United States. The authors used data from individual bookings, and thus not at an aggregated departure level.

To the author's knowledge, less research has been published on the prediction of cancellations and no-show bookings in the sea freight industry of vehicles. It is an industry that faces the same problem regarding the uncertainty of no-shows and cancellations with a similar industry standard regarding penalty fees as the container liner shipping industry.

The aim of this thesis is to investigate the possibility of using machine learning with artificial neural networks to predict future no-shows, transfers and cancellations in the industry of sea freight of vehicles, such as cars, trucks and trains. These types of bookings are hereafter referred to as non-shippable bookings. The predictive model could in practice be used to simplify the decision of whether to accept or decline a booking inquiry at a given time before the departure to maximise expected revenue and profits.

1.2 Aim and Scope

As mentioned in Section 1.1, this thesis aims to investigate the possibility of using machine learning with artificial neural networks to predict future non-shippable bookings in the sea freight industry of vehicles. More specifically, at a given time before the departure of the vessel, the goal is to predict whether an active booking — a booking that takes up capacity of a vessel — is a future non-shippable booking. It is important to note that the specific time when a booking is first considered non-shippable is not predicted in this thesis. The prediction is whether the booking is going to be non-shippable sometime before the time of departure.

The thesis is written in collaboration with the shipping company Stena Line, which is a part of the Stena Sphere. Stena Line provided the data used in this project, which includes several ferry routes for vehicle transportation in Europe.

1.3 Ethical considerations

The predictive model developed in this thesis could lead to more heavily booked vessels, resulting in the possibility of rolled cargo — cargo that is not loaded on the vessel it is intended to sail on due to overbooking. This could lead to a loss of customer goodwill and reliability [1], [3]. It should be noted that the predictive model's results may not necessarily generalise to other data sets.

2
Theory

This chapter begins by describing the Poisson-binomial distribution and some of its properties. Thereafter, some theoretical concepts of artificial neural networks are introduced, including the subclass recurrent neural networks. The chapter ends with the presentation of the benchmark model and the evaluation metrics used in the thesis.

2.1 The Poisson-binomial distribution

The Poisson-binomial distribution is defined as the distribution of the sum of independent Bernoulli distributed indicators I_i [8]. Each indicator has a corresponding probability of success p_i. Thus in mathematical terms, this can be expressed by $\Pr(I_i = 1) = p_i = 1 - \Pr(I_i = 0)$. Therefore, the probability mass function (PMF) of I_i can be given by:

$$p_{I_i}(k_i) = Pr[I_i = k_i] = p_i^{k_i}(1 - p_i)^{1-k_i}, \ for \ k_i \in \{0, 1\}. \tag{2.1}$$

The mean and variance of I_i are:

$$\mu_{I_i} = E[I_i] = p_i, \tag{2.2}$$

$$\sigma_{I_i}^2 = Var[I_i] = p_i(1 - p_i). \tag{2.3}$$

Considering Equation (2.1), the PMF of the Poisson-binomial random variable $N = \sum_i I_i$ can be expressed using the formula presented in [9]:

$$p_N(k) = Pr[N = k] = \sum_{A \in F_k} \prod_{i \in A} p_i \prod_{j \in A^c} (1 - p_j). \tag{2.4}$$

Equation (2.4) can be used to calculate the probability of having k successful events out of n trials. F_k is the set of all integer subsets of size k and A refers to one of its subsets. A^c refers to the complement of A. To clarify with an example: if $n = 4$, then $F_1 = \{\{1\}, \{2\}, \{3\}, \{4\}\}$. For instance, A could be equal to $\{1\}$ and this means that A^c is in this case equal to $\{2, 3, 4\}$. This corresponds to calculating the probability of having exactly one successful event out of four trials with individual probability of success.

Because of the linearity of the mean and the assumption of independence of the indicators, the mean and variance of N can be obtained with the following formulas:

$$\mu_N = E[\sum_i I_i] = \sum_i E[I_i] = \sum_i p_i, \tag{2.5}$$

$$\sigma_N^2 = Var[\sum_i I_i] = \sum_i Var[I_i] = \sum_i p_i(1-p_i). \tag{2.6}$$

In this thesis, the random variable $X_i = l_i I_i$ is of interest, where l_i is a constant. The mean and variance of $X = \sum_i X_i$ is given by:

$$\mu_X = E[\sum_i l_i I_i] = \sum_i l_i E[I_i] = \sum_i l_i p_i, \tag{2.7}$$

$$\sigma_X^2 = Var[\sum_i l_i I_i] = \sum_i l_i^2 Var[I_i] = \sum_i l_i^2 p_i(1-p_i). \tag{2.8}$$

It can be computationally expensive to use the Poisson-binomial distribution in practice, and as a result, approximation methods are often applied [8]. One common approximation is the normal approximation, which assumes that X follows a normal distribution $N(\mu_X, \sigma_X^2)$ based on the central limit theorem. This approximation can be reliable when the number of samples is sufficiently large. However, when the number of samples is small, the normal approximation can yield poor results. Hong [8] recommends the Refined normal approximation, first proposed by [10] in 1996, to correct for the skewness of the distribution of N,

$$\gamma_N = \frac{E[(N-\mu_N)^3]}{\sigma_N^3} = \frac{E[(\sum_i (I_i - \mu_{I_i}))^3]}{\sigma_N^3} = \frac{\sum_i p_i(1-p_i)(1-2p_i)}{\sigma_N^3}, \tag{2.9}$$

but this approximation method will not be presented in detail in this thesis. The last equality in Equation (2.9) holds because of two reasons: the independence assumption of the indicators I_i and the fact that the third central moment is a cumulant. Therefore holds the additive property in Equation (2.9) [11]. Cumulants of a probability distribution are similar to its moments. To mention a few cumulants: the first is the mean, the second is the variance and the third is the third central moment [12], and the third central moment of I_i is $p_i(1-p_i)(1-2p_i)$.

The skewness of X can therefore be defined by:

$$\gamma_X = \frac{E[(X-\mu_X)^3]}{\sigma_X^3} = \frac{E[(\sum_i l_i (I_i - \mu_{I_i}))^3]}{\sigma_X^3} = \frac{\sum_i l_i^3 p_i(1-p_i)(1-2p_i)}{\sigma_X^3}, \tag{2.10}$$

since the following holds:

$$E[(l_i(I_i - \mu_{I_i}))^3] = E[l_i^3 (I_i - \mu_{I_i})^3] = l_i^3 E[(I_i - \mu_{I_i})^3]. \tag{2.11}$$

In this thesis is p_i estimated with $\hat{p}_i$ from an artificial neural network approach. Given these estimates can μ_X, σ_X^2 and γ_X be estimated, and since these estimates can vary significantly, an averaged parameter estimation is often preferred [13]. In terms of N samples and M instances:

$$\hat{\mu}_X = \frac{1}{M} \sum_{m=1}^{M} \sum_{i=1}^{N} l_i \hat{p}_{im}, \tag{2.12}$$

$$\hat{\sigma}_X^2 = \frac{1}{M} \sum_{m=1}^{M} \sum_{i=1}^{N} l_i^2 \hat{p}_{im}(1-\hat{p}_{im}), \tag{2.13}$$

$$\hat{\gamma}_X = \frac{\frac{1}{M}\sum_{m=1}^{M}\sum_{i=1}^{N} l_i^3 \hat{p}_{im}(1-\hat{p}_{im})(1-2\hat{p}_{im})}{\hat{\sigma}_X^3}. \tag{2.14}$$

The probabilities could be considered as unknown and as random variables P_i. Therefore, the conditional random variable $X|P_1, P_2, ..., P_N$ becomes of interest. In this context, the law of total variance can be applied [13]:

$$Var[X] = E[Var[X|P_1, P_2, ..., P_N]] + Var[E[X|P_1, P_2, ..., P_N]]. \tag{2.15}$$

This depends on the joint probability distribution of $P_1, P_2, ..., P_N$ [14], which could be challenging to handle. To address this, a simulation approach based on artificial neural networks is proposed in Chapter 3. In practice, the mean is estimated as in Equation (2.12), while the method will give larger estimates for the variance in comparison with Equation (2.13). The skewness of X with uncertain probabilities will not be estimated in this thesis.

2.2 Artificial neural networks

The cerebral cortex is one of the largest and most developed parts of the mammalian brain [15], containing approximately ten billion nerve cells, or neurons. Each neuron can have thousands of connections with other neurons, resulting in the formation of neural networks that are responsible for processing various types of information, including sensory, visual, and auditory signals.

An artificial neural network is a machine learning algorithm inspired by the networks of neurons in the human brain and mimics the ability of processing information [15]. Despite having complex architectures with thousands of connections and parameters, artificial neural networks are still simplified in comparison with the dynamics of neural networks in the brain. However, both biological and artificial neural networks share the fundamental principle of learning through changes in the connections between neurons. The purpose of machine learning with artificial neural networks is to learn structures and patterns in data, with the goal of generalising the knowledge to unseen data. According to Cybenko [16] and Becker, Zhang, and Lee [17], an artificial neural network with just one hidden layer is capable of approximating any smooth function to arbitrary accuracy.

Machine learning with artificial neural networks has a long history, dating back to the middle of the 20th century. In 1943, McCulloch and Pitts [18] published their research about the McCulloch-Pitts neuron [15], which is presented in more detail in Section 2.2.1. In the late 1980s was the training algorithm for the Boltzmann machine with many hidden layers introduced, paving the way for research in deep learning. In the last couple of decades, artificial neural networks and deep learning have demonstrated impressive achievements in a wide range of tasks, including object recognition for self-driving cars [15], defeating the world champion in the board game Go [19], and OpenAI's recent success with ChatGPT [20].

Henceforth, the terms neural networks and artificial neural networks are used interchangeably, and unless otherwise specified, do not refer to biological neural networks.

2.2.1 The McCulloch-Pitts neuron and perceptrons

The neurons in an artificial neural network (ANN) are based on the work of McCulloch and Pitts [18]. The state of a neuron can be active or inactive, represented by 1/-1 or 1/0. The McCulloch-Pitts neuron is shown in Figure 2.1, with an arbitrary nonlinear activation function $g()$ replacing $sgn()$ to describe the neuron's state. The notations used in Section 2.2 of this thesis are adopted from Mehlig's book "Machine Learning with Neural Networks" [15].

The state of the output neuron O_i can be expressed by a weighted sum of the outputs from the previous layer and the activation function $g()$, further described in Section 2.2.4, according to:

$$O_i^{(L,\mu)} = g(b_i^{(L,\mu)}) = g(\sum_j w_{ij}^{(L)} V_j^{(L-1,\mu)} - \theta_i^{(L)}), \tag{2.16}$$

where b is called the local field of a neuron. The connection and the weight between neuron j in the previous layer and neuron i in the current layer, the output layer in this context, is denoted with w_{ij}. The state of neurons is denoted by V, except for the output layer where it is denoted with O and x for the input layer. The first superscript describes the layer number, which can be used to clarify when several hidden layers are used in the architecture of the network. Hidden layers refer to the layers between the output layer and the input layer. The second superscript μ refers to the sample in the input layer. The parameter θ_i is called the threshold of neuron i since the neuron change state if the incoming weighted sum from the neurons in the previous layers exceeds θ_i. In other machine learning literature, the parameter can be referred to as the bias, but it is then used with a plus sign [15].

The concept to connect several McCulloch-Pitts neurons in a layered structure where information flows forward in the network was proposed by Rosenblatt [21] in 1958. The author referred to the networks as perceptrons, and Figure 2.2 is an illustration of a fully connected perceptron with a single hidden layer. A perceptron with several hidden layers is often referred to as a multilayer perceptron (MLP) [15].

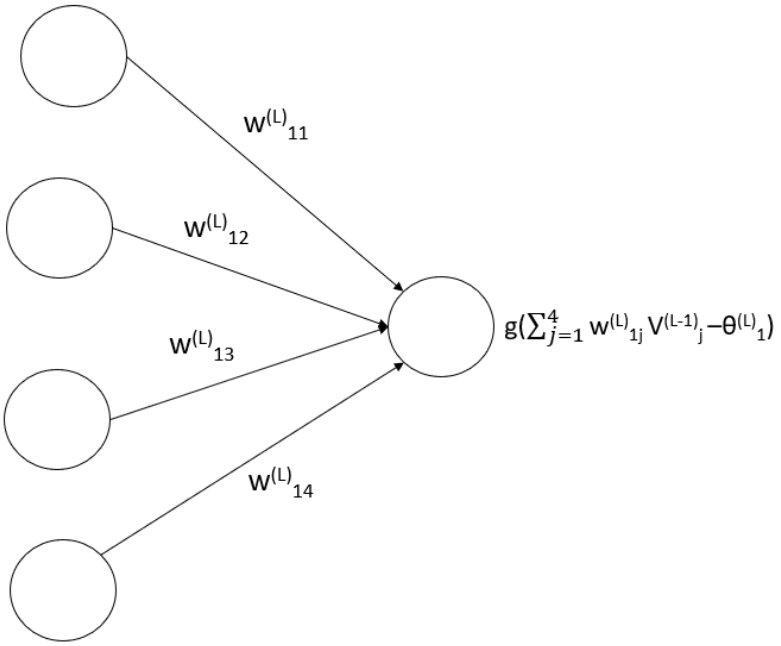

Figure 2.1: Illustration of the McCulloch-Pitts neuron with an arbitrary activation function $g()$ instead of $sgn()$. Notations are adopted from Mehlig's book "Machine Learning with Neural Networks" [15].

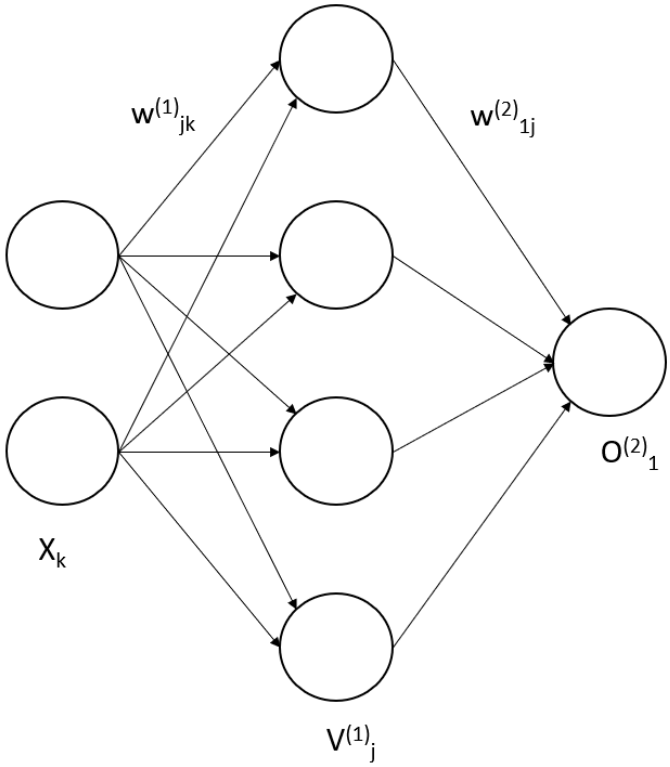

Figure 2.2: Illustration of a multilayer perceptron (MLP) with two input neurons, one hidden layer of four neurons and one output neuron.

To enable an ANN to learn, a loss function must be defined. Two commonly used

loss functions, which are used in this thesis, are the sum of squared errors and the binary cross entropy, defined by Equations (2.17) and (2.18) [15]:

$$H = \sum_{i,\mu} (t_i^{(\mu)} - O_i^{(\mu)})^2, \tag{2.17}$$

$$H = -\sum_{i,\mu} (t_i^{(\mu)} log(O_i^{(\mu)}) + (1 - t_i^{(\mu)}) log(1 - O_i^{(\mu)})). \tag{2.18}$$

The i-th target value of sample μ is denoted by $t_i^{(\mu)}$ and the corresponding predicted output is denoted by $O_i^{(\mu)}$. In this thesis, the binary cross entropy (2.18) is defined to work with target values represented by 1 or 0.

Mehlig [15] notes that training an ANN with Equation (2.17) is similar to regression analysis in statistical mathematics when minimising the sum of squared errors. Using Equation (2.18) instead, the training corresponds to minimise the negative log-likelihood for binary targets:

$$-log(L) = -log(\prod_{\mu} (O^{(\mu)})^{t^{(\mu)}} (1 - O^{(\mu)})^{1-t^{(\mu)}}). \tag{2.19}$$

Guo, Pleiss, Sun, *et al.* [22] suggest that this is why ANNs are often well-calibrated, meaning that $P(O = t|\hat{p} = p) \approx p$ for all samples in the input data, at least for networks with simple architectures. They argue that Equation (2.18) can indirectly be used as a measure of calibration. However, the authors also note that deep ANNs with many hidden layers and complex architectures may overfit and become miscalibrated based on their empirical findings.

ANNs are trained iteratively with gradient descent until convergence. The weights and thresholds of a neuron are updated using Equations (2.20) and (2.21):

$$w_{mn}^{(l)\prime} = w_{mn}^{(l)} - \delta w_{mn}^{(l)} = w_{mn}^{(l)} - \eta \frac{\partial H}{\partial w_{mn}^{(l)}}, \tag{2.20}$$

$$\theta_m^{(l)\prime} = \theta_m^{(l)} - \delta \theta_m^{(l)} = \theta_m^{(l)} - \eta \frac{\partial H}{\partial \theta_m^{(l)}}. \tag{2.21}$$

Where η is the learning rate and ∂ is used for the partial derivative. Ideally, the training process converges to the global minimum or a nearby local minimum of H.

2.2.2 Gradient descent for a simple artificial neural network

Consider the simple ANN in Figure 2.2 and using loss function (2.17). Since the output layer only has one neuron, the subscript i is not needed for this layer. To refer to any arbitrary weight in the network, the subscripts m and n are used. The partial derivative of the loss function with respect to a weight connecting to the output neuron $O^{(2,\mu)}$ can be expressed as:

$$\frac{\partial H}{\partial w_n^{(2)}} = 2 \sum_{\mu} (t^{(2,\mu)} - O^{(2,\mu)}) \frac{\partial O^{(2,\mu)}}{\partial w_n^{(2)}}, \tag{2.22}$$

$$\begin{aligned}\frac{\partial O^{(2,\mu)}}{\partial w_n^{(2)}} &= \sum_j \frac{dg(b^{(2,\mu)})}{db^{(2,\mu)}} \frac{\partial b^{(2,\mu)}}{\partial w_n^{(2)}} = \sum_j \frac{dg(b^{(2,\mu)})}{db^{(2,\mu)}} V_j^{(1,\mu)} \frac{\partial w_j^{(2)}}{\partial w_n^{(2)}} = \\ &= \sum_j \frac{dg(b^{(2,\mu)})}{db^{(2,\mu)}} V_j^{(1,\mu)} \delta_{jn} = \frac{dg(b^{(2,\mu)})}{db^{(2,\mu)}} V_n^{(1,\mu)}.\end{aligned} \tag{2.23}$$

It should be noted that the Kronecker delta, represented as δ_{jn}, is equal to 1 if j equals n, and 0 otherwise. As a result, only the n-term contributes to the summation over j. The variable d is utilised in the context of the derivative. Consequently, a weight in the second hidden layer is updated as follows:

$$\delta w_n^{(2)} = \eta \sum_\mu (t^{(2,\mu)} - O^{(2,\mu)}) \frac{dg(b^{(2,\mu)})}{db^{(2,\mu)}} V_n^{(1,\mu)} = \eta \sum_\mu \Delta^{(2,\mu)} V_n^{(1,\mu)}. \tag{2.24}$$

Note that the constant 2 is included in the learning rate η. To derive the update rule for the weights in the first layer, the following four equations are used:

$$\frac{\partial H}{\partial w_{mn}^{(1)}} = 2 \sum_\mu (t^{(2,\mu)} - O^{(2,\mu)}) \frac{\partial O^{(2,\mu)}}{\partial w_{mn}^{(1)}}, \tag{2.25}$$

$$\frac{\partial O^{(2,\mu)}}{\partial w_{mn}^{(1)}} = \sum_h \frac{\partial O^{(2,\mu)}}{\partial V_h^{(1,\mu)}} \frac{\partial V_h^{(1,\mu)}}{\partial w_{mn}^{(1)}}, \tag{2.26}$$

$$\frac{\partial O^{(2,\mu)}}{\partial V_h^{(1,\mu)}} = \frac{dg(b^{(2,\mu)})}{db^{(2,\mu)}} w_h^{(2)}, \tag{2.27}$$

$$\frac{\partial V_h^{(1,\mu)}}{\partial w_{mn}^{(1)}} = \sum_k \frac{dg(b_h^{(1,\mu)})}{db_h^{(1,\mu)}} \frac{\partial w_{hk}^{(1)}}{\partial w_{mn}^{(1)}} x_k^{(\mu)} = \sum_k \frac{dg(b_h^{(1,\mu)})}{db_h^{(1,\mu)}} \delta_{hm} \delta_{kn} x_k^{(\mu)} = \frac{dg(b_h^{(1,\mu)})}{db_h^{(1,\mu)}} \delta_{hm} x_n^{(\mu)}. \tag{2.28}$$

This results in:

$$\begin{aligned}\delta w_{mn}^{(1)} &= \eta \sum_\mu (t^{(2,\mu)} - O^{(2,\mu)}) \sum_h \frac{dg(b^{(2,\mu)})}{db^{(2,\mu)}} w_h^{(2)} \frac{dg(b_h^{(1,\mu)})}{db_h^{(1,\mu)}} \delta_{hm} x_n^{(\mu)} = \\ &= \eta \sum_\mu \Delta^{(2,\mu)} w_m^{(2)} \frac{dg(b_m^{(1,\mu)})}{db_m^{(1,\mu)}} x_n^{(\mu)} = \eta \sum_\mu \delta_m^{(1,\mu)} x_n^{(\mu)}.\end{aligned} \tag{2.29}$$

The variable $\delta_m^{(1,\mu)}$ can be interpreted as a weighted error in terms of $\Delta_i^{(2,\mu)}$:

$$\delta_m^{(1,\mu)} = \Delta^{(2,\mu)} w_m^{(2)} \frac{dg(b_m^{(1,\mu)})}{db_m^{(1,\mu)}}, \tag{2.30}$$

$$\Delta^{(2,\mu)} = (t^{(2,\mu)} - O^{(2,\mu)}) \frac{dg(b^{(2,\mu)})}{db^{(2,\mu)}}, \tag{2.31}$$

and should not be mixed up with the Kronecker delta.

The error in the output layer flows backwards through the layers, from right to left in Figure 2.2, and this is referred to as backpropagation. This is in contrast to the feed-forward updates of the neurons, which are carried out from left to right in the network [21].

The update rule for the thresholds in the second layer is derived in a similar way:

$$\begin{aligned}\frac{\partial H}{\partial \theta^{(2)}} &= 2\sum_{\mu}(t^{(2,\mu)} - O^{(2,\mu)})\frac{\partial O^{(2,\mu)}}{\partial \theta^{(2)}} = 2\sum_{\mu}(t^{(2,\mu)} - O^{(2,\mu)})\frac{dg(b^{(2,\mu)})}{db^{(2,\mu)}}\frac{\partial b^{(2,\mu)}}{\partial \theta^{(2)}} = \\ &= -2\sum_{\mu}(t^{(2,\mu)} - O^{(2,\mu)})\frac{dg(b^{(2,\mu)})}{db^{(2,\mu)}}.\end{aligned} \tag{2.32}$$

This results in:

$$\delta\theta^{(2)} = \eta\sum_{\mu}(t^{(2,\mu)} - O^{(2,\mu)})\frac{-dg(b^{(2,\mu)})}{db^{(2,\mu)}} = -\eta\sum_{\mu}\Delta^{(2,\mu)}. \tag{2.33}$$

Lastly, the update rule for the thresholds in the first layer needs to be derived. The following equations are used:

$$\frac{\partial H}{\partial \theta_m^{(1)}} = 2\sum_{\mu}(t^{(2,\mu)} - O^{(2,\mu)})\frac{\partial O^{(2,\mu)}}{\partial \theta_m^{(1)}}, \tag{2.34}$$

$$\frac{\partial O^{(2,\mu)}}{\partial \theta_m^{(1)}} = \sum_h \frac{\partial O^{(2,\mu)}}{\partial V_h^{(1,\mu)}}\frac{\partial V_h^{(1,\mu)}}{\partial \theta_m^{(1)}}, \tag{2.35}$$

$$\frac{\partial V_h^{(1,\mu)}}{\partial \theta_m^{(1)}} = \frac{dg(b_h^{(1,\mu)})}{db_h^{(1,\mu)}}\frac{\partial b_h^{(1,\mu)}}{\partial \theta_m^{(1)}} = -\frac{dg(b_h^{(1,\mu)})}{db_h^{(1,\mu)}}\frac{\partial \theta_h^{(1)}}{\partial \theta_m^{(1)}} = -\frac{dg(b_h^{(1,\mu)})}{db_h^{(1,\mu)}}\delta_{hm}. \tag{2.36}$$

When used together with Equation (2.27), this results in the update rule for the thresholds in the first layer:

$$\delta\theta_m^{(1)} = -\eta\sum_{\mu}(t^{(2,\mu)} - O^{(2,\mu)})\sum_h \frac{dg(b^{(2,\mu)})}{db^{(2,\mu)}}w_h^{(2)}\frac{dg(b_h^{(1,\mu)})}{db_h^{(1,\mu)}}\delta_{hm} = -\eta\sum_{\mu}\delta_m^{(1,\mu)}. \tag{2.37}$$

While the above example only involved a single layer, the reasoning can be generalised to more complex architectures. The general update rules are given by [15]:

$$\delta w_{mn}^{(l)} = \eta\sum_{\mu}\delta_m^{(l,\mu)}V_n^{(l-1,\mu)}, \tag{2.38}$$

$$\delta\theta_m^{(l)} = -\eta\sum_{\mu}\delta_m^{(l,\mu)}. \tag{2.39}$$

Where the weighted errors are defined as:

$$\delta_j^{(l,\mu)} = \sum_i (t_i^{(L,\mu)} - V_i^{(L,\mu)})\frac{\partial V_i^{(L,\mu)}}{\partial V_j^{(l,\mu)}}\frac{dg(b_j^{(l,\mu)})}{db_j^{(l,\mu)}}. \tag{2.40}$$

Note that $V_i^{(L,\mu)}$ is equal to $O_i^{(L,\mu)}$. Furthermore, Equation (2.40) actually corresponds to [15]:

$$\delta_j^{(l)} = \sum_i \delta_i^{(l+1)} w_{ij}^{(l+1)} \frac{dg(b_j^{(l,\mu)})}{db_j^{(l,\mu)}}. \tag{2.41}$$

The purpose of Equation (2.41) is to show the recursion of the weighted errors, which is an important aspect of training deep neural networks. With many hidden layers, the problem of vanishing or exploding gradients can occur [15]. This is apparent in Equation (2.41), where the weighted errors of weights and thresholds of layers close to the input layer will have many factors of $dg(b_j^{(l,\mu)})/db_j^{(l,\mu)}$ in the product. As a result, the updates either tend to be very small or very large, leading to unwanted behaviour in the training of the network

2.2.3 Stochastic gradient descent

The weight and threshold update rules, (2.38) and (2.39), include a sum over all samples μ in the input data set, and this corresponds to batch training [15]. Sequential training, on the other hand, refers to using only a single pattern in each training iteration. When all patterns have been fed to the network, this corresponds to an epoch. The term for random sequential training is stochastic gradient descent, and this method is less prone to getting stuck in local minima of the loss function compared to batch training [15]. The loss function can be non-convex in the parameter space, especially in networks with many hidden layers, and thus stochastic gradient descent is preferred. However, stochastic gradient descent increases the training time compared to batch training since it cannot efficiently utilise parallel computation in the backpropagation and can be too noisy. Therefore, mini-batches are often used in practice to average the updates over randomly shuffled samples. Equation (2.38) and (2.39) are replaced with [15]:

$$\delta w_{mn}^{(l)} = \eta \sum_{\mu=1}^{m_B} \delta_m^{(l,\mu)} V_n^{(l-1,\mu)}, \tag{2.42}$$

$$\delta \theta_m^{(l)} = -\eta \sum_{\mu=1}^{m_B} \delta_m^{(l,\mu)}, \tag{2.43}$$

where m_B is the size of the mini-batches.

In addition to the standard stochastic gradient descent with mini-batches, there exist several more sophisticated optimisation methods. However, choosing an appropriate learning rate can be a difficult task. For some problems, a large learning rate may result in rapid convergence to an optimal solution, but if the loss function varies greatly, the training may fail and produce ineffective models [15]. Conversely, a small learning rate can lead to slow convergence and hinder the escape of local minima.

In 2015, Kingma and Ba [23] introduced the Adam optimiser, which applies individual adaptive learning rates for the parameters based on estimates of the first

and second moments of the gradients. According to the authors, this optimiser is effective for problems involving a significant amount of noise and sparse gradients, such as those in which only a small subset of features is informative. The authors were inspired by the work of Duchi, Hazan, and Singer [24], who proposed the AdaGrad optimiser and Tieleman and Hinton [25] who proposed the RMSProp optimiser. However, some research suggests that the standard stochastic gradient descent may generalise better than adaptive optimisers such as Adam [26]–[28]. The Adam optimiser can be described by Equations (2.44)-(2.48), following similar notations as used by Kingma and Ba [23]:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\frac{\partial H}{\partial w_{\theta t}}, \tag{2.44}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)\frac{\partial^2 H}{\partial w_{\theta t}^2}, \tag{2.45}$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \tag{2.46}$$

$$\hat{v}_t = \frac{v_t}{(1 - \beta_2^t)}, \tag{2.47}$$

$$\theta_t = \theta_{t-1} - \eta\frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}. \tag{2.48}$$

Equations (2.44) and (2.45) provide estimates of the first and second moment, while Equations (2.46) and (2.47) provide estimates of the bias-corrected first and second moment of the gradients. The authors describe that the bias in Equations (2.44) and (2.45) arises from the zero initialisation, resulting in estimates biased towards zero.

It should be noted that there is a slight difference in notation compared to earlier, θ refers to all parameters of the neural network, including weights and thresholds. The variable t refers to the iteration in the training loop. According to Goodfellow, Bengio, and Courville [29], the Adam optimiser is quite robust in terms of values for β_1, β_2 and ϵ, and the default values of $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$ are usually adequate for practical applications. However, the default learning rate $\eta = 0.001$ may require tuning.

2.2.4 Activation functions

In this thesis, the activation function is denoted by $g()$. While the $sgn()$ function could be used as the activation function if the states are represented by 1/-1, it is not used in practice due to the significant impact on the output from even a small change in the local fields. Therefore, $g()$ is typically a continuous, nonlinear, and differentiable function, or at least piecewise differentiable [15]. Commonly used activation functions include those in Equations (2.49)-(2.51). The logistic sigmoid function is often used for the last layer to model binary outcomes.

The logistic sigmoid function:

$$g(b) = \frac{1}{1 + e^{-b}}. \tag{2.49}$$

The hyperbolic tangent (tanh):

$$g(b) = \frac{e^{b} - e^{-b}}{e^{b} + e^{-b}}. \tag{2.50}$$

Rectified linear units (ReLU):

$$g(b) = max\{0, b\}. \tag{2.51}$$

Glorot, Bordes, and Bengio [30] argue that the logistic sigmoid function is biologically plausible in terms of the firing rate as a function of the input current of a biological neuron in the mammalian brain. However, from an optimisation standpoint, *tanh* is preferred since it has a steady state of zero. The authors suggest that the *ReLU* function is more plausible for modelling biological neurons and achieves great performance in deep ANNs compared to *tanh*, resulting in sparse representation where neurons are equal to zero. The ReLU activation function is also suggested to mitigate the vanishing-gradient problem [15]. However, the exploding-gradient problem remains an issue, since the weights tend to increase in accordance with $g'(b) = max\{0, 1\}$. Therefore, regularisation may be necessary.

2.2.5 Regularisation

Glorot, Bordes, and Bengio [30] recommend using weight decay or L_2-regularisation in conjunction with *ReLU* as the activation function. This corresponds to adding a term to the weight updates in Equation (2.38). In mathematical terms:

$$\delta w_{mn}^{(l)} += -\epsilon w_{mn}^{(l)}, 0 < \epsilon < 1. \tag{2.52}$$

With the loss function (2.17), this leads to:

$$H = \sum_{i,\mu} (t_i^{(\mu)} - O_i^{(\mu)})^2 + \gamma \sum_{ij} w_{ij}^2. \tag{2.53}$$

With the same steps as in Section 2.2.2, it can be shown that Equation (2.53) is equivalent to Equation (2.52), where $\epsilon = \eta\gamma$. The purpose of weight decay is to prevent excessively large weights in the network. Mehlig [15] argues that networks with smaller weights are more robust to noise in the input and may result in better generalisation. However, L_2-regularization and weight decay are not equivalent for adaptive optimisers, such as Adam [31]. To address this issue, Loshchilov and Hutter [31] proposed AdamW, Adam with decoupled weight decay. The result is a modification of Equation (2.48):

$$\theta_t = \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} - \gamma\theta_{t-1}. \tag{2.54}$$

Early stopping is another method used to prevent the model from overfitting [15]. The idea is to learn general features by splitting the training data into two sets: a training set and a validation set. Training continues as long as both the training loss and the validation loss decrease, and stops when the validation loss starts to increase. Early stopping is easy to implement and can decrease the training time. However, the loss function can vary significantly, meaning that it can increase for some iterations and then start to decrease again. Therefore, it is common to implement a patience constant k, which is used to stop the training when the loss function has not decreased for k iterations or epochs. Additionally, given a neural network architecture, training with more samples can also help reduce overfitting [15].

2.2.6 Weight initialisation and preprocessing of the input data

The fundamental idea of initialisation methods is to ensure that the weight updates remain stable during the training process. However, according to Mehlig [15], the training dynamics are inherently unstable due to the recursion in Equation 2.41. The interested reader is referred to Section 7.2 in [15] whereupon the author uses mean-field theory to derive an initialisation rule. He, Zhang, Ren, *et al.* [32] also discusses this problem in the context *ReLU*. The authors propose the following initialisation method for neural networks with *ReLU* activation function:

$$w_{ij}^{(l)} \sim N(0, 2/n_l), \tag{2.55}$$

where n_l refers to the number of neurons in the previous layer. Glorot and Bengio [33] propose the following initialisation method for neural networks with logistic sigmoid (2.49) or *tanh* (2.50) activation function:

$$w_{ij}^{(l)} \sim U[-1/n_l, 1/n_l]. \tag{2.56}$$

Although the two initialisation methods described above originate from the original papers, it is important to note that alternative versions of the equations and different initialisation methods are also widely used in practice.

The thresholds are commonly zero initialised since it is argued that they learn faster than the weights [15]. Additionally, it is often recommended to centre and scale the features of the input data set, such that the mean and variance are zero and one, respectively. Furthermore, the features of the validation and test set should be centred and scaled using the same transformation applied to the training data.

2.2.7 Entity embedding of categorical features

Categorical data can be handled in many different ways, and one of the most common methods is one-hot encoding [34]. Since the input to a neural network has to be a real value, categorical data needs to be converted into numerical data. One-hot encoding represents each group i of the categorical variable x with a vector $\boldsymbol{u}_i$ with a one at position i and zeros elsewhere. As a result, the length of the input vector

$|\boldsymbol{u}|$ of the categorical variable x is equivalent to the number of groups of x, which is also referred to as the cardinality of x. However, this can lead to problems in the context of Big Data and large sample sizes since the high cardinality of x results in large and sparse input $\boldsymbol{u}_i$. This could have a negative impact on the storage capacity [34].

Another strategy is to assign an integer to each group i, known as label encoding. This approach is intuitive and easy to implement, but it includes artificial hierarchical importance and order among the groups [34]. Therefore, it is not recommended for neural networks.

Embedding is a commonly used technique in natural language processing to represent words and phrases as real-valued vectors. The concept can also be applied to categorical data, and Guo, Pleiss, Sun, *et al.* [22] introduced the corresponding term entity embedding in 2016. This method has demonstrated promising results for various problems [35], [36]. The method can map similar categories to each other and this can lead to valuable insights through visualisation of the embedding [36]. In addition, the authors claim that entity embedding is more computationally efficient than one-hot encoding. Equation (2.57) clarifies the concept for an embedding of dimension d for the first group of some arbitrary categorical variable with cardinality c:

$$\begin{bmatrix} w_{11} & w_{12} & \dots & w_{1d} \\ w_{21} & w_{22} & & \vdots \\ \vdots & & \ddots & \\ w_{c1} & w_{c2} & \dots & w_{cd} \end{bmatrix}^T \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} w_{11} \\ w_{12} \\ \vdots \\ w_{1d} \end{bmatrix} = \boldsymbol{w}_1. \tag{2.57}$$

The embeddings are trained using stochastic gradient descent along with the other parameters of the neural network. It is important to note that the embedding layer functions similarly to the other layers in the network but without thresholds and an activation function. The input is the one-hot vector $\boldsymbol{u}_i$. As far as the author of this thesis is aware, there is no standard rule for determining the dimension d. In this thesis, the following rule [37] is used, where m is a hyperparameter and $\lfloor\rfloor$ is the floor function:

$$d_x = min\{m, \lfloor c_x/2 \rfloor\},\ m \in \{2, 3, ..., 50\}. \tag{2.58}$$

2.2.8 Ensembles of neural networks

Ensembles are useful in the context of ANNs since they can easily overfit to noise in the data set. While several techniques to prevent this were presented in Section 2.2.5, ensembles can be even more effective [38]. The reason why ensembles are not more commonly used in the context of ANNs is their extensive training times. To address this, Srivastava, Hinton, Krizhevsky, *et al.* [38] proposed the regularisation technique dropout, where a percentage p of the neurons are ignored during a training step and update. Thus, only the parameters connecting to and from the remaining

$1 - p$ percentage of the neurons are updated. The neurons are chosen at random and probably updated several times during training. After training, all neurons are activated and their outputs are multiplied by 1-p since the outputs should be expected to be the same during training and testing [38].

However, Lakshminarayanan, Pritzel, and Blundell [39] empirically found that an ANN trained with dropout can be poorly calibrated in comparison with the average prediction of several ANNs. The authors apply a simple average of ANNs instead of an average based on bootstrapping, which is commonly used in machine learning. Bootstrapping produces n training sets of size m with samples drawn uniformly with replacement from the training set. Lakshminarayanan, Pritzel, and Blundell [39] argues that ANNs tend to generalise better when trained on the entire data set and they are likely to converge to different local minima anyway. A conclusion of their work is that an average of five instances can make a significant difference in the context of uncertainty prediction and calibration. The authors further write that the ensemble technique of stacked generalisation could be useful. This means that a meta-learner is trained on several models' outputs and can potentially produce better results than a simple average of the models. This requires a second validation set in order to utilise early stopping described in Section 2.2.5.

2.3 Recurrent neural networks

Recurrent neural networks (RNNs) are more general than MLPs since they incorporate feedback connections [15]. The feedback connections can be implemented in various ways in the network, such as in the hidden layer shown in Figure 2.3. In this way, the network is able to remember the previous states of the same hidden layer. Figure 2.3 can be a bit misleading as the connections from $V_i(t)$ to $V_j(t-1)$ are not entirely accurate. The idea is to save previous states of the neurons, thus in this context, the connections are constant 1 and connect $V_i(t-1)$ and $V_j(t-1)$. The weights $w_{ij}^{(hh)}$ connect $V_i(t)$ and $V_j(t-1)$, and it is worth noting that they are not indexed with t. These types of artificial neural networks are often referred to as Elman RNNs, which were presented by Elman [40] in 1991. The RNN in Figure 2.3 is unfolded in time [15] and can be trained by stochastic gradient descent, which is commonly referred to as backpropagation through time (BPTT). One of the purposes of RNNs is to model sequential data of any arbitrary length T, which could be useful in the context of natural language processing or time series modelling. In this thesis, an RNN with only one output neuron will be used. The network in Figure 2.4 with an output neuron at T could be used for this purpose. The notations in this section are also similar to the notations in [15].

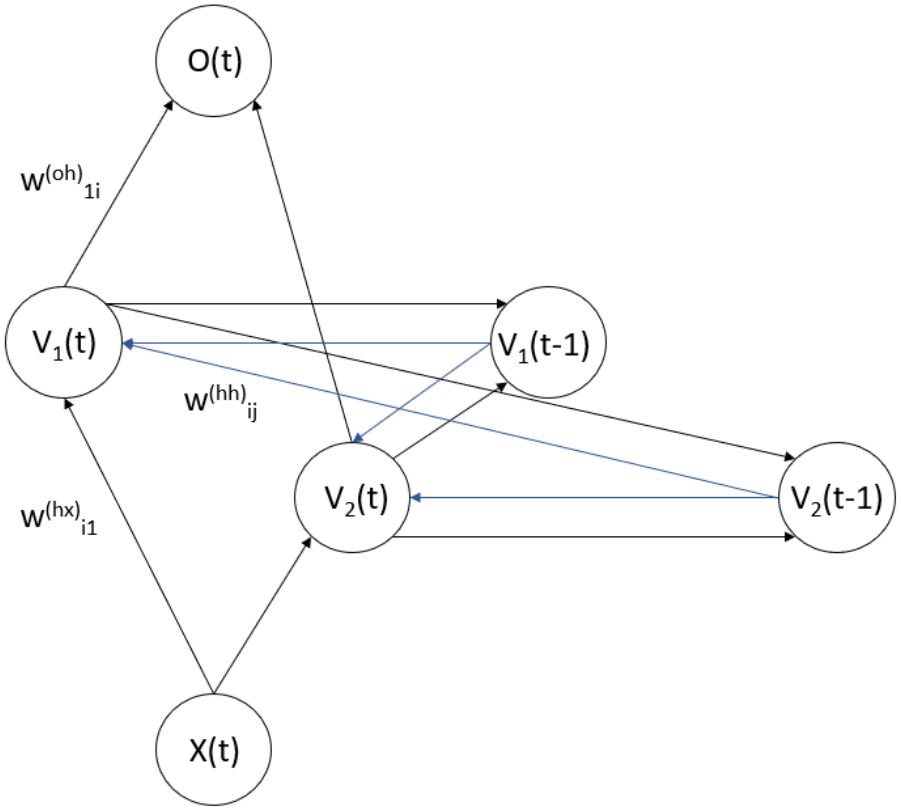

Figure 2.3: Illustration of a recurrent neural network (RNN), often referred to as an Elman RNN, with one input neuron, two hidden neurons a one output neuron.

The neurons are updated via the following feed-forward passes:

$$V_i(t) = g(b_i(t)) = g(\sum_j w_{ij}^{(hh)} V_j(t-1) + \sum_k w_{ik}^{(hx)} x_k(t) - \theta_i^{(h)}), \tag{2.59}$$

$$O_l = g(B_l(t)) = g(\sum_i w_{li}^{(oh)} V_i(t) - \theta_l^{(o)}). \tag{2.60}$$

The *tanh* function is commonly used as the activation function $g()$ for the hidden layers of RNNs [15]. The sum of the squared errors loss function for an RNN with output neurons at every time step can be defined as [15]:

$$H = \sum_{i,\mu,t} (y_i^{(\mu)}(t) - O_i^{(\mu)}(t))^2 = \sum_{i,\mu,t} (r_i^{(\mu)}(t))^2. \tag{2.61}$$

An RNN can be trained with BPTT when unfolded in time, using Equations (2.20) and (2.21).

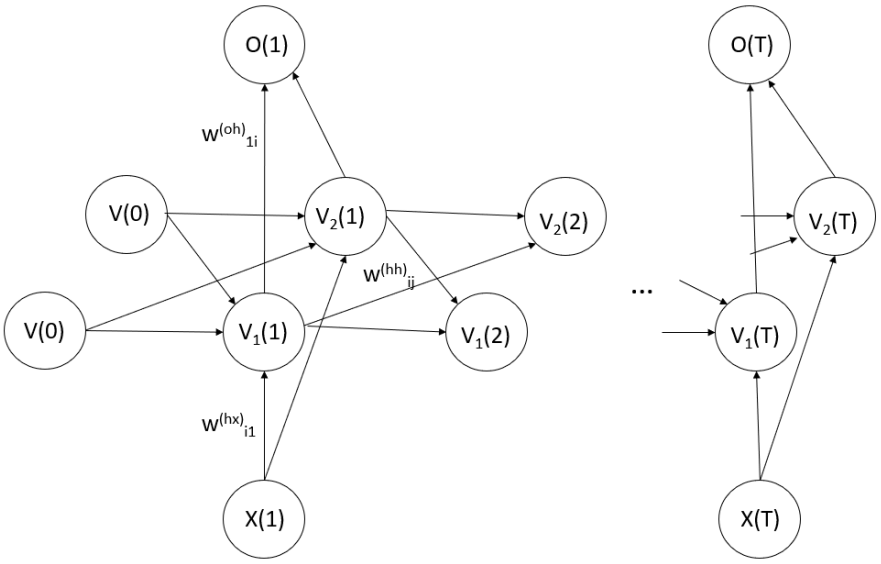

Figure 2.4: Unfolded representation of the RNN in Figure 2.3, visualising the hidden neurons' recurrent connections across multiple time steps. The arrows point in the direction of the information flow. The weights are not indexed with time.

2.3.1 Gradient descent for a simple recurrent neural network

Consider the following scenario: a simple RNN with a single input neuron, hidden neuron and output neuron. The RNN is trained on sequences with length $T = 3$ using the loss function (2.61) and the stochastic gradient descent ($\mu = 1$) with constants included in the learning rate η. The updates for the weights connecting the hidden neurons in time can be derived as follows:

$$\delta w^{(hh)} = \eta \sum_{t=1}^{3} r(t) \frac{\partial O(t)}{\partial w^{(hh)}}, \tag{2.62}$$

$$\frac{\partial O(t)}{\partial w^{(hh)}} = \frac{\partial O(t)}{\partial V(t)} \frac{\partial V(t)}{\partial w^{(hh)}} = \frac{dg(B(t))}{dB(t)} w^{(oh)} \frac{\partial V(t)}{\partial w^{(hh)}}, \tag{2.63}$$

where B is used for the local field of the output neuron and b is used for the local field of the hidden neuron. Based on the product rule and Equation (2.59), the following holds for $t \geq 1$:

$$\frac{\partial V(t)}{\partial w^{(hh)}} = \frac{dg(b(t))}{db(t)} (V(t-1) + w^{(hh)} \frac{\partial V(t-1)}{\partial w^{(hh)}}). \tag{2.64}$$

It is worth noting the recursion in Equation (2.64), which implies that RNNs face similar issues with the vanishing or exploding gradient problem as MLPs [15]. At t=0, $\partial V(t)/\partial w^{(hh)}$ is equal to zero. Therefore, according to Equation (2.64), the following holds:

$$
\begin{aligned}
\frac{\partial V(1)}{\partial w^{(hh)}} &= \frac{dg(b(1))}{db(1)} V(0) \\
\frac{\partial V(2)}{\partial w^{(hh)}} &= \frac{dg(b(2))}{db(2)} (V(1) + w^{(hh)} \frac{\partial V(1)}{\partial w^{(hh)}}) = \\
&= \frac{dg(b(2))}{db(2)} V(1) + \frac{dg(b(2))}{db(2)} w^{(hh)} \frac{dg(b(1))}{db(1)} V(0) \\
\frac{\partial V(3)}{\partial w^{(hh)}} &= \frac{dg(b(3))}{db(3)} (V(2) + w^{(hh)} \frac{\partial V(2)}{\partial w^{(hh)}}) = \\
&= \frac{dg(b(3))}{db(3)} V(2) + \frac{dg(b(3))}{db(3)} w^{(hh)} \frac{dg(b(2))}{db(2)} V(1) + \\
&+ \frac{dg(b(3))}{db(3)} w^{(hh)} \frac{dg(b(2))}{db(2)} w^{(hh)} \frac{dg(b(1))}{db(1)} V(0).
\end{aligned}
$$

Equation (2.62) sums over t, thus can the following be useful:

$$
\begin{aligned}
&\frac{\partial V(1)}{\partial w^{(hh)}} + \frac{\partial V(2)}{\partial w^{(hh)}} + \frac{\partial V(3)}{\partial w^{(hh)}} = \\
&= V(0)(\frac{dg(b(1))}{db(1)} + \frac{dg(b(2))}{db(2)} w^{(hh)} \frac{dg(b(1))}{db(1)} + \frac{dg(b(3))}{db(3)} w^{(hh)} \frac{dg(b(2))}{db(2)} w^{(hh)} \frac{dg(b(1))}{db(1)}) + \\
&+ V(1)(\frac{dg(b(2))}{db(2)} + \frac{dg(b(3))}{db(3)} w^{(hh)} \frac{dg(b(2))}{db(2)}) + \\
&+ V(2) \frac{dg(b(3))}{db(3)}.
\end{aligned}
\tag{2.65}
$$

Rewriting Equation (2.62) with Equation (2.65) and Equation (2.30):

$$
\begin{aligned}
r(1)\frac{\partial O(1)}{\partial w^{(hh)}} + r(2)\frac{\partial O(2)}{\partial w^{(hh)}} + r(3)\frac{\partial O(3)}{\partial w^{(hh)}} &= \\
&= V(0)(\Delta(1)\frac{dg(b(1))}{db(1)} + \Delta(2)\frac{dg(b(2))}{db(2)} w^{(hh)} \frac{dg(b(1))}{db(1)} + \\
&+ \Delta(3)\frac{dg(b(3))}{db(3)} w^{(hh)} \frac{dg(b(2))}{db(2)} w^{(hh)} \frac{dg(b(1))}{db(1)}) + \\
&+ V(1)(\Delta(2)\frac{dg(b(2))}{db(2)} + \Delta(3)\frac{dg(b(3))}{db(3)} w^{(hh)} \frac{dg(b(2))}{db(2)}) + \\
&+ V(2)\Delta(3)\frac{dg(b(3))}{db(3)}.
\end{aligned}
\tag{2.66}
$$

If the weighted errors $\delta(t)$ are defined recursively as [15]:

$$
\delta(t) = \begin{cases} \Delta(t) w^{(oh)} \frac{dg(b(t))}{db(t)}, & t = 3 \\ \Delta(t) w^{(oh)} \frac{dg(b(t))}{db(t)} + \delta(t+1) w^{(hh)} \frac{dg(b(t))}{db(t)}, & t = \{1, 2\}. \end{cases} \tag{2.67}
$$

The connection between the hidden neurons in time is updated with:

$$
\delta w^{(hh)} = \eta \sum_{t=1}^{3} \delta(t) V(t-1). \tag{2.68}
$$

Note the similarity between the update rule (2.38) for MLPs and the update rule (2.68) for RNNs. Using the Equations (2.60) and (2.64), the update rule for $w^{(hx)}$ can be written as:

$$\delta w^{(hx)} = \eta \sum_{t=1}^{3} \delta(t) x(t). \tag{2.69}$$

Since $\partial O(t)/\partial w^{(oh)} = dg(B(t))/dB(t)V(t)$, the update rule for $w^{(oh)}$ is:

$$\delta w^{(oh)} = \eta \sum_{t=1}^{3} r(t) \frac{dg(B(t))}{dB(t)} V(t) = \eta \sum_{t=1}^{3} \Delta(t) V(t). \tag{2.70}$$

The corresponding equation of (2.64) for the the threshold $\theta^{(h)}$ is:

$$\frac{\partial V(t)}{\partial \theta^{(h)}} = \frac{dg(b(t))}{db(t)}(-1 + w^{(hh)} \frac{\partial V(t-1)}{\partial \theta^{(h)}}). \tag{2.71}$$

By comparing Equation (2.71) and (2.64), the corresponding equation of (2.68) for the threshold $\theta^{(h)}$ is given by:

$$\delta \theta^{(h)} = -\eta \sum_{t=1}^{3} \delta(t). \tag{2.72}$$

Since $\partial O(t)/\partial \theta^{(o)} = -dg(B(t))/dB(t)$, the update rule for $\theta^{(o)}$ is:

$$\delta \theta^{(o)} = -\eta \sum_{t=1}^{3} \Delta(t) \tag{2.73}$$

If the reader is interested in the derivation of the update rules for the general case, it can be found in [15]. The purpose of this section is to demonstrate that BPTT is similar to backpropagation for an MLP. However, as previously mentioned, the recursion in Equation (2.64) easily lead to problems with vanishing or exploding gradient during the training procedure. Therefore, adjustments to the Elman RNN have been proposed, for instance, gated recurrent units and long short-term memory units.

2.3.2 Long short-term memory

RNNs with gates are proposed to address the vanishing or exploding gradients by controlling the flow of information through different gates. The architecture of the Long short-term memory (LSTM) unit was introduced by Hochreiter and Schmidhuber [41] in 1997 and has been widely used for numerous applications in the context of sequence modelling [29]. The LSTM unit is visualised in Figure 2.5. The notations in this section are adopted from [29], [15].

The circles in Figure 2.5 represent elementwise operators, the rectangles represent activation functions and the arrows point in the direction of information flow. The logistic sigmoid function (2.49) is represented with σ.

In addition to a hidden neuron $V_i(t)$, an LSTM unit includes an internal state cell $s_i(t)$ that also has recurrent connections. The forget gate $f(t)$ allows the internal state cell to determine how much information from the previous cell state $s_i(t-1)$ to discard and the input gate $g(t)$ allows the cell to learn how much of the new information to include.

$$f_i(t) = \sigma(\sum_j w_{ij}^{(fx)} x_j(t) + \sum_j w_{ij}^{(fh)} V_j(t-1) - \theta_i^{(f)}). \tag{2.74}$$

$$s_i(t) = f_i(t) s_i(t-1) + g_i(t) \tilde{h}_i(t). \tag{2.75}$$

$$\tilde{h}_i(t) = tanh(\sum_j w_{ij}^{(\tilde{h}x)} x_j(t) + \sum_j w_{ij}^{(\tilde{h}h)} V_j(t-1) - \theta_i^{(\tilde{h})}). \tag{2.76}$$

$$g_i(t) = \sigma(\sum_j w_{ij}^{(gx)} x_j(t) + \sum_j w_{ij}^{(gh)} V_j(t-1) - \theta_i^{(g)}). \tag{2.77}$$

Note that h refers to the hidden layer and $\tilde{h}$ refers to the new proposed information for the cell state $s_i(t)$. However, it is filtered by the input gate before it reaches $s_i(t)$. The next hidden state $V_i(t)$ is calculated as follows:

$$V_i(t) = tanh(s_i(t)) q_i(t). \tag{2.78}$$

Where $q_i(t)$ is the output gate, which could be used to discard information from the internal state cell, $s_i(t)$.

$$q_i(t) = \sigma(\sum_j w_{ij}^{(qx)} x_j(t) + \sum_j w_{ij}^{(qh)} V_j(t-1) - \theta_i^{(q)}). \tag{2.79}$$

All parameters of Equations (2.74)-(2.77) are incorporated in the BPTT described in Section 2.3.1. However, using a complex LSTM architecture can be computationally expensive [42]. The following section introduces a simpler gated RNN with fewer parameters.

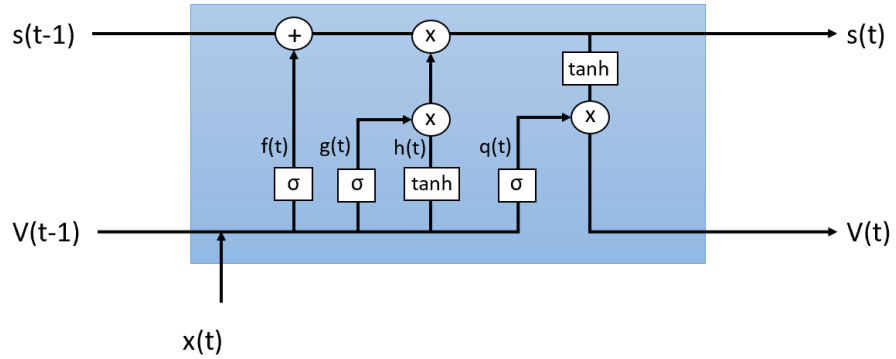

Figure 2.5: Visualisation of the information flow through a long short-term memory (LSTM) unit. The circles represent elementwise operators, and the rectangles represent activation functions. The logistic sigmoid (2.49) is denoted by σ and $tanh$ refers to the hyperbolic tangent function (2.50). The internal cell state s, defined in Equation (2.75), has recurrent connections similar to the hidden neuron V. The functions $f, \tilde{h}, g, q$ are defined in Equations (2.74), (2.76), (2.77) and (2.79), respectively.

2.3.3 Gated recurrent units

Gated recurrent units (GRUs) were proposed by Cho, Van Merriënboer, Gulcehre, *et al.* [42] in 2014 as an alternative to LSTM units. GRUs are more computationally efficient and have fewer parameters. Equations (2.80)-(2.83) describe the unit, where the first two equations are referred to as the gates [15]. The filtered information from the input $x(t)$ and the previous state $V(t-1)$ is described by $\tilde{h}_t$. The information flow through GRUs can be visualised by Figure 2.6.

$$r_i(t) = \sigma(\sum_k w_{ik}^{(zx)} x_k(t) + \sum_j w_{ij}^{(zh)} V_j(t-1)). \tag{2.80}$$

$$z_i(t) = \sigma(\sum_k w_{ik}^{(rx)} x_k(t) + \sum_j w_{nj}^{(rh)} V_j(t-1)). \tag{2.81}$$

$$\tilde{h}_i(t) = tanh(\sum_k w_{ik}^{(\tilde{h}x)} x_k(t) + \sum_j w_{ij}^{(\tilde{h}h)} r_j(t) V_j(t-1)). \tag{2.82}$$

$$V_i(t) = (1 - z_i(t))\tilde{h}_i(t) + z_i(t) V_i(t-1). \tag{2.83}$$

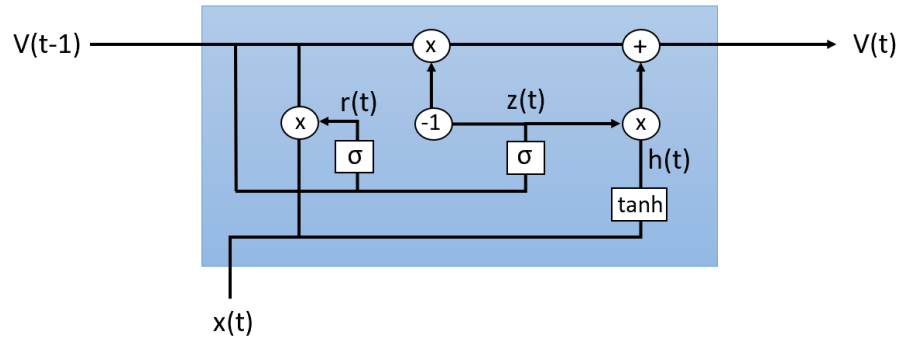

Figure 2.6: Visualisation of the information flow through a gated recurrent unit (GRU). The circles represent elementwise operators, and the rectangles represent activation functions. The logistic sigmoid (2.49) is denoted by σ and *tanh* refers to the hyperbolic tangent function (2.50). The functions $r, z, \tilde{h}$ are defined in Equations (2.80)-(2.82)

.

2.4 LightGBM

Gradient Boosting Trees is a popular machine learning algorithm and it has achieved state-of-the-art performance for many problems, including classification problems with tabular data [43]. LightGBM is one of many implementations of the algorithm and it is designed to be efficient and accurate for large sample sizes and high-cardinality features [43]. Gradient Boosting Trees often outperform predictive models based on ANNs on tabular data [44].

Gradient Boosting Trees is an ensemble method of classification and regression trees. Therefore, it can be used together with both loss functions presented in this thesis, Equations (2.17) and (2.18). The notations in this section are adopted from Guo and Berkhahn [36].

The feature space is divided into M regions, where R_m denotes a region and $|R_m|$ represents the number of samples in R_m. The output function is modelled as:

$$\hat{f}(\boldsymbol{x}^{(\mu)}) = \sum_{m=1}^{M} \hat{c}_m I(\boldsymbol{x}^{(\mu)} \in R_m). \tag{2.84}$$

Where μ is the input pattern and I is the indicator function. If the loss function is the sum of squared errors (2.17), $\hat{c}_m$ is equivalent to:

$$\hat{c}_m = \frac{1}{|R_m|} \sum_{\boldsymbol{x}^{(\mu)} \in R_m} y^{(\mu)}. \tag{2.85}$$

If the loss function is the binary cross entropy (2.18), c_m is equivalent to:

$$\hat{c}_m = \underset{c \in \{0,1\}}{max} \frac{1}{|R_m|} \sum_{\boldsymbol{x}^{(\mu)} \in R_m} I(c_\mu = c). \tag{2.86}$$

Classification and regression trees are used to approximate R_m through a sequence of binary axis-parallel splits [36]. Binary splitting means finding a threshold t_i if i is a continuous feature, or if it is a categorical feature, a subset of its groups. The threshold is selected to maximise the node purity, defined as minimising the sum of squared errors (2.17) or binary cross entropy (2.18). At each iteration, each feature is considered, and the best split is chosen, leading to two new child nodes. In mathematical terms, with the loss function (2.17):

$$\min_{i,t_i}[\sum_{\boldsymbol{x}^{(\mu)}\in R_1}(y^{(\mu)}-\hat{c}_1)^2+\sum_{\boldsymbol{x}^{(\mu)}\in R_2}(y^{(\mu)}-\hat{c}_2)^2]. \tag{2.87}$$

This procedure is then applied recursively until a stopping criterion is met.

Approximations from classification and regression trees have high variance and thus are ensemble methods often preferred [36]. Two popular ensemble methods are Random Forests, which are based on bootstrapping, and Gradient boosting trees. Informally, Gradient Boosting Trees combine weak learners into one single learner. Consider the tree T_k, the iteration t and sample μ:

$$\hat{f}^{(t)}(\boldsymbol{x}^{(\mu)})=\sum_{k=1}^{t}T_k(x^{(\mu)}). \tag{2.88}$$

The t-th tree is based on the following as inputs:

$$r_{\mu,t}=-\frac{\partial L(y^{(\mu)},f^{(t-1)}(\boldsymbol{x}^{(\mu)}))}{\partial f^{(t-1)}(\boldsymbol{x}^{(\mu)})}. \tag{2.89}$$

Therefore, the tree at iteration t is based on the previous $t-1$ trees and the corresponding error in the loss function L. For instance, when using loss function (2.17), the tree t-th is fitted on the residuals $y^{(\mu)}-f^{(t-1)}$ [36]. With this in mind, it could be easier to grasp the informal description of combining several weak learners into a strong learner. Additionally, it is worth noting that categorical features are not represented by one-hot encoding. When splitting a categorical feature with cardinality c, its groups are divided into two different subsets according to the optimal split amongst $2^{(c-1)}-1$ different combinations [45].

2.5 Evaluation metrics

The performance of binary classification tasks can be measured in terms of true positives (TP), false positives (FP), true negatives (TN) and false negatives (FN), and in terms of the indicator function I:

$$TP=I(\lceil O^{(\mu)}\rceil=1=t^{(\mu)}),\ O^{(\mu)}>0.5, \tag{2.90}$$

$$FP=I(\lceil O^{(\mu)}\rceil=1\neq t^{(\mu)}),\ O^{(\mu)}>0.5, \tag{2.91}$$

$$TN=I(\lfloor O^{(\mu)}\rfloor=0=t^{(\mu)}),\ O^{(\mu)}<0.5, \tag{2.92}$$

$$FN=I(\lfloor O^{(\mu)}\rfloor=0\neq t^{(\mu)}),\ O^{(\mu)}<0.5. \tag{2.93}$$

A positive prediction corresponds to one and a negative corresponds to zero. The accuracy is defined as:

$$\frac{TP + FP}{TP + FP + TN + FN}. \tag{2.94}$$

The accuracy can be misleading for unbalanced data sets, and therefore the F-score can be more useful:

$$\frac{TP}{TP + \frac{1}{2}(FP + FN)}. \tag{2.95}$$

In this thesis, the mean squared error (MSE), Equation (2.96), and the mean absolute percentage error (MAPE), Equation (2.97), will be useful:

$$\frac{1}{N_1}\sum_{i=1}^{N_1}(O_i - L_i)^2, \tag{2.96}$$

$$\frac{1}{N_2}\sum_{i=1}^{N_2}\frac{|O_i - L_i|}{L_i},\ L_i \neq 0, \tag{2.97}$$

where L_i is a real-valued target value and O_i is a real-valued prediction.

3
Methods

This chapter starts with a description of the problem addressed in the thesis. Subsequently, the data is presented, followed by a description of the workflow of the forecast and the method of the hyperparameter tuning. The models were implemented using Python and the deep-learning library PyTorch [46], and to some extent, the Machine Learning library Scikit-Learn [47].

3.1 Problem specification

The problem consists of two fundamental elements: bookings i with length l_i in meters and sailings j. The time of evaluation t corresponds to the time until departure of j and I denotes the indicator function. Thus, $I_{NS}^{(j,t)}(i)$ represents a Bernoulli random variable describing whether booking i will become non-shippable (NS) between the time of evaluation t and the corresponding departure time $t = 0$:

$$I_{NS}^{(j,t)}(i) = \begin{cases} 1, & i \in NS \\ 0, & i \notin NS. \end{cases} \tag{3.1}$$

The probability of this event occurring is measured by $\hat{p}_{ji}$. A booking has a unique booking number and thus $i \to j$, meaning that a booking i also identifies the sailing j, at least at time until departure t.

The random variable $S_j^{(t)}$ represents the sum of bookings active at time t that will become non-shippable before the departure, measured in length meters:

$$S_j^{(t)} = \sum_i l_{ji} I_{NS}^{(j,t)}(i) \tag{3.2}$$

It is modelled using a Poisson-binomial distribution (Section 2.1) with the corresponding estimated mean $\hat{\mu}_{jt}$ (2.12), variance $\hat{\sigma}_{jt}^2$ (2.13) and skewness $\hat{\gamma}_{jt}$ (2.14).

The main objective of this thesis is to predict $S_j^{(t)}$ as accurately as possible, which will be evaluated with the metrics mean squared error (2.96) and mean absolute percentage error (2.97) in terms of the observed values $s_j^{(t)}$. Additionally, the accuracy (2.94) in the prediction of $I_{NS}^{(j,t)}(i)$ will be measured, but it is not often used in practice according to the collaborating company. The performance will be compared to a LightGBM model tuned by the collaborating company and applied with the normal approximation of the Poisson-binomial distribution. The benchmark model is trained on the structured data of individual bookings described in Section 3.2.

Section 1.1 briefly presents two research papers where predictive models of non-shippable bookings in the shipping industry were developed. One of the models is based on data from individual bookings and the other is based on data at an aggregated sailing level. The predictive model developed in this thesis takes into account both information of individual bookings and information from the aggregated sailing level to forecast $S_j^{(t)}$.

3.2 Data

The data used in the thesis was provided by Stena Line and it included data from their sailings in 2022. In order to predict $S_j^{(t)}$ at any given time before departure, t had to be included in the predictive model. Therefore, data had to be gathered from a set of arbitrary times $T = \{t_1, t_2, ..., t_n\}$. For this thesis, T was selected as {28d, 27d, ..., 2d, 40h, 32h, ..., 24h, 20h, ..., 8h, 6h, 4h, 210m, 180m, ..., 60m, 45m, 30m, 20m, 10m, 5m, 1m}, which captured a smooth relationship between shippable and non-shippable bookings, at least for route A, as shown in Figure 3.1. Collecting more data closer to the departure time was avoided as it would have affected the balance of the data. This was because shippable bookings were over-represented from approximately 72 hours until departure. In the data set for route A, there were approximately four times as many shippable bookings as non-shippable bookings. It should be noted that a booking consists of a sequence of versions, but for the purpose of this thesis, only the latest version at a given time was considered. This decision was based on the assumption that the latest version contained the majority of the booking information and it helped to reduce the complexity of the data. Additionally, the natural logarithm of the time until the departure was used in order to make the time feature less skewed.

Table 3.1: Overview of the data used in the thesis.

Route	Number of samples	Number of sailings	Label ratio 0:1
A	1.831.165	1538	3.99

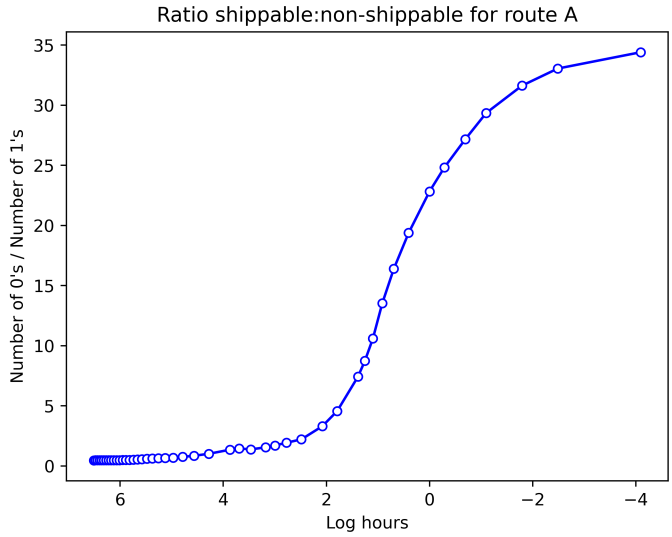

Figure 3.1: Relationship between shippable and non-shippable bookings for route A, plotted against the natural logarithm of the time until departure. The ratio starts to increase rapidly around 8 hours until departure and the break-even point is around 72 hours until departure. All samples in the data set for route A are included in the plot.

Tables 3.2 and 3.3 list the features of the data set. While most of the features are self-explanatory, the booking status code and the origin code may require some further clarification. The booking status could, for instance, be confirmed and not confirmed, and the origin code could, for instance, be externally booked or booked by an employee at the collaborating company.

The categorical features were modelled using entity embedding, as presented in Section 2.2.7, and initialised according to the default method in PyTorch, $N(0, 1)$. For customers not included in the training data but present in the validation sets and prediction set, their corresponding embedding was set to the average of the weights for the customers included in the training data.

To visualise the distribution of the embedding space for each categorical feature, principal component analysis (PCA) was used. PCA projects the samples $\boldsymbol{x_i}$ in the weight matrix of the entity embedding, denoted as $\boldsymbol{X}$, onto a unit vector $\hat{\boldsymbol{v}}$. The vector $\hat{\boldsymbol{v}}$ is called a principal component. This transformation serves to reduce the dimensionality of the data while preserving as much information as possible, by finding the directions (principal components) that describe the most variance in the data [13]. In this thesis, the first and second principal component was used to visualise the entity embedding space. The first and second principal components correspond to the two directions which describe the most variance of the data. In mathematical terms:

$$\underset{\hat{v}}{max}\ \sigma^2_{\hat{\boldsymbol{v}}} = \underset{\hat{v}}{max}\ \frac{1}{N}\sum_i (\hat{\boldsymbol{v}}^T \boldsymbol{x}_i - \hat{\boldsymbol{v}}^T \bar{\boldsymbol{x}})^2 = \underset{\hat{v}}{max}\ \hat{\boldsymbol{v}}^T \boldsymbol{C} \hat{\boldsymbol{v}}, \tag{3.3}$$

where $\boldsymbol{C}$ is the sample covariance matrix of the centred data:

$$\boldsymbol{C} = \frac{1}{N}\sum_{i}(\boldsymbol{x}_i - \bar{\boldsymbol{x}})(\boldsymbol{x}_i - \bar{\boldsymbol{x}})^T. \tag{3.4}$$

It is well known that $\hat{\boldsymbol{v}}_\mathbf{1}$ and $\hat{\boldsymbol{v}}_\mathbf{2}$ correspond to the eigenvectors of $\boldsymbol{C}$ with the largest and second largest eigenvalue.

Table 3.2: Continuous features of the data set.

Features	Unit
Version	
Occupied length	Meters
Weight	Kilograms
Number of changes internally	
Number of changes externally	
Natural logarithm of the time until departure	Log hours
Number of previous transfers of the booking	
Mean of the time until departure of previous transfers	Hours
Time difference between the first version and departure	Hours
Time since the last change	Hours
Time since the last transfer	Hours

Table 3.3: Categorical features of the data set. The cardinality is for route A, this to give an example of the number of unique values for each categorical feature.

Features	Cardinality - Route A
Sailing month	12
Sailing weekday	7
Sailing hour	24
Origin code	5
Booking status code	4
Check-in status code	3
Customer number	886
Vehicle type	19

Table 3.4: The sequential data used in the thesis. The non-shippable (NS) ratio is defined as the number of known cancellations and transfers from a sailing at a given time t before departure, divided by the total number of bookings known at t. The time of evaluation, which is equivalent to the time until departure, is transformed with the natural logarithm.

Features	Unit
Cumulative non-shippable ratio	
Natural logarithm of the time until departure	Log hours

Table 3.4 lists the features included in the sequential data. Since the data was gathered at the times until departure t_i in T, t_i was also provided in the modelling, but transformed with the natural logarithm. To reduce the time required to gather the data, T was used for the sequential data as well, and the data was summarised on an aggregated sailing level. The known cumulative non-shippable ratio at a given time before departure was calculated by dividing the sum of known cancellations and transfers by the total number of bookings. It should be noted that this is different from the ratio of future non-shippable bookings amongst the active bookings at a given time, shown in Figure 3.1, which is the label of interest for prediction. The input of the samples consisted of two sequences containing the previous time instances transformed with the natural logarithm and the corresponding cumulative non-shippable ratios. All time instances in T were used for all sailings, resulting in a total of 73776 samples for route A.

Except for standardising the continuous features, the data was used without filtering out features. The reason was that the data already contained few features and the algorithms presented in this thesis should ideally use near-zero weights for redundant features. However, the benchmark model and the normal approximation presented in Section 2.4 and 2.1, respectively, were used to visualise the performance of different subsets of features. There were exponentially many combinations, thus it was not reasonable to test all possible subsets. Two greedy techniques were tested, forward selection and backward selection. Backward selection starts with all features and in each iteration removes the one which reduces the considered performance measure the least. Forward selection starts with no features and in each iteration adds the feature which increases the considered performance measure the most. This method was used because standard methods based on the well-known p-value in the context of big data and almost two million samples are useless. The interested reader is referred to Lin, Lucas, and Shmueli [48] to read about the problem with p-values in the context of Big Data. Furthermore, the benchmark model is computationally efficient and this made the method reasonable in terms of running time.

In order to utilise early stopping, which was described in Section 2.2.5, the data set of each route was divided into a training set (70 %), two validation sets (10 % each) and a test set (10 %), with no overlapping sailings. A booking version could be active at two different times before the departure and both were included in the data set in order to include the time of evaluation as a feature. Therefore are the number of samples not equivalent to the number of unique bookings. This could be considered to be a naive sampling method and the author of the thesis recognises that this could be an area of improvement if the predictive model is trained and used in practice.

3.3 The workflow of the forecasting procedure

The workflow of the forecasting procedure was inspired by the work of Wang and Meng [7] which was presented briefly in Section 1.1. The authors utilised an approach

of stacked generalisation to incorporate three different models, where one modelled sequential time series data. The final model in this thesis is a stacked model of two different approaches, an ensemble of ANNs based on structured data of individual bookings and a gated RNN based on cumulative cancellation and transfer rates known at a given time before departure. The workflow is visualised in Figure 3.2. It is worth mentioning that $\hat{\mu}_{jt}$, $\hat{\sigma}^2_{jt}$ and $\hat{\gamma}_{jt}$ from the ANN ensemble and $\hat{r}_{jt}$ from the gated RNN were standardised before used as input in the stacked model.

The output of ANNs varied significantly since different iterations converged to different local minima, for instance, because of the stochastic gradient descent and random initialisation of the weights. As described in Section 2.2.8, Lakshminarayanan, Pritzel, and Blundell [39] showed empirically that an average of ANNs can be more calibrated in comparison to a single ANN. Since the estimated probabilities of a booking becoming non-shippable are multiplied by the corresponding booking length, a small change in the estimated probabilities can potentially affect the final output significantly. Thus it seemed reasonable to not rely on a single ANN. However, the downside of using an ensemble is that it leads to increased training time.

The reason for utilising some type of RNN was to explore the hypothesis of a temporal dependency between known non-shippable bookings at a given time and future non-shippable bookings on a sailing, much like the usage of an AR-model in [7]. The target variable was defined as the fraction of non-shippable bookings amongst the active bookings at a given time. The reason why ratios were used as both input and output was to normalise the sailings. Therefore, in order to predict future non-shippable bookings in length meters, the output from the model was multiplied by the total length of the active bookings at the time.

The predicted mean, variance and skewness were provided for the stacked model of a simple ANN. As previously mentioned in Section 2.1, Hong [8] suggests that the normal approximation of a Poisson-binomial distribution may not accurately reflect highly skewed data.

The last approach involved utilising linear regression trained on $\hat{\mu}_{jt}$, $\hat{\sigma}^2_{jt}$ and $\hat{\gamma}_{jt}$ from the ANN ensemble with $s_j^{(t)}$ as the dependent variable. This approach was used instead of relying on the normal approximation. Linear regression corresponds to modelling the outputs y_{jt} as [13]:

$$y_{jt} = \boldsymbol{x_{jt}}^T \boldsymbol{\beta} + \epsilon_{jt}, \tag{3.5}$$

where $\boldsymbol{x_{jt}}^T = (1, \hat{\mu}_{jt}, \hat{\sigma}^2_{jt}, \hat{\gamma}_{jt})^T$ is the input data of sample i as a row vector, the regression coefficients $\boldsymbol{\beta}$ as a column vector and the error term e_{jt} assumed to be normally distributed with mean 0 and some variance $\bar{\sigma}^2_{jt}$. Minimising the error terms $\boldsymbol{\epsilon} = \boldsymbol{X\beta} - \boldsymbol{y}$, where $\boldsymbol{X}$ is a matrix with $\boldsymbol{x_i}^T$ as the i-th row, with the sum of squared errors (2.17) leads to the well know least-squares solution:

$$\hat{\boldsymbol{\beta}} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}. \tag{3.6}$$

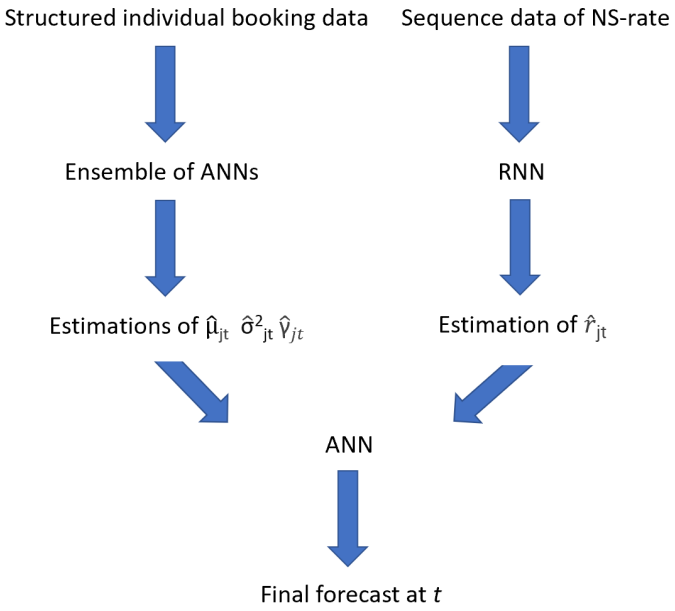

Figure 3.2: The workflow of the forecast procedure. The continuous features were standardised before the data was used. $\hat{\mu}_{jt}$, $\hat{\sigma}^2_{jt}$ and $\hat{\gamma}_{jt}$ from the ANN ensemble and $\hat{r}_{jt}$ from the gated RNN were also standardised before being fed into the stacked model.

3.4 Hyperparameter tuning

The hyperparameter tuning of the models was performed using the Python library Ray Tune. Due to limited computational resources, the Asynchronous Successive Halving Algorithm (ASHA) scheduler was employed [49]. This algorithm is more efficient than running each trial to the maximum number of epochs since it discards underperforming hyperparameter settings in the early stages of training. In brief, the combinations of hyperparameter settings were divided into groups, periodically evaluated, and a fraction of each group (in this case, 1/2) was eliminated. Parallelisation was utilised to take advantage of available computing capacity. The assumption is that settings that perform poorly in the early stages of training continue to underperform in later epochs. However, a potential drawback of the method is that promising settings may be discarded before convergence. Hyperparameter settings were sampled using random search, and the evaluation metric for tuning was the total loss of the first validation set. After evaluating 1000 different combinations, the hyperparameter setting with the lowest validation loss at any checkpoint was chosen. The validation loss was reported 10 times each epoch, and the maximum number of epochs was set to eight.

The hyperparameter search space for the ANNs architecture in the ensemble can be found in Table 3.5. The activation function was either *tanh* (2.50) or *ReLU* (2.51). Equation (2.56) was used to initialise *tanh* neurons and Equation (2.55) for *ReLU* neurons, either based on the normal or uniform distribution, resulting in two combinations conditioned on the type of activation function. In Section 2.2.6 are only

two of the functions presented, the other two can be found in [50]. The maximum embedding size is denoted by m in Equation (2.58), and the entity embedding of the categorical variables was concatenated with the continuous variables to form the input layer, following the approach used in [35], [36].

Table 3.5: Hyperparameter search space for the ANNs in the ensemble. η and γ are uniformly distributed. The weight initialisation methods (2.55) and (2.56) can be used with a uniform distribution and normal distribution, the two other combinations can be found in [50]. The Stochastic Gradient Descent (SGD) was used with Nesterov momentum [51].

Hyperparameter	Search space
Learning rate η	[0.0001, 0.01]
Weight decay γ	[0, 0.01]
Number of hidden layers	{1,2,3,4,5}
Number of neurons in the hidden layers	{1, 2, ..., 100}
Activation function $g()$	$\{ReLU, tanh\}$
Initialisation distribution	{Normal, Uniform}
Training mini batch size m_B	{16, 32, 64, 128, 256}
Maximum embedding size	{2,3, ..., 50}
Optimization method	$\{SGD, Adam(W)\}$

The output from the ANNs in the ensemble is the estimated probabilities of the bookings becoming non-shippable, denoted as $\hat{p}_{ji}$. Hence, the binary cross entropy (BCE) (2.18) was chosen as the loss function. However, an alternative approach was considered, involving multiplication of $\hat{p}_{ji}$ with the corresponding length l_{ji}. In this approach, the targets were defined as the length of the actual non-shippable bookings. The mean of squared errors (MSE) was then used as the loss function. It corresponds to Equation (2.17) divided by the batch size m_B. By multiplying $\hat{p}_{ji}$ with l_{ji}, errors in the forecast of longer vehicles contributed more to the total loss. Both approaches were compared in the study to evaluate their respective performances and determine the most effective approach.

Table 3.6: Hyperparameter search space for the RNN. η and γ are uniformly distributed.

Hyperparameter	Search space
Learning rate η	[0.0001, 0.01]
Weight decay γ	[0, 0.01]
RNN type	{RNN, GRU, LSTM}
Number of hidden layers	{1,2,3}
Number of neurons in the hidden layers	{1,2, ..., 10}

The hyperparameter search space for the RNN can be found in Table 3.6. The recurrent neural networks were trained with the Adam(W) optimiser and a constant training batch size of 32 due to the time-consuming process of preparing the sequential data. Sequence bucketing was used as the sampling technique for the batches,

meaning that each batch only contained a single sequence length with randomly selected samples from the subset of the given sequence length. The hidden states $V_i(0)$ were zero initialised, and so were the LSTM internal cell states $s_i(0)$. The weights and thresholds were initialised with the default function in PyTorch, which corresponds to Equation (2.56) with n_l equal to the number of neurons in the hidden layer. The validation loss was reported 10 times each epoch, and the maximum number of epochs was set to 10.

Although the search space for the ANN and RNN could be extended, the presented search spaces were considered large enough for the purpose of this problem, given the relatively small number of features in each data set. The search space for the hyperparameters in the stacked ANN is provided in Table 3.7. The training batch size was set to 32, a single hidden layer was used, and the AdamW optimiser was used.

Table 3.7: Hyperparameter search space for the stacked ANN visualised in Figure 3.2. η and γ are uniformly distributed. The batch size of 32 was fixed, as well as the number of hidden layers, which was set to one.

Hyperparameter	Search space
Learning rate η	[0.001, 0.1]
Weight decay γ	[0, 0.01]
Number of neurons in the hidden layer	{1,2, ..., 10}

3.5 Variance prediction and error analysis

The variance of $S_j^{(t)}$ with uncertain probabilities $P_{j1}, P_{j2}, ..., P_{jN}$ could be an important consideration for developing an overbooking strategy in the freight ferry industry. This thesis compared two approaches, where both of them avoid calculating the joint probability distribution of $P_{j1}, P_{j2}, ..., P_{jN}$.

The first approach involved treating the estimated probabilities $\hat{p}_{ji}$ as certain values. Equation (2.13) was used with 10 iterations to estimate the variance. This approach provided a simplified estimation of the variance.

The second approach considered the uncertainty of the probabilities. This approach recognised that the probabilities themselves are random variables. The approach was based on Equation (2.15) and can be described as follows:

1. Train M ANNs.
2. Estimate $E[Var[S_j^{(t)}|P_{j1}, P_{j2}, ..., P_{jN}]]$ with $\hat{\sigma}_j^2 = \sum_{m=1}^{M} \sum_i l_{ji}^2 \hat{p}_{jim}(1-\hat{p}_{jim})$ and the M instances of each estimate of P_{ji}.
3. Estimate $Var[E[S_j^{(t)}|P_{j1}, P_{j2}, ..., P_{jN}]]$ with the sample variance of the predicted means $\frac{1}{M-1}\sum_{m=1}^{M}(\hat{\mu}_{jm}-\bar{\mu}_j)^2$ where $\hat{\mu}_{jm} = \sum_i l_{ji}\hat{p}_{jim}$ and $\bar{\mu}_j = \frac{1}{M}\sum_{m=1}^{M}\sum_i l_{ji}\hat{p}_{jim}$.

4. Estimate $Var[S_j^{(t)}] = E[Var[S_j^{(t)}|P_{j1}, P_{j2}, ..., P_{jN}]] + Var[E[S_j^{(t)}|P_{j1}, P_{j2}, ..., P_{jN}]]$ with step 2 + step 3.

Where M=10 in this thesis.

Ideally, $Var[S_j^{(t)}]$ for sailing should be close to the corresponding mean squared error of the k predictions:

$$\frac{1}{K}\sum_{k=1}^{K}(\hat{s}_{jk}^{(t)} - s_{jk}^{(t)})^2 \tag{3.7}$$

Where K is the number of compared samples. However, comparing the predicted variance and the mean squared error for $S_j^{(t)}$ was challenging due to the uniqueness of each sailing. The random variable $S_j^{(t)}$ corresponds to a specific sailing at a given time before departure, and there is only one corresponding observed value, $s_j^{(t)}$. To address this issue, a normalisation step was applied to the sailings by dividing them by the total booked length at time t for sailing j, denoted as $|l_j^{(t)}|$. This normalisation allowed for a more meaningful comparison between the predicted variance and the mean squared error across different sailings. The mean, variance and skewness of $S_j^{(t)}/|l_j^{(t)}|$ with certain probability estimates $\hat{p}_{ji}$ could be expressed with Equations (3.8)-(3.10):

$$\hat{\mu}_{S_j^{(t)}/|l_j^{(t)}|} = \frac{1}{M}\sum_{m=1}^{M}\frac{1}{|l_j^{(t)}|}\sum_{i=1}^{N} l_{ji}\hat{p}_{jim}, \tag{3.8}$$

$$\hat{\sigma}^2_{S_j^{(t)}/|l_j^{(t)}|} = \frac{1}{M}\sum_{m=1}^{M}\frac{1}{|l_j^{(t)}|^2}\sum_{i=1}^{N} l_{ji}^2\hat{p}_{jim}(1-\hat{p}_{jim}), \tag{3.9}$$

$$k_{S_j^{(t)}/|l_j^{(t)}|} = \frac{\frac{1}{M}\sum_{m=1}^{M}\frac{1}{|l_j^{(t)}|^3}\sum_{i=1}^{N} l_{ji}^3\hat{p}_{jim}(1-\hat{p}_{jim})(1-2\hat{p}_{jim})}{\hat{\sigma}^3_{S_j^{(t)}/|l_j^{(t)}|}}. \tag{3.10}$$

Similar sailings were grouped into predefined intervals based on the normalised predicted variance. The chosen intervals were as follows:

$B = \{0-0.005, 0.005-0.01, 0.01-0.02, 0.02-0.03, 0.03-0.04, 0.05-0.1, 0.1-0.15, 0.15-0.2, 0.2-0.25\}$.

For each interval, the mean squared error and the predicted variance were calculated and compared across six different random seeds.

4

Results

This chapter first presents the result of the forward and backward selection of the features, followed by presenting the best-performing hyperparameter settings from the tuning experiments. The chapter ends with presenting forecast results and visualisations of the entity embedding space of the categorical features.

4.1 Forward and backward selection

The left and right frames of Figure 4.1 show the result of the forward and backward selection, respectively. The result indicates that six features have a significant impact on the metrics for the data of route A: time of evaluation, sailing hour, sailing weekday, check-in status code, customer number and sailing month. The origin code seems to provide a slight improvement. The inclusion of customer numbers as a feature contributes to the increase in the performance of binary classification of individual bookings, measured in accuracy and F1-score. On average, the benchmark model can classify almost 9 out of 10 bookings correctly. Some of the important features are related to individual booking information, while others pertain to general information about the corresponding sailing.

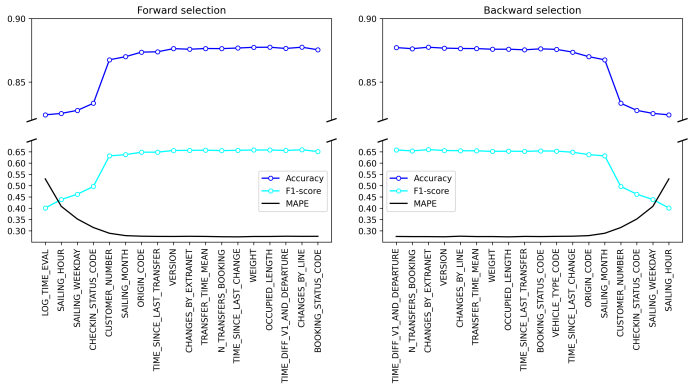

Figure 4.1: Forward (left frame) and backward (right frame) selection with the benchmark model on the data set for route A. The metrics are based on the first validation set and the model was trained on the training data. The x-axis shows which feature was either added or removed. The selection process was based on the metric MAPE. Note that the y-axis is not continuous.

4.2 Hyperparameters

Table 4.1 presents the hyperparameters associated with the lowest validation loss for the ANN. Several instances of this architecture were used in the ANN ensemble. Figures A.1-A.5 illustrate that the network dynamics converge within the first epoch. Figure 4.2 visualises the performance of the ANN ensemble in terms of MSE and MAPE on the first validation set as the number of ANNs increases. The results indicate that the prediction performance of different ANNs varies, and an average could be more reliable.

To assess the convergence of the predicted mean, variance and skewness, the sum of the absolute differences obtained using N and $N-1$ ANNs were compared. The probabilities were in this case treated as certain values. A sum close to zero indicates convergence. The result, illustrated in Figure 4.3, indicates that the predicted mean has not fully converged with 20 instances in the ensemble. For the purpose of this thesis, 10 instances were used.

Table 4.1: Hyperparameters for the ANNs in the ensemble.

Hyperparameter	Value
Learning rate η	0.0008
Weight decay γ	0.009
Number of hidden layers	3
Number of neurons in the hidden layers	56
Activation function $g()$	$tanh$
Initialisation distribution	Uniform
Training mini-batch size m_B	32
Maximum embedding size	13
Optimization method	AdamW

Table 4.2 presents the hyperparameters associated with the lowest validation loss for the RNN. Table 4.3 presents the hyperparameters associated with the lowest validation loss on the second validation set for the stacked ANN based on the ANN ensemble and the GRU network. The monitoring metrics of the training procedure of the neural networks are presented in Appendix A.

Table 4.2: Hyperparameters for the RNN. Trained with the Adam(W) optimiser and training batch size 32.

Hyperparameter	Value
Learning rate η	0.003
Weight decay γ	0.005
Number of hidden layers	1
Number of neurons in the hidden layer	8
RNN type	GRU

Table 4.3: Hyperparameters for the stacked ANN. Trained with the Adam(W) optimiser.

Hyperparameter	Value
Learning rate η	0.05
Weight decay γ	0
Number of hidden layers	1
Number of neurons in the hidden layer	5
Activation function $g()$	$tanh$
Initialisation distribution	Uniform
Training mini-batch size m_B	32

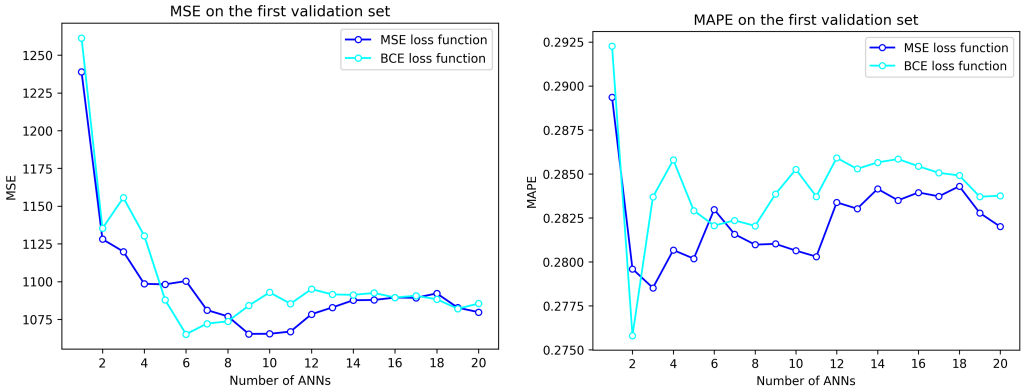

Figure 4.2: MSE and MAPE on the first validation set plotted against the number of ANNs in the ensemble. The results are based on the normal approximation of the Poisson-binomial distribution.

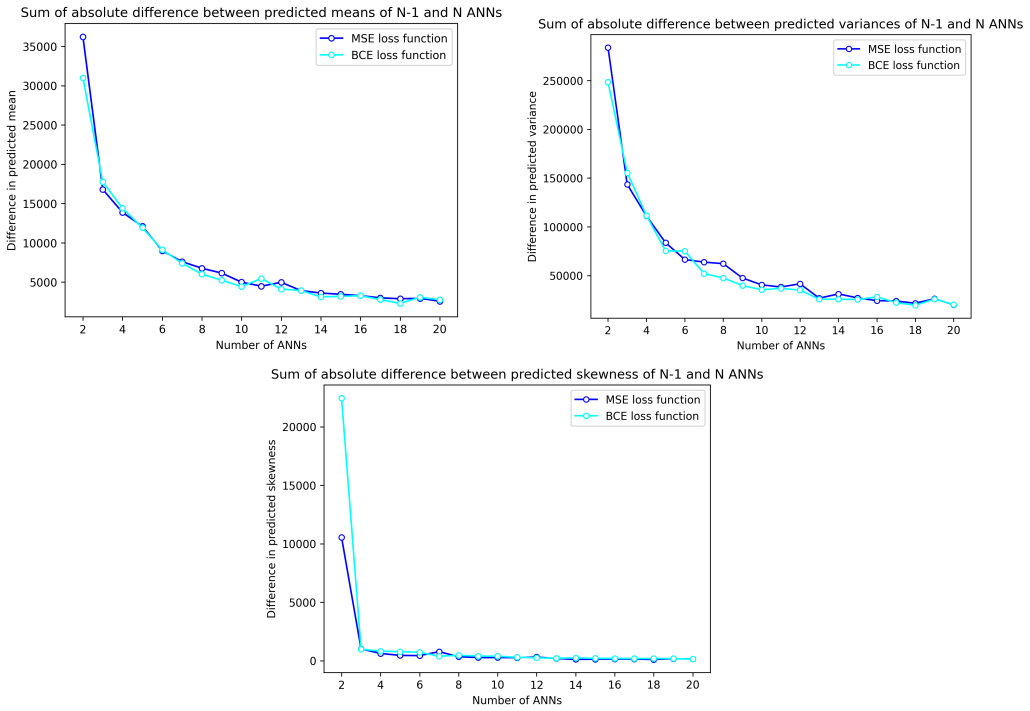

Figure 4.3: Difference between the sum of absolute differences of the predicted mean, variance and skewness of N and N-1 ANNs in the ensemble. The probabilities were treated as known values.

4.3 Forecasting

The results of the forecast performance for the different predictive models on the second validation set and the test set are in Tables 4.4 and 4.5. In this thesis, the linear regression approximation method for the Poisson-binomial distribution is referred to using linear regression to predict $S_j^{(t)}$ based on $\hat{\mu}_{jt}$, $\hat{\sigma}_{jt}^2$ and $\hat{\gamma}_{jt}$ from the ANN ensemble. This method, when trained on observed values $s_j^{(t)}$ in the first validation set with the MSE loss function, yielded the following:

$$\hat{s}_j^{(t)} = 0.967 + 1.0180\hat{\mu}_{jt} - 0.00264\hat{\sigma}_{jt}^2 + 0.324\hat{\gamma}_{jt}. \tag{4.1}$$

And with BCE loss function:

$$\hat{s}_j^{(t)} = 0.679 + 1.0352\hat{\mu}_{jt} - 0.00623\hat{\sigma}_{jt}^2 + 0.216\hat{\gamma}_{jt}. \tag{4.2}$$

Table 4.4: Forecast result on the second validation set of route A.

Model	MSE	MAPE	Accuracy	F1-score
LightGBM + Normal	947	0.295	0.885	0.676
ANN ensemble MSE + Normal	956	0.301	0.886	0.675
ANN ensemble BCE + Normal	967	0.306	0.885	0.675
ANN ensemble MSE + Lin. reg.	955	0.305		
ANN ensemble BCE + Lin. reg.	941	0.298		

Table 4.5: Forecast result on the test set of route A.

Model	MSE	MAPE	Accuracy	F1-score
LightGBM + Normal	1246	0.275	0.884	0.679
ANN ensemble MSE + Normal	1440	0.293	0.881	0.667
ANN ensemble BCE + Normal	1362	0.303	0.881	0.667
ANN ensemble MSE + Lin. reg.	1324	0.296		
ANN ensemble BCE + Lin. reg.	1329	0.293		
RNN + ANN ensemble MSE	1500	0.313		
RNN + ANN ensemble BCE	1548	0.275		

Figure 4.4 visualise the point predictions of the mean, variance and skewness for one specific sailing in the first validation set. The predicted mean, variance and skewness are shown as the average in black.

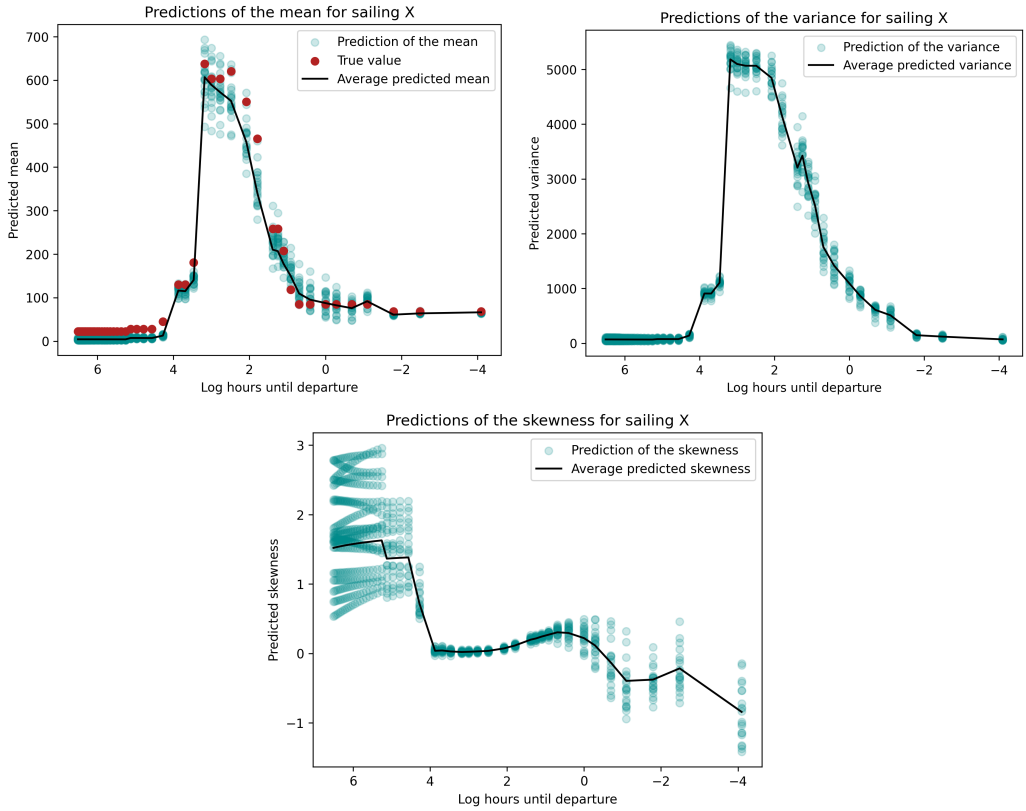

Figure 4.4: Point predictions of the mean, variance and skewness of $S_j^{(t)}$ for one of the sailings in the first validation set with the ANNs in the ensemble. The BCE loss function was used, and the probabilities were treated as known values.

Figure 4.5 illustrates the forecast performance of the ANN ensembles and the LightGBM model using the normal approximation of the Poisson-binomial distribution on the second validation set. Additionally, Figure 4.6 shows the corresponding performance with predictions adjusted using the linear regression based on the predicted mean, variance and skewness from the ANN ensembles.

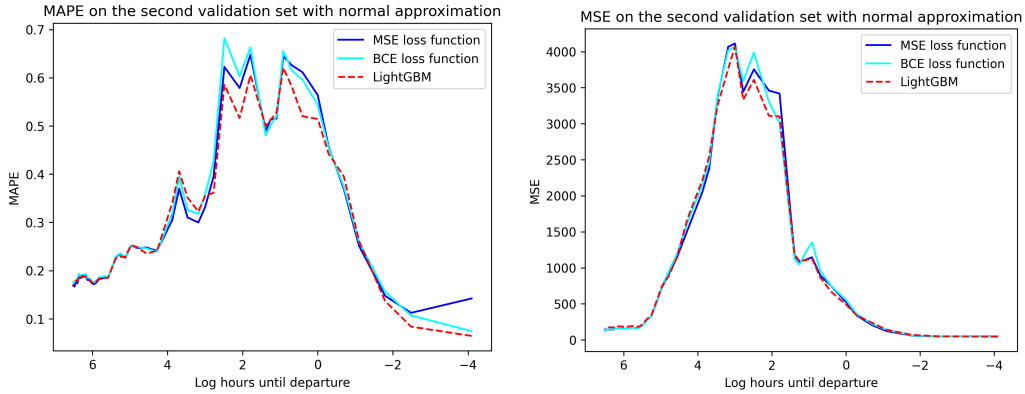

Figure 4.5: MAPE and MSE on the second validation set with the normal approximation of the Poisson-binomial distribution.

The normalised predicted mean, variance and skewness for all sailings in the first validation set using the ANN ensemble, trained with the BCE loss function, are visualised in Figure 4.7. The probabilities were in this case assumed to be known, and therefore were Equations (3.8)-(3.10) used. The corresponding Figures of 4.4 and 4.7 with the MSE loss function are presented in Appendix B.

Table 4.6 shows the forecast result across six random seeds, with the mean and sample standard deviation presented. The approach of the stacked ANN based on the RNN and the ANN ensemble was not included in the test due to extensive training and preprocessing.

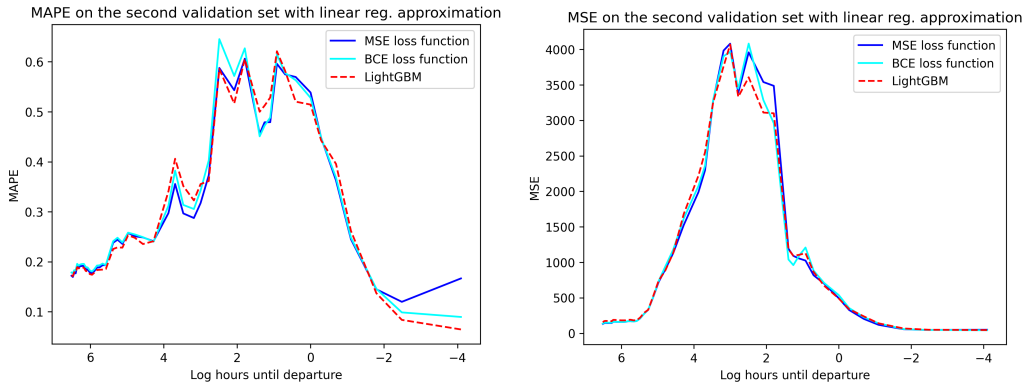

Figure 4.6: MAPE and MSE on the second validation set with linear regression approximation of the Poisson-binomial distribution. In this thesis, this approximation method is referred to using $\hat{\mu}_{jt}$, $\hat{\sigma}^2_{jt}$ and $\hat{\gamma}_{jt}$ to predict $S_j^{(t)}$ with linear regression.

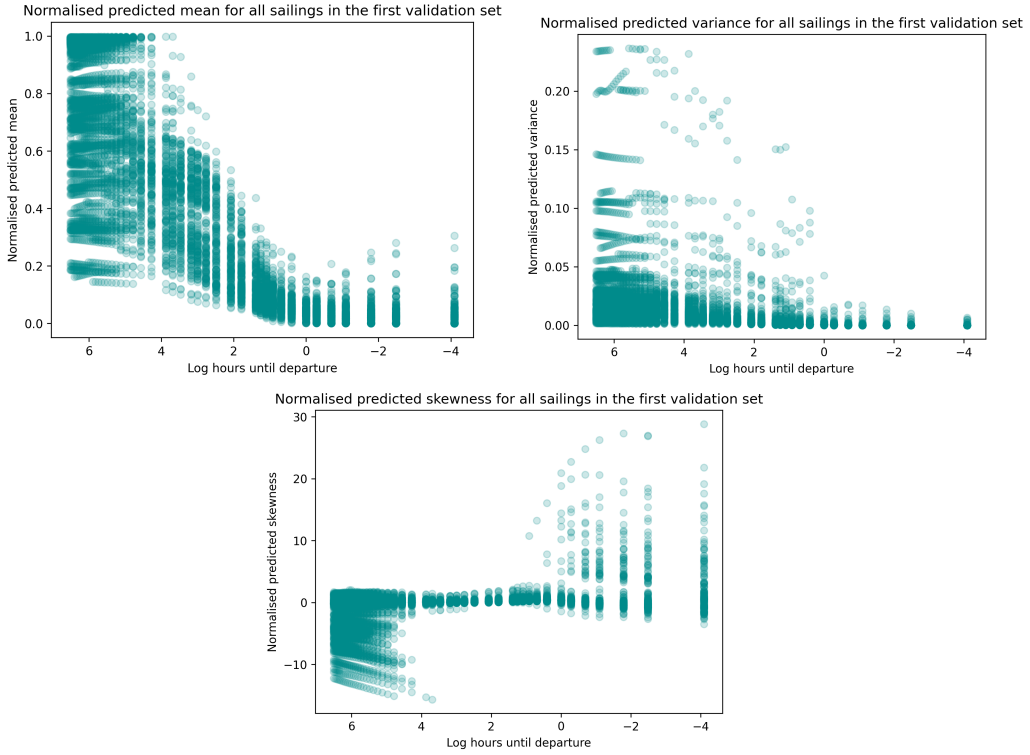

Figure 4.7: Normalised predicted mean, variance and skewness for all sailings in the first validation set. The normalisation was performed based on the total length booked at time t on sailing j. Hence, the variable of interest is $S_j^{(t)}/|l_j^{(t)}|$, where $|l_j^{(t)}|$ is the booked length at time t on sailing j. The BCE loss function was used.

Table 4.6: Forecast result across 6 random seeds. ANN correspond to an ANN ensemble with 10 instances. (Sample mean, sample standard deviation)

Model	MSE	MAPE	Accuracy	F1-score
LightGBM+N	(1100,139)	(0.289,0.00845)	(0.881,0.00356)	(0.668, 0.0109)
ANN MSE+N	(1150,159)	(0.296,0.00689)	(0.881,0.00306)	(0.661, 0.0125)
ANN BCE+N	(1170,173)	(0.305,0.00482)	(0.881,0.00289)	(0.661,0.0127)
ANN MSE+LR	(1158,162)	(0.298,0.0120)		
ANN BCE+LR	(1174,174)	(0.306,0.0106)		

4.4 Analysis of the predicted variance

Figures 4.8 and 4.9 show the MSE between normalised predictions $\hat{s}_j^{(t)}/|l_j^{(t)}|$ and $s_j^{(t)}/|l_j^{(t)}|$ plotted against the normalised predicted variance. The sailings were divided into the predefined intervals after the normalised predicted variance, as described in Section 3.5. The ANN ensemble approach was used as the predictive model with 10 instances. The left frame presents the result of the first approach where the probabilities $\hat{p}_{ji}$ are treated as certain values. The right frame corresponds to the approach of treating the probabilities as random variables P_{ji}. In this case,

both methods yield similar results. The results are based on the mean and standard deviation of the MSE obtained from six different random seeds. The x-axis represents the average of the normalised predicted variance in each interval. Although the averages varied slightly across the random seeds, this variation is not illustrated in Figures 4.8 and 4.9.

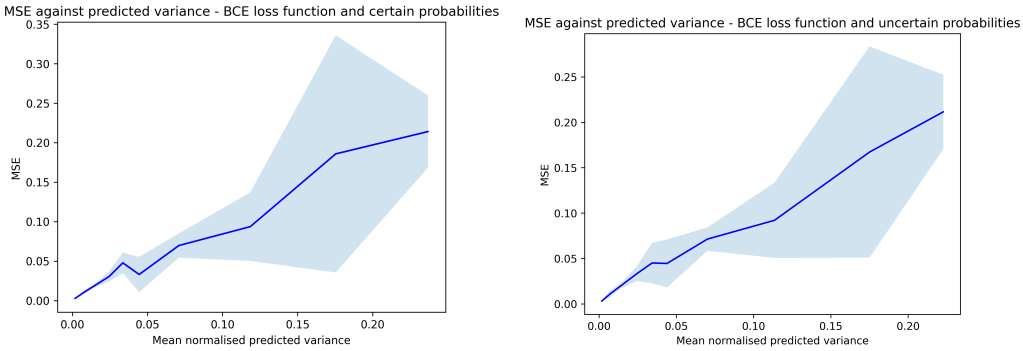

Figure 4.8: Mean squared errors between $\hat{s}_j^{(t)}/|l_j^{(t)}|$ and $s_j^{(t)}/|l_j^{(t)}|$ plotted against the normalised predicted variance, using the ANN ensemble trained with BCE loss function. The x-axis represents the average of the normalised predicted variance in the predefined intervals described in Section 3.5, while the y-axis represents the MSE in the corresponding predefined interval. The approach of certain probabilities corresponds to the left frame, while the approach of uncertain probabilities corresponds to the right frame. The sample standard deviation of the MSE in each predefined interval is shown in light blue colour, calculated over 6 random seeds.

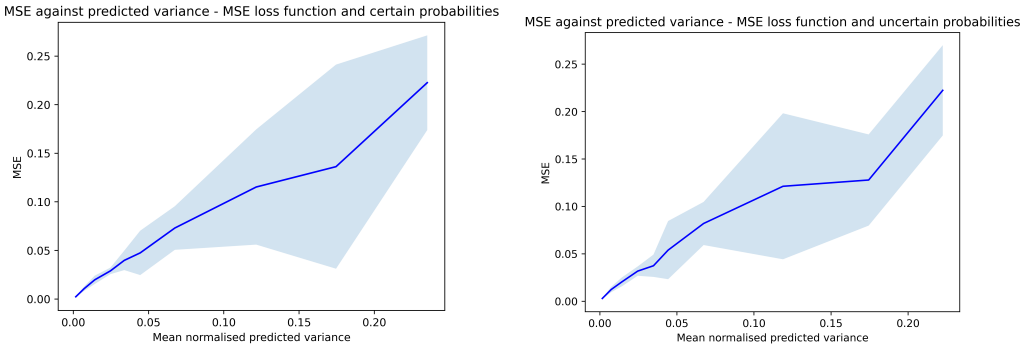

Figure 4.9: Mean squared errors between $\hat{s}_j^{(t)}/|l_j^{(t)}|$ and $s_j^{(t)}/|l_j^{(t)}|$ plotted against the normalised predicted variance, using the ANN ensemble trained with MSE loss function. The x-axis represents the average of the normalised predicted variance in the predefined intervals described in Section 3.5, while the y-axis represents the MSE in the corresponding predefined interval. The approach of certain probabilities corresponds to the left frame, while the approach of uncertain probabilities corresponds to the right frame. The sample standard deviation of the MSE in each predefined interval is shown in light blue colour, calculated over 6 random seeds.

4.5 Distributions in the entity embedding space

Figures 4.10-4.14 visualise the embedding of the categorical variables along the first and second principal components. Figures 4.11-4.14 correspond to the entity embedding of two different ANNs trained with the BCE loss function.

The embedding space varied across the models and it is challenging to visualise all significant information solely based on the first and second principal components, especially for high-cardinality features. However, some patterns appeared consistent across different iterations. For instance, weekend sailings are distinct from the workweek. Clustering can be observed between certain months, such as January, February and March, as well as September and October. The sailing times 15:00 and 16:00 are clustered close together and the same for the early shipping times, before 12:00. Notably, no distinct clustering is observed among the check-in status codes.

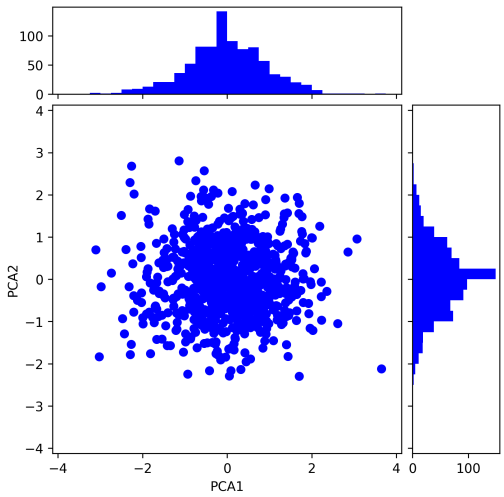

Figure 4.10: Visualisation of the distribution of the customer embedding projected onto the first two principal components.

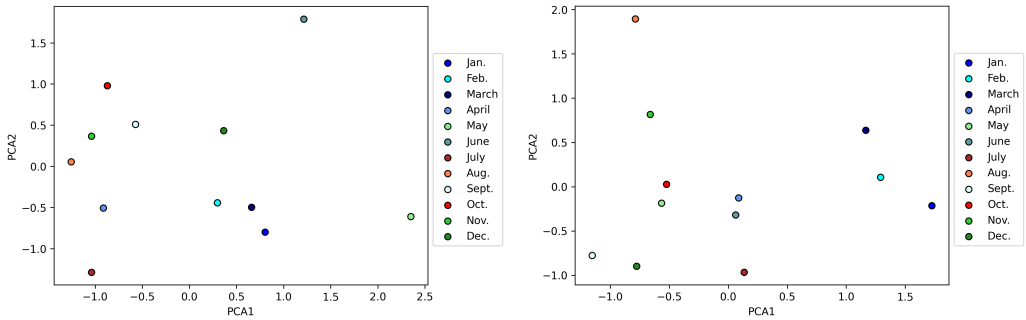

Figure 4.11: Visualisation of the distribution of the sailing month embedding projected onto the first two principal components using two different ANNs trained with BCE loss function.

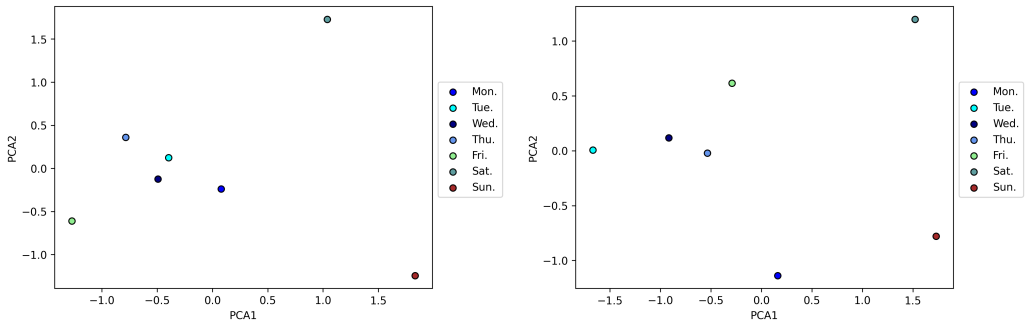

Figure 4.12: Visualisation of the distribution of the sailing weekday embedding projected onto the first two principal components using two different ANNs trained with BCE loss function.

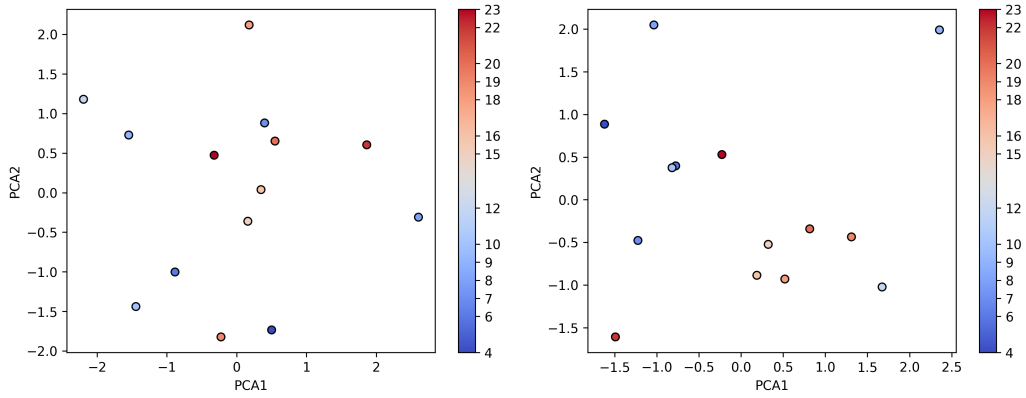

Figure 4.13: Visualisation of the distribution of the sailing hour embedding projected onto the first two principal components using two different ANNs trained with BCE loss function.

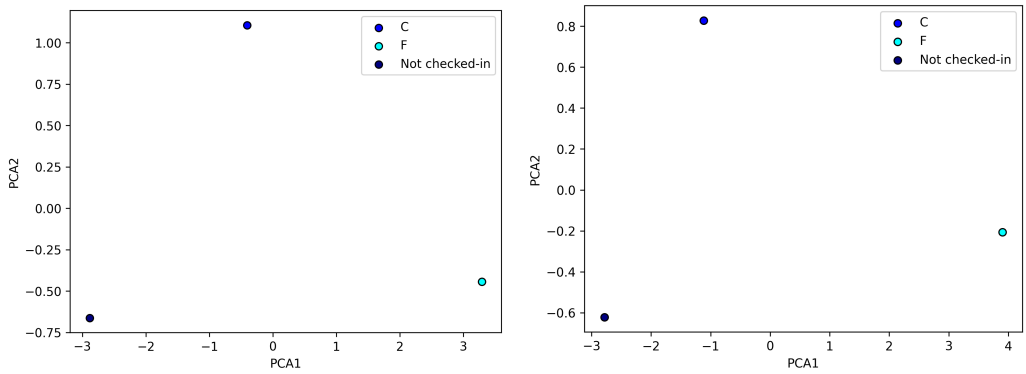

Figure 4.14: Visualisation of the distribution of the check-in code embedding projected onto the first two principal components using two different ANNs trained with BCE loss function.

5
Discussion

This chapter provides a discussion of the results presented in Chapter 4. Firstly, the performance of forecasting the mean and variance of $S_j^{(t)}$ is examined. Thereafter, the interpretation of the entity embedding is discussed and lastly, potential areas for further research.

5.1 Forecasting performance

The forecast results of the different approaches involving ANNs and RNNs demonstrate a slightly worse performance compared to the benchmark model based on LightGBM. This outcome is not surprising, since LightGBM is considered to be state-of-the-art on tabular data, as stated by [43], [44]. However, the ensembles of ANNs still yield adequate performance and come close to matching the benchmark model's performance.

Attempting to model the ratio of future non-shippable bookings based on a sequence of previous cancellation and transfer rates with a GRU does not appear to provide additional valuable information beyond what is already captured by the ANN ensembles. However, it is important to note that this could be influenced by the limitations of the stacked ANN model, which is responsible for generating the final prediction. Although, Figure A.6 shows convergence to similar MAPE for the GRU network as the MAPE achieved by the LightGBM model with only the time until departure as a feature, visualised in Figure 4.1. This could indicate that previous transfer and cancellations rates are not informative for future non-shippable booking rates. The GRU network could benefit from more training as well.

The overall performance is influenced by factors such as hyperparameter settings, the number of instances included in the ANN ensemble, and the selection of specific ANN instances to be part of the ensemble. It is possible that certain instances may have converged to suboptimal local minimums of the loss function. Given the stochastic nature of the training process, the performance of neural networks can vary. This variability is evident in Appendix A, and it underscores the potential for an ensemble of multiple instances to offer increased reliability. It is worth noting that the ensemble technique can also be applied to the GRU network and the stacked ANN model, which may have an impact on the overall performance. However, one downside to consider is the increased training time associated with ensembles.

In summary, the neural networks tested in this thesis are neither impressive from a forecast perspective nor computationally efficient in comparison with the benchmark model.

It is likely that an overbooking strategy based on predictions from machine learning models should not solely focus on the final point prediction of $S_j^{(t)}$. Incorporating the estimated variance of $S_j^{(t)}$ could be equally important due to the stochastic nature of the problem. The errors appear to peak between 20 hours and 1 hour before departure, suggesting that the predicted variance of $S_j^{(t)}$ should ideally reflect the discrepancy between the predictions and actual outcomes. Figure 4.4 visualises the predicted variance for sailing X, indicating a peak around 20 hours before departure.

Examining the normalised predicted variance for all sailings in the first validation set, as shown in Figure 4.7, reveals significant differences among the sailings up until 1 hour before departure. This observation suggests the importance of adapting the error estimates based on the attributes of the active bookings at time t for sailing j. Figures 4.8 and 4.9 demonstrate promising results, indicating that the predicted variance reflects the error of the predictions from the ANN ensembles on average. The findings further suggest greater volatility in the estimates for higher predicted variances, which could be influenced by a smaller number of samples in the corresponding predefined intervals and larger intervals.

The proposed simulation approach for estimating the variance of a Poisson-binomial random variable with uncertain probabilities shows empirically promising results. This approach addresses the challenge of calculating the properties of uncertain probabilities, which can be difficult in practice [14]. In this thesis, the mathematical aspects of the Poisson-binomial distribution lack rigour, and interested readers could be referred to [14] as a starting point for further exploration.

Both the linear regression approach and the stacked ANN approach did not yield improvements compared to the normal approximation with the ANN ensemble and the benchmark model based on LightGBM. However, it is worth considering that other methods of stacked generalisation may be beneficial when applied with different algorithms and features in the data set. The initial hypothesis was that the skewness of the Poisson-binomial distribution could be a significant factor, but the results do not support this notion.

However, Figure 4.7 reveals interesting patterns. Many samples for long durations before departure demonstrate negative skewness, while positive skewness is observed for numerous samples close to the departure time. This observation aligns with intuition, as many active bookings made well in advance of the departure date often undergo transfers or cancellations, while a significant number of active bookings close to the departure time tend to appear on the day of departure. These insights indicate that $S_j^{(t)}$ could be considered to be skewed at certain times before departure t. Although, it might not be a significant factor to consider. During long durations before departure, the average number of active bookings is relatively low, and close

to the departure time, the average number of true non-shippable bookings is limited. Consequently, in absolute terms, it could be reasonable that the skewness of $S_j^{(t)}$ does not significantly impact the results of an overbooking strategy. However, as the departure time approaches and the ferries are potentially close to full capacity, the predicted skewness of $S_j^{(t)}$ could become relevant from a risk perspective of short shipment.

The computational complexity associated with calculating the Poisson-binomial distribution could make it impractical for real-world applications. Hence, the normal approximation, as described in Section 2.1, becomes a viable alternative. Hong [8] suggests that this approximation method may introduce inaccuracies when dealing with highly skewed data. However, in the specific context of this thesis, skewness does not appear to be a significant factor for improving the forecast performance. The reader could be referred to [8] and the Refined normal approximation, as mentioned in Section 2.1, for further investigation.

5.2 Interpretation of the entity embedding

At the collaborating company, an overbooking model serves as a decision-making tool for accepting or declining booking requests at a given time t before departure for a specific sailing j. Therefore, the process is not necessarily fully automated. With this in mind, according to the author of this thesis, it can be just as important to describe how the predictive model interprets the attributes of the active bookings, in addition to the resulting prediction or recommendation. Therefore, discussing the entity embedding of the categorical features becomes relevant. This technique can be compared to clustering techniques, and the predictive models based on ANNs with entity embedding provide the clustering result "for free".

Research studies such as [35], [36], show how entity embedding of categorical features can unveil intrinsic properties. The results presented in Section 4.5 provide indications of no-show patterns during different seasons, weekly patterns, and patterns related to sailing times. Furthermore, the check-in codes demonstrate distinct separation in the two-dimensional space of the first and second principal components. Although it may be challenging to see a pattern among the customers in Figure 4.10, the distribution of PCA1 and PCA2 appears to follow a normal distribution. This observation could depend on the random initialisation of weights in the entity embedding from a standard normal distribution, $N(0, 1)$.

5.3 Further studies

The author of this thesis recognises certain limitations and potential areas for improvement in the presented work. Firstly, the data gathering procedure employed can be considered naive, and the convergence plots in Appendix A demonstrate rapid convergence within a single epoch, indicating that a smaller data set might be useful. Additionally, the exploration of feature engineering was limited. Specifically, in the

context of active bookings at a given time before departure, only the latest version was considered. Bookings can be viewed as a sequence of states, and incorporating information from previous versions could potentially boost the performance. One possible enhancement could be to treat the feature values as changes themselves, rather than solely focusing on the current state of the booking.

Furthermore, it is important to recognise that this thesis only considers a subset of relevant factors for an effective overbooking strategy. One potential factor that has not been explored is the demand and how it varies over time leading up to the departure. To incorporate this aspect, the analysis would likely benefit from including not only accepted bookings but also declined bookings. However, it should be noted that obtaining data on declined bookings may be challenging since it might not be recorded.

In terms of machine learning algorithms, Bayesian Neural Networks (BNNs) [13] could offer interesting possibilities for this problem. BNNs are based on probabilistic layers instead of weights, and uncertainty estimates are a natural aspect of the modelling technique [13]. However, it is important to note that training BNNs can be computationally expensive [39]. Although, the findings of this thesis suggest that only a subset of the considered features are informative, which implies that a BNN approach could be a viable option. Further exploration of Bayesian techniques, such as BNNs, could provide more robust uncertainty estimates and enhance the decision-making process in overbooking strategies.

6
Conclusion

The freight ferry industry is characterised by a high uncertainty regarding the presence of vehicles on the departure day. This thesis has explored the use of artificial neural networks (ANNs) to predict bookings that are transferred, cancelled, or fail to appear on the departure day, referred to as non-shippable bookings. The total number of active non-shippable bookings at time t for sailing j was assumed to follow a Poisson-binomial distributed random variable, denoted as $S_j^{(t)}$. The proposed approach, based on an ensemble of ANNs, demonstrated promising results in modelling both the mean and variance of $S_j^{(t)}$. However, the mean of $S_j^{(t)}$ was better approximated by Gradient Boosting Trees (LightGBM) based on the data used in this thesis. On average, the predicted variance of $S_j^{(t)}$ reflects the mean squared error between the observed values $s_j^{(t)}$ and the predicted mean $\hat{\mu}_{S_j^{(t)}}$ with an ANN ensemble, which can provide valuable risk insights in practical applications. Additionally, dealing with the Poisson-binomial probability distribution can be challenging, particularly when the probabilities of specific events occurring are unknown. The simulation-based approach proposed in this thesis to estimate the variance of a Poisson-binomial distributed random variable with unknown probabilities shows promising empirical results. Lastly, the entity embedding technique applied to categorical features unveiled indications of patterns related to seasonal variations, weekly patterns, and sailing hours.

Bibliography

[1] L. R. Weatherford and S. E. Bodily, "A Taxonomy and Research Overview of Perishable-Asset Revenue Management: Yield Management, Overbooking, and Pricing," *Operations Research*, vol. 40, no. 5, 1992, ISSN: 0030-364X. DOI: 10.1287/opre.40.5.831.

[2] J. I. McGill and G. J. Van Ryzin, "Revenue management: research overview and prospects," *Transportation Science*, vol. 33, no. 2, 1999, ISSN: 00411655. DOI: 10.1287/trsc.33.2.233.

[3] J. Feldman, P. Kouvelis, and Y. Qiu, "A New Class of Revenue Management Problems with Overbooking and No-Shows: Shoring up Trust between Shippers and Carriers in Maritime Container Shipping," *SSRN Electronic Journal*, 2022. DOI: 10.2139/ssrn.4124363.

[4] Eurostat, *Maritime freight and vessels statistics*, Nov. 2022.

[5] H. Zhao, Q. Meng, and Y. Wang, "Probability estimation model for the cancellation of container slot booking in long-haul transports of intercontinental liner shipping services," *Transportation Research Part C: Emerging Technologies*, vol. 119, 2020, ISSN: 0968090X. DOI: 10.1016/j.trc.2020.102731.

[6] H. Zhao, Q. Meng, and Y. Wang, "Exploratory data analysis for the cancellation of slot booking in intercontinental container liner shipping: A case study of Asia to US West Coast Service," *Transportation Research Part C: Emerging Technologies*, vol. 106, 2019, ISSN: 0968090X. DOI: 10.1016/j.trc.2019.07.009.

[7] Y. Wang and Q. Meng, "Integrated method for forecasting container slot booking in intercontinental liner shipping service," *Flexible Services and Manufacturing Journal*, vol. 31, no. 3, pp. 653–674, Sep. 2019, ISSN: 19366590. DOI: 10.1007/S10696-018-9324-Z/TABLES/7. [Online]. Available: https://link.springer.com/article/10.1007/s10696-018-9324-z.

[8] Y. Hong, "On computing the distribution function for the Poisson binomial distribution," *Computational Statistics and Data Analysis*, vol. 59, no. 1, 2013, ISSN: 01679473. DOI: 10.1016/j.csda.2012.10.006.

[9] Y. Yang, "On the number of successes in independent trials," *Statistica Sinica*, vol. 3, pp. 295–312, 1993. [Online]. Available: https://www3.stat.sinica.edu.tw/statistica/oldpdf/A3n23.pdf.

[10] A. Y. Volkova, "A Refinement of the Central Limit Theorem for Sums of Independent Random Indicators," *Theory of Probability & Its Applications*, vol. 40, no. 4, 1996, ISSN: 0040-585X. DOI: 10.1137/1140093.

[11] G. R. Grimmet and D. R. Stirzaker, *Probability and Random Processes*, 4th ed. Oxford: Oxford University Press, 2020, pp. 207–207.

[12] P. McCullagh and J. Kolassa, "Cumulants," *Scholarpedia*, vol. 4, no. 3, p. 4699, 2009, ISSN: 1941-6016. DOI: 10.4249/SCHOLARPEDIA.4699.

[13] C. Bishop, *Pattern Recognition and Machine Learning*. Springer, Jan. 2006. [Online]. Available: https://www.microsoft.com/en-us/research/publication/pattern-recognition-machine-learning/.

[14] I. Diakonikolas, D. M. Kane, and A. Stewart, "Properly Learning Poisson Binomial Distributions in Almost Polynomial Time," 2015.

[15] B. Mehlig, *Machine Learning with Neural Networks*. 2021. DOI: 10.1017/9781108860604.

[16] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals, and Systems*, vol. 2, no. 4, 1989, ISSN: 09324194. DOI: 10.1007/BF02551274.

[17] S. Becker, Y. Zhang, and A. A. Lee, "Geometry of Energy Landscapes and the Optimizability of Deep Neural Networks," *Physical Review Letters*, vol. 124, no. 10, 2020, ISSN: 10797114. DOI: 10.1103/PhysRevLett.124.108301.

[18] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The Bulletin of Mathematical Biophysics*, vol. 5, no. 4, 1943, ISSN: 00074985. DOI: 10.1007/BF02478259.

[19] D. Silver, A. Huang, C. J. Maddison, *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, 2016, ISSN: 14764687. DOI: 10.1038/nature16961.

[20] *Introducing ChatGPT*. [Online]. Available: https://openai.com/blog/chatgpt.

[21] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, vol. 65, no. 6, 1958, ISSN: 0033295X. DOI: 10.1037/h0042519.

[22] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On calibration of modern neural networks," in *34th International Conference on Machine Learning, ICML 2017*, vol. 3, 2017.

[23] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 2015.

[24] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, 2011, ISSN: 15324435.

[25] T. Tieleman and G. Hinton, "Lecture 6.5 - rmsprop," *COURSERA: Neural Networks for Machine Learning*, 2012.

[26] P. Zhou, J. Feng, C. Ma, C. Xiong, S. Hoi, and E. Weinan, "Towards theoretically understanding why SGD generalizes better than ADAM in deep learning," in *Advances in Neural Information Processing Systems*, vol. 2020-December, 2020.

[27] N. S. Keskar and Socher Richard, "Improving Generalization Performance by Switching from Adam to SGD," 2017.

[28] A. C. Wilson, R. Roelofs, M. Stern, N. Srebro, and B. Recht, "The marginal value of adaptive gradient methods in machine learning," in *Advances in Neural Information Processing Systems*, vol. 2017-December, 2017.

[29] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org.

[30] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Journal of Machine Learning Research*, vol. 15, 2011.

[31] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *7th International Conference on Learning Representations, ICLR 2019*, 2019.

[32] K. He, X. Zhang, S. Ren, and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," in *2015 IEEE International Conference on Computer Vision (ICCV)*, IEEE, Dec. 2015, pp. 1026–1034, ISBN: 978-1-4673-8391-2. DOI: 10.1109/ICCV.2015.123.

[33] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Journal of Machine Learning Research*, vol. 9, 2010.

[34] J. T. Hancock and T. M. Khoshgoftaar, "Survey on categorical data for neural networks," *Journal of Big Data*, vol. 7, no. 1, 2020, ISSN: 21961115. DOI: 10.1186/s40537-020-00305-w.

[35] A. De Brébisson, Simon, A. Auvolat, P. Vincent, and Y. Bengio, "Artificial neural networks applied to taxi destination prediction," in *CEUR Workshop Proceedings*, vol. 1526, 2015.

[36] C. Guo and F. Berkhahn, "Entity embeddings of categorical variables," *arXiv preprint arXiv:1604.06737*, 2016.

[37] *Embedding Layer Size Rule - Part 1 (2019) / Advanced (Part 1 v3) - fast.ai Course Forums*. [Online]. Available: https://forums.fast.ai/t/embedding-layer-size-rule/50691.

[38] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, 2014, ISSN: 15337928.

[39] B. Lakshminarayanan, A. Pritzel, and C. Blundell, "Simple and scalable predictive uncertainty estimation using deep ensembles," in *Advances in Neural Information Processing Systems*, vol. 2017-December, 2017.

[40] J. L. Elman, "Distributed Representations, Simple Recurrent Networks, And Grammatical Structure," *Machine Learning*, vol. 7, no. 2, 1991, ISSN: 15730565. DOI: 10.1023/A:1022699029236.

[41] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, 1997, ISSN: 08997667. DOI: 10.1162/neco.1997.9.8.1735.

[42] K. Cho, B. Van Merriënboer, C. Gulcehre, *et al.*, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," in *EMNLP 2014 - 2014 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*, 2014. DOI: 10.3115/v1/d14-1179.

[43] G. Ke, Q. Meng, T. Finley, *et al.*, "LightGBM: A Highly Efficient Gradient Boosting Decision Tree," in *Advances in Neural Information Processing Systems*, I Guyon, U. V. Luxburg, S Bengio, *et al.*, Eds., vol. 30, Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf.

[44] V. Borisov, T. Leemann, K. Sessler, J. Haug, M. Pawelczyk, and G. Kasneci, "Deep Neural Networks and Tabular Data: A Survey," *IEEE Transactions on Neural Networks and Learning Systems*, 2022, ISSN: 21622388. DOI: 10.1109/TNNLS.2022.3229161.

[45] *Features — LightGBM 3.3.5.99 documentation*. [Online]. Available: https://lightgbm.readthedocs.io/en/latest/Features.html.

[46] A. Paszke, S. Gross, F. Massa, *et al.*, "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[47] *scikit-learn: machine learning in Python — scikit-learn 1.2.2 documentation*. [Online]. Available: https://scikit-learn.org/stable/index.html.

[48] M. Lin, H. C. Lucas, and G. Shmueli, "Too big to fail: Large samples and the p-value problem," *Information Systems Research*, vol. 24, no. 4, 2013, ISSN: 15265536. DOI: 10.1287/isre.2013.0480.

[49] L. Li, K. Jamieson, A. Rostamizadeh, *et al.*, "Massively Parallell Hyperparameter Tuning," 2018. [Online]. Available: https://openreview.net/forum?id=S1Y7OOlRZ.

[50] *torch.nn.init — PyTorch 2.0 documentation*. [Online]. Available: https://pytorch.org/docs/stable/nn.init.html.

[51] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *30th International Conference on Machine Learning, ICML 2013*, 2013.

A

Monitoring of the training procedure

In this appendix, some results from the monitoring of the training process of the artificial neural networks are presented. Training loss, validation loss and MAPE on the first or second validation set were recorded.

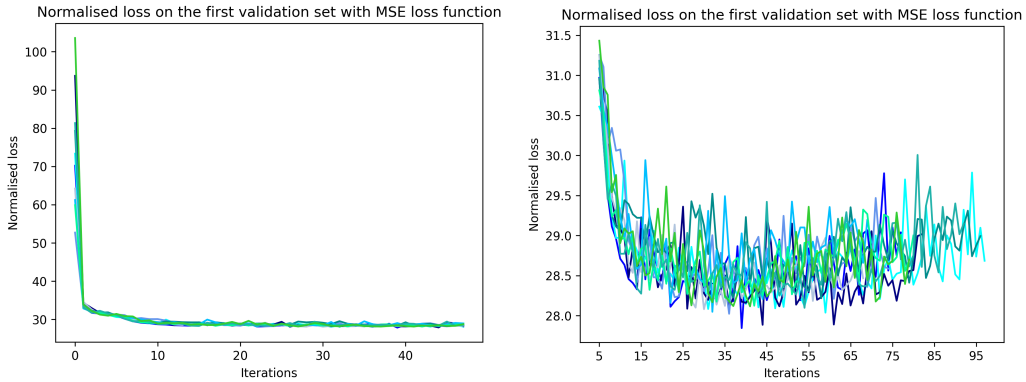

Figure A.1: Normalised validation loss (first) plotted against iterations of gradient descent. The loss is normalised with the number of batches in the validation set. Ten iterations correspond to one epoch, therefore, one iteration approximately corresponds to 120.000 gradient descent steps. The MSE was used as the loss function.

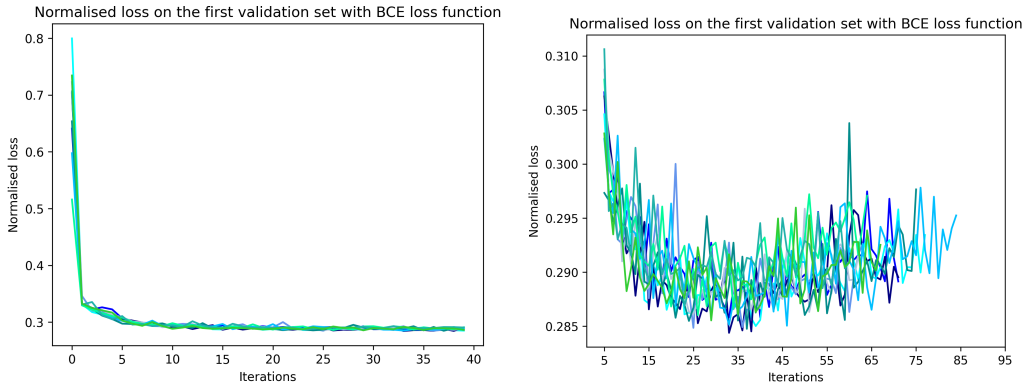

Figure A.2: Normalised validation loss (first) plotted against iterations of gradient descent. The loss is normalised with the number of batches in the validation set. Ten iterations correspond to one epoch, therefore, one iteration approximately corresponds to 120.000 gradient descent steps. The BCE was used as the loss function.

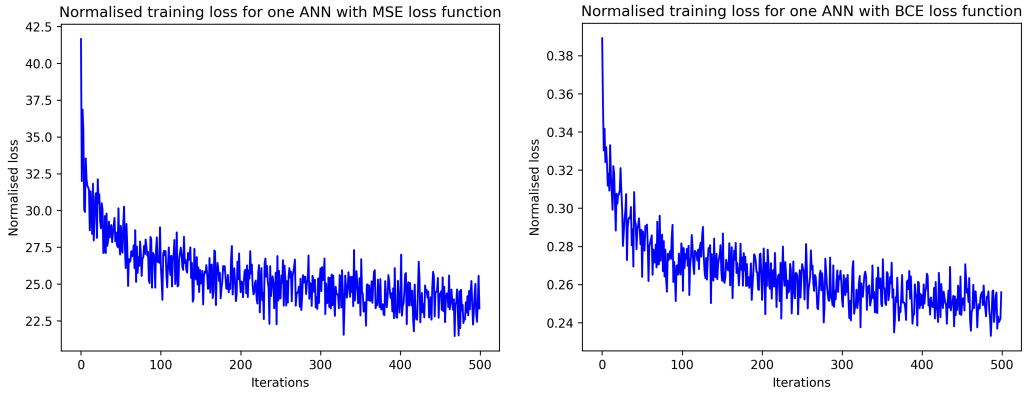

Figure A.3: Normalised training loss plotted against iterations of gradient descent. The loss is normalised with the number of batches in the validation set. One iteration corresponds to 100 gradient descent steps. In the left frame was MSE used as the loss function, while the BCE loss function was used in the right frame.

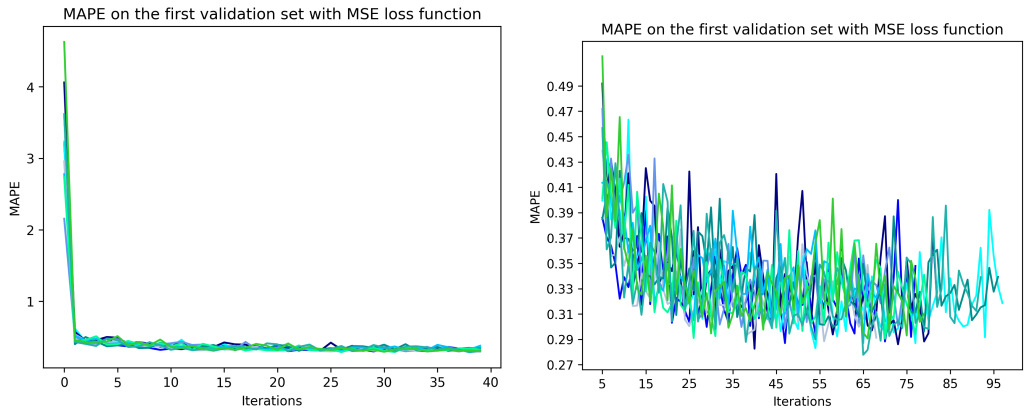

Figure A.4: MAPE plotted against iterations of gradient descent. Ten iterations correspond to one epoch, therefore, one iteration approximately corresponds to 120.000 gradient descent steps. The MSE was used as the loss function.

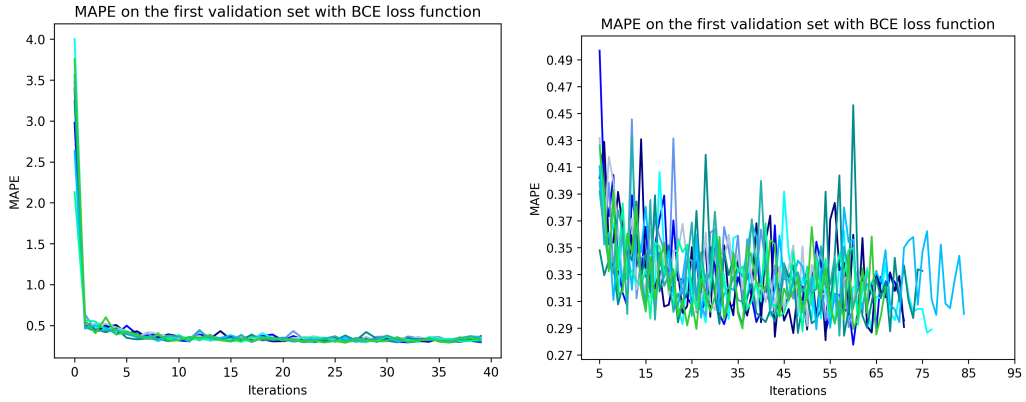

Figure A.5: MAPE plotted against iterations of gradient descent. Ten iterations correspond to one epoch, therefore, one iteration approximately corresponds to 120.000 gradient descent steps. The BCE was used as the loss function.

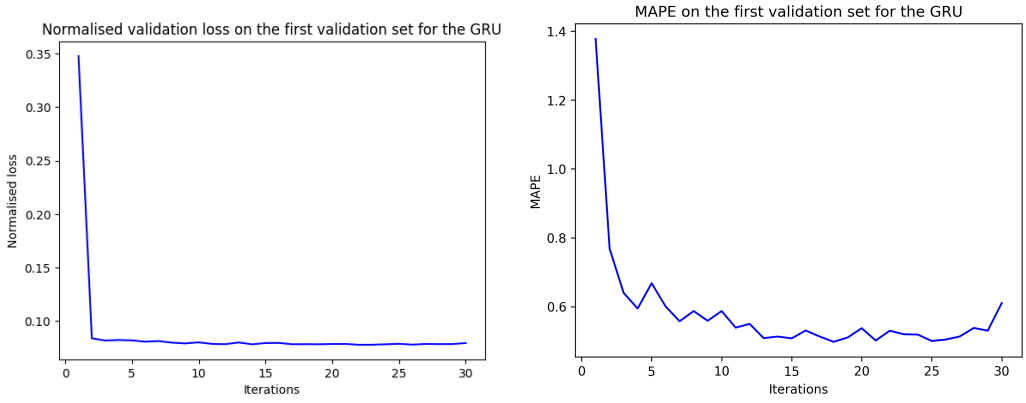

Figure A.6: Normalised validation loss and MAPE on the first validation set plotted against iterations of gradient descent for the GRU. The loss is normalised with the number of batches in the validation set. Ten iterations correspond to one epoch. The sharp decrease in the validation loss depends on the infrequent recording of the loss.

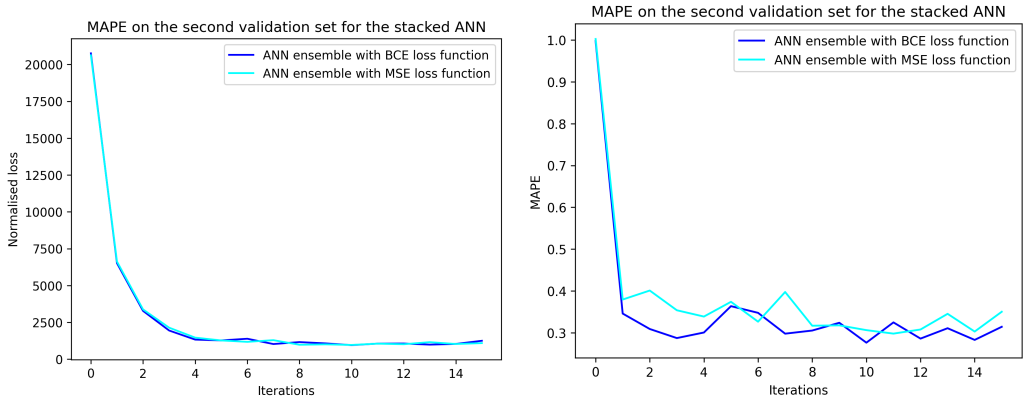

Figure A.7: Normalised validation loss and MAPE on the second validation set plotted against iterations of gradient descent for the stacked ANN model. The loss is normalised with the number of batches in the validation set. One iteration corresponds to five epochs. The stacked ANN model combines both the output from the ANN ensemble and the GRU. The ANN ensemble was trained with both BCE and MSE loss functions, both options are visualised in the figure.

B

Predicted summary statistics using an ANN ensemble with MSE loss function

In this appendix, the results of the predicted mean, variance and skewness of $S_j^{(t)}$ for sailing X in the first validation set, using the ANN ensemble with the MSE loss function, are visualised. The normalised predicted mean, variance and skewness for all sailings in the first validation set, using the ANN ensemble with the MSE loss function, are also visualised.

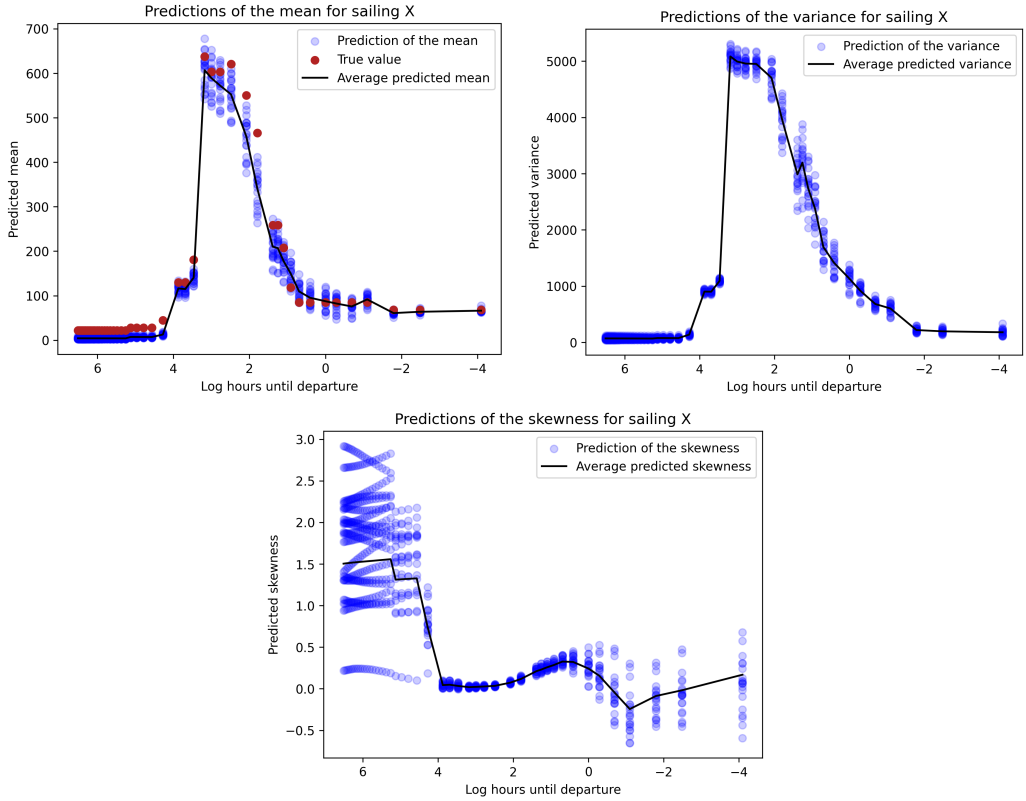

Figure B.1: Point predictions of the mean, variance and skewness of $S_j^{(t)}$ for one of the sailings in the first validation set with the ANNs in the ensemble. The MSE loss function was used, and the probabilities were treated as known values.

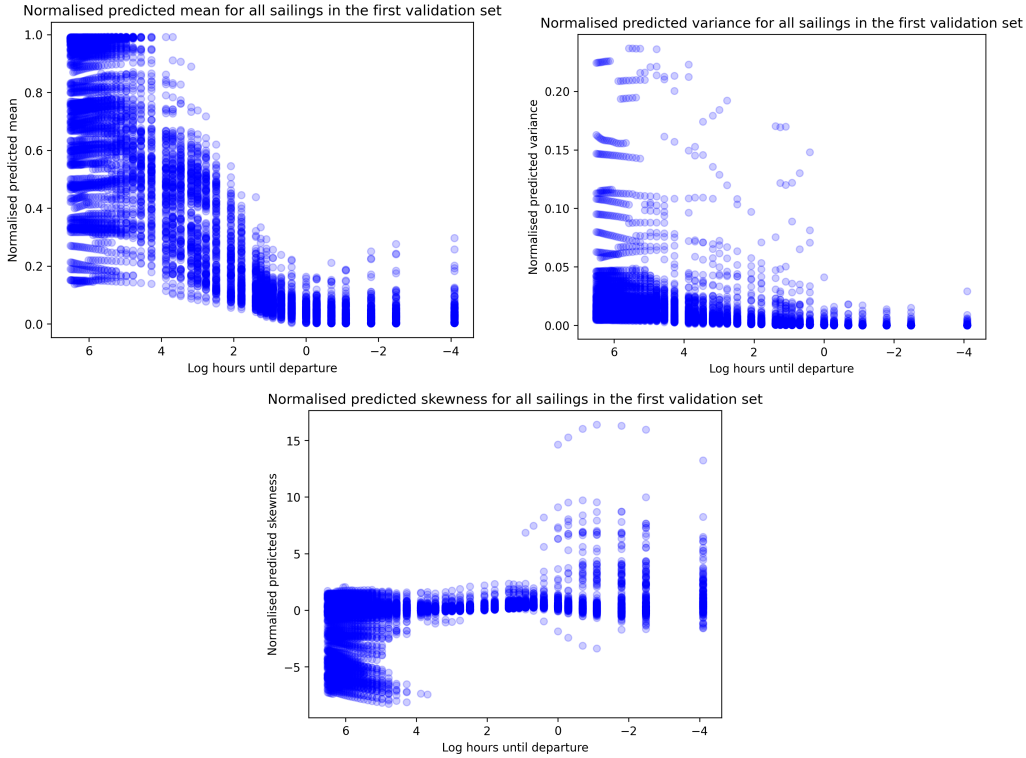

Figure B.2: Normalised predicted mean, variance and skewness for all sailings in the first validation set. The normalisation was performed based on the total length booked at time t on sailing j. Hence, the variable of interest is $S_j^{(t)}/|l_j^{(t)}|$, where $|l_j^{(t)}|$ is the booked length at time t on sailing j. The MSE loss function was used.

DEPARTMENT OF SOME SUBJECT OR TECHNOLOGY
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY