Using Artificial Neural Networks in the Calculation of Mortgage Prepayment Risk

Author: Robben Riksen
Supervisor: dr. P.J.C Spreij

Examination date: August 22, 2017
Daily supervisor: dr. B. Wemmenhove
dr. P.W. den Iseger

Korteweg-de Vries Institute for Mathematics
ABN AMRO Bank N.V.
Abstract

A mortgage loan comes with the option to prepay (part of) the full amount of the loan before the end of the contract. This is called mortgage prepayment, and poses a risk to the bank issuing the mortgage due to the loss of future interest payments. This thesis reviews some general properties of artificial neural networks, which are then applied to predict prepayment probabilities on mortgage loans. The Universal Approximation Theorem for neural networks with continuous activation functions will be treated extensively. Suggestions for a prepayment model based on neural networks are made.
Contents

Introduction 5

List of Abbreviations 10

1. Mortgage Prepayment 11
   1.1. An introduction to mortgages 11
   1.2. The current prepayment model 13
      1.2.1. Explanatory variables 14
      1.2.2. Model description 16
   1.3. The underlying short-rate model 17
      1.3.1. The one-factor Hull-White model 18
      1.3.2. Fitting the model to the initial forward curve 20
      1.3.3. The Nelson-Siegel-Svensson model 20

2. Artificial Neural Networks 23
   2.1. Neural networks, an introduction 23
   2.2. Universality of neural networks 26
   2.3. Optimization methods 40
      2.3.1. Cost functions 40
      2.3.2. Gradient descent 41
      2.3.3. The Adam optimizer 42
   2.4. Backpropagation 43
      2.4.1. The backpropagation algorithm 43
      2.4.2. Learning slowdown 45
   2.5. Activation and output functions 45
      2.5.1. Output functions 45
      2.5.2. Activation functions 46
   2.6. Weight initialization 49
      2.6.1. ReLU initialization 49
   2.7. Reduction of overfitting 52

3. Simulations 55
   3.1. Method 55
      3.1.1. Data collection 55
      3.1.2. Model selection 58
      3.1.3. Pricing the prepayment option 61
   3.2. Results 62
4. Conclusion 68

5. Discussion and Further Research 69
   5.1. Advantages and disadvantages of artificial neural networks . . . . . . . . . 69
   5.2. Further research . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 71
   5.3. Review of the process . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 71

Popular summary 73
Bibliography 75
Appendices 78

A. Additional Theorems and Definitions 79
   A.1. Theorems for Chapter 1 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 79
   A.2. Theorems for Chapter 2 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 83

B. The Stone-Weierstass Theorem 86
C. Historical Proof of the Universal Approximation Theorem 90
Introduction

A mortgage is a special kind of loan issued by a bank or another mortgagor. It is used to fund the purchase of real estate by the client, with the purchased property as collateral. Since the property purchased with the mortgage loan can be sold if the client fails to make his contractual payments, the conditions of a mortgage loan are often better than those of an ordinary client loan. The term mortgage is derived from the word ‘mortgage’, death pledge, in old French. So called because the deal ends, or dies, either when the debt is paid or when payment fails. While in modern French the word is replaced by ‘hypothèque’, the word mortgage was introduced in the English language in the Middle Ages and has been used since.

When buying a house in the Netherlands, it is very common to take a mortgage on that property. Over 81% of Dutch home owners and half of the Dutch households have a mortgage\(^1\). With an average mortgage loan of around €267,000 and a summed total of almost 81 billion euros for new mortgages in 2016\(^2\), the dutch mortgage market was worth a total of 665 billion euros in terms of outstanding loans in September 2016. At the end of 2016 ABN AMRO had a market share of 22% in this market\(^3\), making it very important to quantify the risks that come with writing out a mortgage loan. A big part of this risk for the mortgagor lies in the possibility of default of the client, as became painfully clear during the 2008 financial crisis. However, there is another factor that poses a risk. Clients have the possibility to pay back (a part of) the loan earlier than discussed in the contract. Since the mortgagor makes money of the loan by receiving interest, this will decrease the profitability of the mortgage. Especially when taking into account that clients are more likely to do this when the interest rates prevailing in the market are lower than the contractual interest rate on the mortgage. Furthermore, these so-called prepayments cause a funding gap in the sense that the prepaid money has to be invested earlier than expected, often leading to less profitable investments when market interest rates are low.

To calculate the prepayment risk and hedge against it, it is necessary to be able to estimate the prepayment ratio for certain groups of clients well. After arranging the clients in groups that show more or less similar behaviour, this is the fraction of clients in that group that prepay their mortgage. A good estimation of the prepayment ratio will reduce the costs that arise by over-hedging prepayment risk and is also required when reporting risks to the market authority (AFM). On top of this, the prepayment risk is used to calculate the fair price of penalties for the client that come with certain prepayment options.

\(^1\)http://statline.cbs.nl
\(^2\)Kadaster
\(^3\)ABN AMRO Annual Report 2016
There are many reasons clients choose to make prepayments on their mortgage loan. When the interest rate for new mortgage contracts is lower than the contractual mortgage of a client, the client has a financial incentive to pay off his mortgage and get a new one, or to sell his house and buy a new house with a mortgage contract with a lower interest rate. But not many people will be tempted to move or renegotiate their mortgage every time the interest rates are low. In economical terms, when not moving they are making ‘irrational’ decision. Another example of economically irrational client behaviour is moving when the interest rates are high, e.g. when in need of more space because of the birth of a child.

This thesis was written during an internship at ABN AMRO. The current model to estimate the prepayment ratio at ABN AMRO is based on a multinomial regression, taking into account many variables that can be of influence in prepayment decisions. This master thesis aims to find an alternative method to calculate the prepayment ratio using artificial neural networks. Neural networks proved their worth in fields including image, pattern and speech recognition, classification problems, fraud detection and many more. Loosely modelled after the brain, they are assumed to be good at tasks humans are better at than computers. Among other things, we therefore hope that neural networks are better at capturing the ‘irrational’ behaviour of clients than traditional methods.

**Artificial neural networks**

Before giving a brief introduction into feedforward artificial neural networks, we will shortly summarize the history and recent developments in this field.

**History and current developments**

Surprisingly, artificial neural networks first appeared even before the age of computers. In 1943 McCulloch and Pitts [24] introduced a model inspired by the working of the brain that took binary inputs and produced binary outputs and regulated the firing of a neuron using a step function. By their ability to represent the logical AND and OR functions, it was possible to implement Boolean functions. In 1958 Rosenblatt [30] published an article about the *perceptron*, a simple model where input is processed in one neuron with a step function and a learning algorithm that could be used in classification problems. Due to its simplicity it is still often used as an introduction to the theory of artificial neural networks. However, it is incapable of implementing the logical *exclusive or* (XOR) function, hence only able to solve linearly separable classification problems. This was seen as a huge drawback. Another drawback was that neural networks with more neurons needed a lot of computational power to train well, which was simply not available at the time. Interest and funding in this field of research faded for a while, until the publication of a famous article in 1986 by Rumelhart, Hinton and Williams [33] popularizing the backpropagation algorithm (Section 2.4) which had already been introduced in the seventies, but had remained largely unnoticed. Backpropagation solved the exclusive or problem, allowing the fast training of bigger and deeper (i.e. more layers
of neurons) neural networks. This made the interest in artificial neural networks rise again and a lot of progress was made. Nowadays, the increase of computational power and the abundance of available data to train big neural networks, has led to many promising results in many fields of application. Artificial neural networks are booming. The developments in this field of research are rapid. A lot of articles introducing state-of-the-art techniques are still in preprint, but the techniques might have already become industry standard. It is an applied research area, so many methods are based solely on empirical results. When proofs do appear, they are often applied in situations where not all of the assumptions are satisfied. The argument ‘I don’t know why, but it works’ always wins. This of course to great frustration of a mathematician. In this thesis we will therefore spend some pages to show the theoretical capabilities of artificial neural networks.

What are artificial neural networks?

As mentioned before, artificial neural networks are more or less modelled after the (human) brain. The nodes in a neural network are therefore often called neurons. Similar to the neurons in the brain, artificial neurons generate output based on the input they receive. An artificial neural network consists of three types of neurons: input neurons, hidden neurons and output neurons. The most common type of artificial neural network is the feedforward network. Information enters the network through the input neurons. The input neurons send the input to the first layer of hidden neurons. These hidden neurons receive the inputs and apply a so-called activation function to an affine transformation of the inputs. That activation is then sent to the neurons in the next layer, where this process repeats itself. When the signal reaches the output layer, the output neurons generate the output of the network by applying an output function to an affine transformation of the received activations. To conclude, a neural network is a function taking input and generating output. The objective is to find the right function for a certain problem. It can be ‘trained’, optimized to generate the right output given its input, by choosing the parameters for the affine transformations made in every neuron in the network. A schematic representation of an artificial neural network is given in Figure 1.

To train a network properly, large amounts of data are required. A neural network without training does not know what function it has to approximate. Therefore, the training data has to consist of many input points with their required (although possibly noisy) outputs. Training data is fed into the network, generating output. The output of the network is then compared to the desired output (the targets), to compute the error of the network. To change the parameters in a way that the error decreases, often a variant of the gradient descent algorithm is applied. All in all, this means that the exact form of the neural network is not pre-programmed, but it ‘learns’ how to deal with the data. After a lot of training, one hopes that the parameters at which the training algorithm arrives make the output of the neural network approximate the desired function well enough.
Outline of the thesis

This thesis aims to explore the application of artificial neural networks in the estimation of prepayment risk on mortgages. It will both look at the theory of neural networks in general and at the application to this specific problem.

The first chapter will deal with mortgage prepayment. It will describe the specifics of a Dutch mortgage in Section 1.1, and explain the model that is currently used at ABN AMRO to predict mortgage prepayments in Section 1.2. Because the interest rate on a mortgage is derived from swap rates, Section 1.3 is dedicated to describing the underlying short-rate model used in the simulations.

Chapter 2 will cover the general theory of artificial neural networks, starting with the basic definitions in Section 2.1. In Section 2.2, the theoretical value of neural networks is shown by two denseness results, often called the Universal Approximation Theorem. This section therefore has a more mathematical character than the other sections. First uniform denseness on compacta is shown in the set of continuous functions. Then, we show denseness in the set of measurable functions with respect to the Ky Fan metric. In the remainder of Chapter 2 several options to optimize the performance and trainability of a network are discussed.

In Chapter 3, simulations to estimate prepayment probabilities with a neural network are described. We will describe how data was collected and how a final model was selected. Then in Section 3.2, the results of the simulations are given.

In Chapter 4, a conclusion about the performance and usability of the selected model will be drawn from the simulation results. In Chapter 5, a discussion of the results and the general usage of artificial neural networks will follow and some recommendations for further model improvements will be made. To conclude this chapter, we will briefly comment on the process of this research.
Then, a popular summary of the subject is given. This summary should be accessible to first year bachelor students in mathematics. In Appendix A, relevant theorems and definitions will be stated. For most of the theorems appearing here, the proof is omitted and the reader is referred to a source where the proof appears. In Appendix B, an elementary proof of the Stone-Weierstrass Theorem is given. The choice to state and prove this theorem in a separate appendix is made because the theorem and proof were new to the author. The selected proof is surprising and uses only elementary properties of compact sets and continuous functions. Finally, Appendix C contains a historically relevant proof of the Universal Approximation Theorem for a specific type of activation function.
List of Abbreviations

RIPU Remaining Interest Period Ultimo 14
HPI House Price Index (base year 2010) 14
LtV Loan to Value 15
ATM At-The-Money 18
NSS Nelson-Siegel-Svensson 20
ECB European Central Bank 20
MSE Mean Squared Error 40
CE Cross-Entropy 40
SGD Stochastic Gradient Descent 41
ReLU Rectified Linear Unit 46
LReLU Leaky Rectified Linear Unit 47
PReLU Parametric Rectified Linear Unit 47
ELU Exponential Linear Unit 47
FRA Forward Rate Agreement 80
1. Mortgage Prepayment

Before we can apply artificial neural networks to calculate the mortgage prepayment rate, we first have to explain some properties of a Dutch mortgage. To compare the model using artificial neural networks with the current model, we will also describe a simplified version of the prepayment model as it is currently used at ABN AMRO. In the first section of this chapter, a short introduction to the general structure of Dutch mortgages is given. In Section 1.2, the current mortgage prepayment model as used at ABN AMRO is explained. Section 1.3 is dedicated to explaining the Hull-White short-rate model, which is necessary in our simulations to generate mortgage interest rates.

1.1. An introduction to mortgages

As defined in the Introduction, a mortgage is a loan issued by a bank or another mortgagor that is used to fund the purchase of real estate by the client, with the purchased property as collateral. The amount that the client borrows is called the principal of the loan. Whenever the borrower fails to pay off the loan, the mortgagor can sell the property in an attempt to reduce its losses and offset the loan (this is called foreclosure). Because of the size of the collateral, the size of a mortgage loan can be relatively high and interest rates for mortgages are relatively low compared to other types of loans. Usually the client pays monthly interest on the mortgage loan until the maturity (end of contract), at which the principal has to be fully repaid. The maturity of a mortgage is often 30 years. Different repayment schedules are discussed below. The interest rate on the mortgage can be variable or fixed. In the Netherlands in 2016, 52% of mortgages have a fixed interest period between five and ten years, 20.5% of ten years or more, 12.5% between one and five years and 15% up to one year or no fixed interest period. Whenever the interest rate is fixed, it will remain the same during the agreed interest period. If the end of the interest period is before the maturity date of the contract, the bank and the client may agree on another interest period in which the interest rate will be fixed again. When a fixed interest period ends and the client and the bank agree on new conditions for a next interest period, we will view this as a total repayment of the loan and settlement of a new loan. Therefore, the maturity of a mortgage will be taken equal to the fixed interest period from now on. This is valid because the new conditions may differ significantly from the conditions on the old loan. Below we will summarize some of the other characteristics of a mortgage loan.

[1] www.dnb.nl
Amortization schedule

When choosing a mortgage, a client can choose different amortization schedules, i.e. ways to repay the principal of the mortgage loan. The three amortization schedules that are currently offered at ABN AMRO are bullet (aflossingsvrij), level paying (fixed-rate mortgage or annuïteitenhypotheek) and linear. In case of the bullet amortization schedule, the client will only pay interest during the running time of the loan and repay the full amount at the end of the contract. In a level paying loan, the client pays a fixed amount each month until the total amount of the loan has been repaid at the end of the contract. This implies that the monthly payments at the beginning of the contract mainly consist of interest and only by a small part of repayment, while towards the end of the contract the repayment size increases while the interest payments decrease. In the linear schedule the client pays off a fixed amount of the loan each payment date. The amortization schedules are schematically depicted in Figure 1.1 below.

Figure 1.1.: A schematic representation of the amortization schedules\(^2\). From left to right: bullet, level and linear.

Loan parts

A mortgage is often split into several loan parts that can have different conditions. A situation in which this typically arises is when a client sells his house and decides to buy a new house. The client now has to pay off his current mortgage and has to get a new one on the new house. The bank offers the option to get a mortgage on the new house with the same conditions and outstanding principal as the old mortgage (meeneemoptie). However, if the new house is more expensive, the client will have a loan part with the conditions and of the size of the old mortgage and has to get another loan part with new conditions for the remaining amount to fund the house. Different loan parts can therefore have different interest rates and different amortization schedules. In the following models we will therefore only look at loan parts, thus viewing every loan part as a full mortgage loan.

Prepayment options

Besides the different amortization schedules to repay the mortgage, the client has several other options to repay (a part) of the loan before the contractual end date. This is called

---

\(^2\) Adaptation of images from
https://www.abnamro.nl/nl/prive/hypotheken/hypotheekvormen/index.html
prepayment of the loan. Prepayment can be done in several ways. Some mortgages come with a reconsider option. Clients with these mortgages have the option to renegotiate the terms of their mortgage without extra costs during the reconsider period, usually the last two years of the fixed interest period. The prepayment categories as stated in the ABN AMRO prepayment documentation [18] are

1. Refinancing: Full repayment of the loan without the collateral being sold, or an early interest rate reset not taking place during an interest reconsider period.

2. Relocation: Full repayment of the loan in general caused by the sale of the collateral.

3. Reconsider (exercise of rentebedenktijdoptie): Full repayment of the loan without the collateral being sold, or an early interest rate reset during an interest reconsider period.

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

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

6. No event: Only contractual repayments.

As discussed in the Introduction, prepayment poses a risk for the bank. Therefore, if the prepayment is bigger than a certain percentage of the loan, called the franchise \( f \), in some situations a penalty has to be payed. The franchise usually is 10% of the original principal, although there are some mortgagors that maintain a franchise of 20%. Note that in the categories above the franchise is set to 10.5%, to avoid the allocation of curtailment low to the event curtailment high due to data inaccuracies. Prepayments of the type relocation and reconsider are penalty free. The penalty therefore applies to the prepayment classes curtailment high and refinancing. A situation where there is no penalty for any of the prepayment classes is when the mortgage interest rates in the market are higher than the fixed mortgage rate in the contract.

The calculation of the penalty assigned to each of the options is beyond the scope of this thesis, but can be found in the ABN AMRO penalty model description [27].

1.2. The current prepayment model

Because the model used at ABN AMRO is confidential and because the main goal of this thesis is the application of artificial neural networks, we will look at a simplified version of the ABN AMRO multinomial prepayment model. For each type of prepayment, the model is used to predict the probability that a loan is prepayed in that way. When grouping similar loans together, these probabilities can then be interpreted as the fraction of the loans in a certain prepayment category.
In Subsection 1.2.1, the relevant explanatory variables that influence the prepayment probabilities will be introduced. In Subsection 1.2.2, a concise description of the multinomial regression model is given.

1.2.1. Explanatory variables

The traditional prepayment model is a multinomial regression model. It aims to predict the prepayment fraction of a group of people with similar characteristics for each of the prepayment classes described in the last section. The model uses the following characteristics of a mortgage loan as explanatory variables:

- Remaining interest period ultimo (RIPU): The number of months until the next interest reset date.
- Interest incentive: The incentive for prepayment caused by the mortgage interest rates.
- Loan age: The number of months from the start of the interest period up to the current month.
- Penalty proxy: As a proxy for the penalty term, the interest incentive multiplied by the remaining interest period is taken.
- HPI ratio: The current House Price Index (HPI) divided by the HPI at the interest period start date.
- Amortization type: This can be bullet, level or linear.
- Interest term: The length of the interest period in months.
- Brand indicator: Can be ‘main brand’ or ‘non-main brand’.
- Seasonality: The current month.
- Personnel indicator: Indicates whether the client is ABN AMRO staff or not.

We will now shortly discuss each of the above mentioned explanatory variables. We will describe how it is used in the multinomial model and describe the possible influence on a prepayment event to justify its use as explanatory variable.

**Remaining interest period ultimo**

This is an important predictive variable since clients can only use the reconsider option during the last two years of the interest period. Furthermore, clients tend to relocate towards the end of the interest period ([18]). As stated above, the RIPU is used in the model as the number of months until the interest end date.
Interest incentive
This is one of the most important prepayment incentives. The interest incentive is
defined as the contractual interest rate minus the mortgage rate in the market for a
mortgage with the same fixed interest period. The bigger the interest incentive, the
more interest costs for the client are reduced by prepayment. A big interest incentive
indicates low interest rates, also on saving accounts. If the interest rate received on a
savings account is low, it is very beneficial to make small prepayments (curtailments)
instead of putting money on the savings account.

Loan age
The older the loan, the higher the probability of refinance, since people will not do this
shortly after they signed the mortgage contract. Also there appears to be a peak in
relocations around 5 years after the start of the loan. On the other hand clients tend to
curtail if the loan age is small, since then the benefit is greatest.

Penalty proxy
Because the calculation of the penalty for certain prepayment events is quite involved,
we will use a simplified penalty proxy. Since the costs for the bank are greatest when the
remaining interest period is large and the interest incentive is big, the interest incentive
multiplied by the remaining interest period is used as a proxy. The penalty proxy is
therefore an indication of how beneficial it is for the client to make a prepayment. Thus,
a higher penalty proxy will increase the prepayment probabilities. This effect holds until
the penalty becomes so large that the prepayment advantage is unmade.

HPI ratio
If the value of the house of a client increases, the loan-to-value ratio (LtV) of the mort-
gage decreases. This makes it more attractive to refinance, since better terms might be
agreed upon in a new contract. Also the probability of relocation will increase, because
profit will be made when the property is sold. Because the the mortgages are bundled
into buckets of loans with similar characteristics, it is impossible to recover the LtV
ratio for an individual loan. Instead, the HPI ratio is used as a proxy. A high HPI ratio
(above 1) is an indication that the value of the house went up, a low HPI ratio indicates
a decrease in value. Curtailment appears to be more likely when the HPI ratio is well
above 1, because those clients have more own funds to curtail with. On the other hand,
if the HPI ratio is slightly below 1 clients will also curtail more to reduce the mortgage
burden.

Amortization type
Different amortization types cause different curtailment behaviour. Clients with a level
amortization scheme curtail most in the beginning of the contract. There the benefit of
curtailment is greatest, since the contractual repayments are small. Linear mortgages curtail least, because their contractual repayments start right after the signing of the contract.

**Interest term**

Longer interest terms will cause a higher curtailment rate, because the benefits of curtailment in the beginning of the contract are bigger for a long maturity.

**Brand indicator**

If the mortgage is taken directly from ABN AMRO, then the brand indicator is ‘main brand’. If the contract is signed through an intermediary it is a ‘non-main brand’ mortgage. Main brand mortgages tend to curtail and refinance less than non-main brand mortgages.

**Seasonality**

The month has a big effect on the prepayment rates and distinct seasonal effects are observed. For example, refinancing and curtailment go up near the end of the year when people review their mortgage and savings balance and curtailment can possibly cause a tax benefit. Furthermore, there is a strong seasonality on the house market. Significantly more houses are sold during the summer months. This will increase the relocation ratio in these months. Possible changes in regulations may drive up the relocation probability near the end of the year as well.

**Personnel indicator**

A member of the ABN AMRO staff is more likely be aware of the reconsider option and therefore the reconsider rate is higher for staff.

### 1.2.2. Model description

Consider a mortgage loan with a known vector $X$ of explanatory variables as described above in Subsection 1.2.1. We want to calculate the probability that a certain prepayment is made. Let $j$ denote the prepayment event as numbered in Section 1.1 (i.e. $j = 1$ means refinancing etc.). We will write $\pi_j$ for the probability of a prepayment of category $j$. Traditionally, using no event as pivot in the multinomial regression, the probabilities on events are estimated as

$$
\pi_j = \frac{e^{X \cdot \beta_j}}{1 + \sum_{k=1}^{5} e^{X \cdot \beta_k}}, \quad \text{for } j = 1, \ldots, 5,
$$

and

$$
\pi_6 = \frac{1}{1 + \sum_{k=1}^{5} e^{X \cdot \beta_k}},
$$
where \( \beta_j \) is a vector of parameters specified in [18], usually different for each \( j \). If an explanatory variable is not relevant for a prepayment category, the corresponding parameter entry in \( \beta_j \) will be zero.

However, to handle the reconsider prepayment category correctly, we need to adapt the method slightly, since a reconsider prepayment can only be done in the reconsider period (conversely, refinancing can only take place outside the reconsider period, for otherwise it would be a reconsider event). We therefore define

\[
\eta_j = e^{X_j \cdot \beta_j}, \quad \text{for } j = 2, 4, 5,
\]

and

\[
\eta_1 = \begin{cases} 
  e^{X_1 \cdot \beta_1} & \text{if not in reconsider period} \\
  0 & \text{if in reconsider period}
\end{cases},
\]

\[
\eta_3 = \begin{cases} 
  0 & \text{if not in reconsider period} \\
  e^{X_3 \cdot \beta_3} & \text{if in reconsider period}
\end{cases}.
\]

The prepayment probabilities are then specified as

\[
\pi_j = \frac{\eta_j}{1 + \sum_{k=1}^{5} \eta_k}, \quad \text{for } j = 1, \ldots, 5,
\]

and

\[
\pi_6 = \frac{1}{1 + \sum_{k=1}^{5} \eta_k}.
\]

For a more detailed description of the ABN AMRO multinomial prepayment model and how the parameter vectors \( \beta_j \) are estimated, see [18].

To evaluate the prepayment probabilities for a portfolio on different time instants in the future, a prediction of the mortgage rate in the market is needed to calculate the interest incentive. This is discussed in Section 1.3. For the HPI on future time points, the HPI forecast of ABN AMRO is used. The construction of this forecast is beyond the scope of this thesis.

1.3. The underlying short-rate model

The market mortgage rate \( M_T(t) \) at time \( t \) for a mortgage with maturity \( T \) is given by the swap rate \( S_T \) of a swap with maturity \( T \) plus a certain spread \( \delta_T \),

\[
M_T(t) = S_T(t) + \delta_T. \tag{1.1}
\]

Since the goal of this thesis is to describe the prepayment model and investigate the use of artificial neural networks in calculating the prepayment risk, we choose to use a simple model to calculate the short rates. Furthermore, it is convenient to choose a
model that has analytic expressions for the swap rates. This saves us the effort of having
to construct a trinomial tree (e.g. Appendix F in [3]) to make numerical calculations of
the swap rates feasible. For this reason we will use the one-factor Hull-White model to
calculate short rate scenarios.

During this section, we assume to be on a probability space \((\Omega, \mathcal{F}, \mathbb{P})\) with filtration
\(\mathcal{F} = \{\mathcal{F}_t\}_{t \geq 0}\) satisfying the usual conditions. We also assume there exists an equiva-
lent martingale measure \(\mathbb{Q} \sim \mathbb{P}\). We will often refer to Appendix A, where necessary
definitions and theorems are stated.

### 1.3.1. The one-factor Hull-White model

This subsection will give a recap of the one-factor Hull-White model [15], in order to
derive an analytic expression for the swap rates \(S_T(t)\) in terms of the initial instantaneous
forward curve, the zero-coupon bond prices at \(t = 0\) and the short rate. The short rate
dynamics under \(\mathbb{Q}\) are

\[
dr(t) = (b(t) + \beta(t)r(t)) \, dt + \sigma(t) \, dW^*_t,
\]

where \(W^*_t\) is a \(\mathbb{Q}\)-Brownian motion, \(\beta(t), \sigma(t)\) are chosen to obtain the desired volatility
structure and \(b(t)\) is chosen to match the current initial forward curve. In this thesis,
we will use a slightly simplified version of this model, where we take \(\beta\) and \(\sigma\) constant:

\[
dr(t) = (b(t) + \beta r(t)) \, dt + \sigma \, dW^*_t. \tag{1.2}
\]

To find an analytic expression for the ATM swap rates (see Appendix A.1, equation
(A.5)), we need to find the \(T\)-bond prices. Recall the definition of a model with an affine
term-structure from [9].

**Definition 1.1.** A short-rate model provides an affine term-structure if there are smooth
functions \(A\) and \(B\) such that

\[
P(t, T) = e^{-A(t,T)-B(t,T)r(t)}. \tag{1.3}
\]

Note that in this definition, for fixed \(T\), \(P\) is a function of \(t \in [0, T]\) and \(r(t)\). To
find a characterization of short-rate models with an affine term structure, we apply the
Feynman-Kac formula (Theorem A.1) to the right hand side of equation (1.3). This
leads to Corollary A.2 in Appendix A.1.

Now clearly the one-factor Hull-White model satisfies the form of equation (A.8) with
\(a(t) = \sigma\), \(a(t) = 0\) and \(\beta(t) = \beta\). Using these to solve the differential equations (A.9)
and (A.10) will show that this model has an affine term-structure. For \(B(t, T)\) we see
that equation (A.10) becomes

\[
\partial_t B(t, T) = -\beta B(t, T) - 1, \quad B(T, T) = 0,
\]

which we can solve straightforwardly to find

\[
B(t, T) = \frac{1}{\beta} \left( e^{\beta(T-t)} - 1 \right).
\]
Integrating equation (A.9) for $A$ yields

$$A(t, T) = -\frac{\sigma^2}{2} \int_t^T B^2(s, T) \, ds + \int_t^T b(s) B(s, T) \, ds. \quad (1.4)$$

To solve this equation further, we first need to find a more explicit expression for $b(t)$. For the first part we follow the line of reasoning of [9]. Using the affine term structure and the definition of the instantaneous forward rate (equation (A.4)), we can write

$$f(0, T) = \partial_T A(0, T) + \partial_T B(0, T) r(0).$$

Using Leibniz integral rule and the fact that $\partial_T B(t, T) = -\partial_t B(t, T)$ we write

$$f(0, T) = \frac{\sigma^2}{2} \int_0^T \partial_s B^2(s, T) \, ds + \int_0^T b(s) \partial_T B(s, T) \, ds + e^{\beta T} r(0).$$

If we now define the function

$$\phi(T) := \int_0^T b(s) e^{\beta(T-s)} \, ds + e^{\beta T},$$

we find that $\phi(T) = f(0, T) + \frac{\sigma^2}{2 \beta^2} (e^{\beta T} - 1)^2$ and, again by Leibniz integral rule, that $\partial_T \phi(T) = \beta \phi(T) + b(T)$. Solving this for the function $b$ gives us

$$b(t) = \partial_t \phi(t) - \beta \phi(t)$$

$$= \partial_t \left( f(0, t) + \frac{\sigma^2}{2 \beta^2} (e^{\beta t} - 1)^2 \right) - \beta f(0, t) - \frac{\sigma^2}{2 \beta} (e^{\beta t} - 1)^2$$

$$= \partial_t f(0, t) - \beta f(0, t) - \frac{\sigma^2}{2 \beta} \left( 1 - e^{2\beta t} \right). \quad (1.5)$$

Having found this expression for $b$ we can now integrate in equation (1.4) to find $A(t, T)$. Using that $B(T, T) = 0$, partial integration and (A.4), we get after a lot of calculus

$$A(t, T) = -\frac{\sigma^2}{2} \int_t^T B^2(s, T) \, ds + \int_t^T b(s) B(s, T) \, ds$$

$$= \int_t^T f(0, s) \, ds - f(0, t) B(t, T) - \frac{\sigma^2}{4 \beta} \left( 1 - e^{2\beta t} \right) B(t, T)^2$$

$$= -\log \left( \frac{P(0,T)}{P(0,t)} \right) - f(0, t) B(t, T) - \frac{\sigma^2}{4 \beta} \left( 1 - e^{2\beta t} \right) B(t, T)^2.$$

Hence, given the initial instantaneous forward curve and the zero-coupon bond prices at $t = 0$, we can by virtue of the affine term structure calculate the zero-coupon bond prices at all $t \leq T$ and therefore the swap rates.
1.3.2. Fitting the model to the initial forward curve

In Subsection 1.3.1, we derived an expression for the function $b$ to match the current term structure (equation (1.5)). However, the expression involved a derivative of the initial forward curve which can be inconvenient and increases the effect of a possible observation error. To get rid of the derivative, we will introduce a different representation of the one-factor Hull-White model, which is described in [3]. Solving the stochastic differential equation for the short rate and substituting the found expression for $b$ gives

$$
r(t) = r(0)e^{\beta t} + \int_0^t e^{\beta(t-s)}b(s)\,ds + \int_0^t \sigma e^{\beta(t-s)}\,dW^*_s
= f(0, t) + \frac{\sigma^2}{2\beta^2} \left(1 - e^{\beta t}\right)^2 + \int_0^t \sigma e^{\beta(t-s)}\,dW^*_s
= \alpha(t) + \int_0^t \sigma e^{\beta(t-s)}\,dW^*_s,
$$

where we defined the function

$$\alpha(t) := f(0, t) + \frac{\sigma^2}{2\beta^2} \left(1 - e^{\beta t}\right)^2.$$

Hence it is clear that $r(t)$ is normally distributed with mean $\alpha(t)$ and variance equal to

$$\int_0^t \sigma^2 e^{2\beta(t-s)}\,ds = \frac{\sigma^2}{2\beta^2} \left(e^{2\beta t} - 1\right).$$

To find a convenient way to simulate these short rate paths, we will define a process $x$ by the dynamics

$$dx(t) = \beta x(t)\,dt + \sigma\,dW^*_t, \quad x(0) = 0.$$

This implies that

$$x(t) = \sigma \int_0^t e^{\beta(t-s)}\,dW^*_s,$$

so

$$r(t) = \alpha(t) + x(t).$$

The short rate now consists of a deterministic part $\alpha(t)$ reflecting the initial term-structure and a stochastic process $x(t)$, independent of the initial market conditions, which we can simulate.

1.3.3. The Nelson-Siegel-Svensson model

Using the results from subsections 1.3.1 and 1.3.2, we can generate swap rates and short rate sample paths, given the initial instantaneous forward rate curve. Therefore, we need a model that fits the observed market data well. This is a field of research by itself, but since we only need one model, we will confine ourselves to a short description of the
Nelson-Siegel-Svensson model (NSS). This model is currently used by the ECB and the parameters for the current market structure are quoted daily on their website. The precursor of the NSS model was introduced by Nelson and Siegel ([25]) and gave a parametrization of the spot rate in four parameters

\[ R(0, T) = \beta_0 + \beta_1 \frac{1 - e^{-T/\tau}}{T/\tau} + \beta_2 \left( \frac{1 - e^{-T/\tau}}{T/\tau} - e^{-T/\tau} \right). \]

They argue that \( \beta_0 \) captures the long term structure, \( \beta_1 \) is the contribution of the short term effects and \( \beta_2 \) is the medium-term component that can cause the typical hump-shape in the curve. Svensson later added a term and two extra parameters to improve the fit with a second hump-shape ([36]). The spot rate model becomes

\[ R(0, T) = \beta_0 + \beta_1 \frac{1 - e^{-T/\tau_1}}{T/\tau_1} + \beta_2 \left( \frac{1 - e^{-T/\tau_1}}{T/\tau_1} - e^{-T/\tau_1} \right) + \beta_3 \left( \frac{1 - e^{-T/\tau_2}}{T/\tau_2} - e^{-T/\tau_2} \right). \]

Recalling the definitions of the spot rate and instantaneous forward rate, equations (A.3) and (A.4) respectively, we see that

\[ f(0, T) = R(0, T) + T \frac{\partial}{\partial T} R(0, T), \]

so in the NSS model we find a forward rate of

\[ f(0, T) = \beta_0 + \beta_1 e^{-T/\tau_1} + \beta_2 \frac{T}{\tau_1} e^{-T/\tau_1} + \beta_3 \frac{T}{\tau_2} e^{-T/\tau_2}. \]

In Figure 1.2 the different components of the forward rate are plotted separately to illustrate their effect on the forward rate curve.

Using the ECB parameters for the Nelson-Siegel-Svensson model, we now have all the ingredients to generate swap rates with initial conditions matching the current market structure.
Figure 1.2.: Two examples of the Nelson-Siegel-Svensson forward rate (uninterrupted line) and all components plotted separately. The parameters used in the upper plot are from the ECB for 1 December 2016, the parameters in the lower plot are from 11 December 2007.
2. Artificial Neural Networks

In this chapter we will introduce the concept of artificial neural networks. We will look at feedforward networks and prove that even the functions produced by neural networks with only one hidden layer are uniformly dense on compacta in $C(\mathbb{R})$. This shows that we can indeed use artificial neural networks to approximate any continuous function. We will also show density in the set of measurable functions with respect to the Ky Fan metric. After this, we will discuss the choice of different cost and activation functions for the network to increase performance and introduce the backpropagation algorithm to update the network after each training cycle. The last section of this chapter will describe methods to reduce overfitting.

Throughout this chapter we will denote by $L$ the number of layers of a network and by $d_l$ the number of neurons in layer $l$. This means that $d_0$ denotes the dimension of the input and $d_L$ the dimension of the output (and targets).

2.1. Neural networks, an introduction

In this section we will use a modified version of the notation used by Bishop [2] and Nielsen [26]. A neural network is used to approximate an often unknown and complicated function $f: \mathbb{R}^{d_0} \rightarrow \mathbb{R}^{d_L}, (x_1, \ldots, x_{d_0}) \mapsto (t_1, \ldots, t_{d_L})$. Thus, the goal is to create a neural network that takes an input vector $x = (x_1, \ldots, x_{d_0})$ and produces an output vector $y = (y_1, \ldots, y_{d_L})$ that is a good approximation of the target vector $t = (t_1, \ldots, t_{d_L})$. A neural network consists of layers of neurons (also called nodes or units) that each receive a certain input and generate an output. A neuron takes an affine transformation of the input it receives, the result is called the weighted input of the neuron. The output of the neuron is called the activation and is obtained by applying an activation function to the weighted input. The activation function can be chosen depending on the application of the network. Historically, so-called sigmoidal functions (non-decreasing functions from $\mathbb{R}$ to $[0, 1]$ approaching 0 in the negative and 1 in the positive limit, Definition C.3) were the most commonly used activation functions. An example of a sigmoidal function is the logit function. For this reason, we denote the activation function in this section by $\sigma$. Nowadays, sigmoidal functions are often replaced by other activation functions. For a discussion about different types of activation functions, see Section 2.5.

A neural network consists of layers: one input layer, a number of hidden layers and one output layer. Figure 2.1 gives a schematic representation of a neural network. The hidden layers are called like this because they only interact with neurons inside the network. In this sense they are ‘hidden’ from the outside of the network. Consider a network with $L$ layers. By convention we do not count the input layer, so there are
$L - 1$ hidden layers and 1 output layer. The input layer is built up out of so called input neurons. These are special in the sense that they take only one input and do not have an activation function. Input neuron $i$ takes input $x_i$ and sends the same $x_i$ as output to all neurons in the first hidden layer. Therefore the number of input neurons is equal to the dimension $d_0$ of the input. Let’s say the first hidden layer consists of $d_1$ neurons. Each hidden neuron in this layer receives input from all the input neurons and takes an affine transformation of these to produce its weighted input. The weighted input of hidden neuron $j$ in hidden layer 1 is therefore

$$z^{(1)}_j = \sum_{i=1}^{d_0} w^{(1)}_{ji} x_i + b^{(1)}_j,$$

where constants $w^{(1)}_{ji}$ are called the weights of hidden neuron $j$ for input $i$ and the constant $b^{(1)}_j$ is called the bias of hidden neuron $j$. Note that there are $d_1$ nodes in the first hidden layer, so there are $d_1 \times d_0$ weights and $d_1$ biases necessary to compute all weighted inputs of this layer. We can represent the $d_1$-dimensional vector of the weighted inputs of the first hidden layer as a matrix vector multiplication

$$z^{(1)} = W^{(1)} x + b^{(1)},$$

where $W^{(1)}$ is the weight matrix with entries $w^{(1)}_{ji}$ and $x$ and $b^{(1)}$ are $d_0$- and $d_1$-dimensional vectors of the inputs and biases respectively. The activation of a hidden neuron is defined as the activation function $\sigma$ applied to the weighted input of the neuron. For neuron $j$ in the first hidden layer, the activation therefore is

$$a^{(1)}_j = \sigma(z^{(1)}_j).$$
After the calculation of the activations in the first hidden layer, the process is repeated as the information flows to the next hidden layer. The output of one layer is the input for the next layer. In general we therefore find that the activation of neuron \( j \) in layer \( l \) is given by

\[
a_j^{(l)} = \sigma(z_j^{(l)}) = \sigma \left( \sum_{i=1}^{d_{l-1}} w_{ji}^{(l)} a_i^{(l-1)} + b_j^{(l)} \right).
\]

Equivalently, the activation vector of layer \( l \) is given by

\[
a^{(l)} = \sigma(W^{(l)}a^{(l-1)} + b^{(l)}),
\]

where the activation function \( \sigma \) is applied component-wise. After passing through the hidden layers, the signal goes to the output layer, consisting of \( d_L \) output neurons. They again take a weighted sum and add a bias. So for \( j = 1, \ldots, d_L \) the output neurons weighted input is

\[
z_j^{(L)} = \sum_{i=1}^{d_{L-1}} w_{ji}^{(L)} a_i^{(L-1)} + b_j^{(L)},
\]

where \( w_{ji}^{(L)} \) is the weight of output neuron \( j \) for hidden neuron \( i \) and \( b_j^{(L)} \) is the bias for output neuron \( j \). The output vector of the network is the activation function of the output layer, denoted by \( h \), applied to its weighted input. The output of the neural network is therefore given by the vector with components

\[y_j = h(z_j^{(L)}), \quad j = 1, \ldots, d_L,
\]

or in vector notation

\[
y = h(z^{(L)}) = h \left( W^{(L)}a^{(L-1)} + b^{(L)} \right).
\]

Note that the output \( y \) of the network is a function of the input \( x \) and the weights and biases of the network. The choice of the output activation function \( h \) depends on the application, see Section 2.5, but for regression problems often a linear output activation function \( h(x) = x \) is chosen.

We thus see that a neural network takes affine transformations of the inputs, applies an activation function and repeats this process for each layer. Because the information moves in one direction – from the first to the last layer – networks of this kind are called feedforward networks. With a random choice of the weights and biases, the probability of getting an output vector \( y \) close to the target vector \( t \) is of course minuscule. Therefore, after initializing the weights and biases (for a discussion about different weight initialization methods see Section 2.6) they need to be updated and improved to get more accurate output vectors. This is done by choosing a suitable cost function to measure the approximation error (Subsection 2.3.1) and minimizing the cost function by updating the weights according to an optimization method (Section 2.3).
To train and judge the performance of a network, a lot of data is needed. The dataset should consist of many $d_0$-dimensional input vectors and corresponding $d_L$-dimensional target values. This is why training these types of networks is called *supervised learning*: a set of input values is needed for which we already know the (maybe noisy) output values of the function we want to approximate. Often the available data is split into a training set, a validation set and a test set to reduce the risk of overfitting (see Section 2.7). The training data is used to optimize the weights and biases in the network. The validation data is then used to choose suitable meta-parameters, i.e. parameters other than the weights and biases, like the number of layers. Finally the performance of the network is tested on the test data. There is a lot to say about the choices to be made when designing a neural network. But before we delve into neural networks any deeper, we want to know what functions artificial neural networks are able to approximate.

### 2.2. Universality of neural networks

In this section we will show that even shallow neural networks, i.e. networks with only one hidden layer, with linear output neurons are able to approximate any Borel measurable function with arbitrary precision if sufficiently many hidden neurons are available. Historically, this result, often called the ‘Universal Approximation Theorem’, was shown in 1989 by Hornik, Stinchcombe and White [14], see Appendix C, and separately by Cybenko [7]. This first version of the Universal Approximation Theorem only works for neural networks with sigmoidal activation functions, see Definition C.3.

The theorem by Hornik et al. sufficed for a long time, when neural networks were mainly used with sigmoidal activation functions, like the standard logit function. However, currently the most used activation function is the *Rectified Linear Unit* ($f(x) = \max\{0, x\}$, see Subsection 2.5.2). This activation function is unbounded and therefore does not satisfy the definition of a sigmoidal function. Luckily, it is possible to prove a Universal Approximation result for activation functions that are continuous but not polynomial.

The proof was originally done by Leshno et al. [20], but we will loosely follow the line of the proof in a review article by Pinkus [28].

In both articles, the proof only applies to one-layer neural networks and shows denseness in the topology of uniform convergence on compacta in $C(\mathbb{R}^n)$, the class of continuous functions from $\mathbb{R}^n$ to $\mathbb{R}$. However, using some lemmas from [14], we can expand the result to multi-layer networks and to denseness in $\mathcal{M}^n$, the set of measurable functions from $\mathbb{R}^n$ to $\mathbb{R}$. The main results of this section are Corollary 2.11, stating that one-layer neural networks are uniformly dense on compacta in $C(\mathbb{R}^n)$ and Corollary 2.24, stating a similar denseness result for multi-layer multi-output neural networks. It also shows denseness with respect to a different metric in $\mathcal{M}^{n,m}$, the class of measurable functions from $\mathbb{R}^n$ to $\mathbb{R}^m$.

In this section we will use notation that is conventional in literature, where $\sigma$ denotes the (possibly not sigmoidal) activation function. We define the class of functions that can be the output of a one-layer neural network with input $x \in \mathbb{R}^n$ and one linear output neuron.
Definition 2.1. For any Borel measurable function $\sigma : \mathbb{R} \to \mathbb{R}$ and $n \in \mathbb{N}$ we define the class of functions

$$\Sigma^n(\sigma) := \text{span} \{ g : \mathbb{R}^n \to \mathbb{R} \mid g(x) = \sigma(w \cdot x + b); w \in \mathbb{R}^n, b \in \mathbb{R} \}$$

$$= \left\{ f : \mathbb{R}^n \to \mathbb{R} \mid f(x) = \sum_{i=1}^d b_i \sigma(A_i(x)); b_i \in \mathbb{R}, A_i \in A^n, d \in \mathbb{N} \right\},$$

where $A^n$ is defined as the set of all affine functions from $\mathbb{R}^n$ to $\mathbb{R}$.

A polynomial is the finite sum of monomials. By the degree of a polynomial we mean the maximum degree of its monomials, which is the sum of the powers of the variables in the monomial. Before we start with proving the main result of this section, we will shortly mention a converse statement. Suppose that the activation function $\sigma$ is a polynomial, say of degree $d$. Then clearly $\Sigma^n(\sigma)$ is the space of polynomials of degree $d$. This directly implies that $\Sigma^n(\sigma)$ cannot be dense in $C(\mathbb{R}^n)$. Thus, if we want to find activation functions for which $\Sigma^n(\sigma)$ is dense in $C(\mathbb{R}^n)$, we do not have to look at polynomials.

Next, we will specify what we mean by uniform denseness on compacta.

Definition 2.2. A subset $S \subset C(\mathbb{R}^n)$ is called uniformly dense on compacta in $C(\mathbb{R}^n)$ if for every compact subset $K \subset \mathbb{R}^n$, for every $\varepsilon > 0$ and for every $f \in C(\mathbb{R}^n)$ there exists a $g \in S$ such that $\sup_{x \in K} |f(x) - g(x)| < \varepsilon$.

The main theorem of this section is based on an older result concerning ridge functions. These are defined as follows.

Definition 2.3. A function $f : \mathbb{R}^n \to \mathbb{R}$ is called a ridge function if it is of the form $f(x) = g(a \cdot x)$, for some function $g : \mathbb{R} \to \mathbb{R}$ and $a \in \mathbb{R}^n \setminus \{0\}$.

In fact, note that the activation of a neuron in a single layer neural network, $\sigma(w \cdot x + b)$, is a ridge function for every activation function $\sigma$, every $w$ and $b$.

The necessary theorem, when making our way to Corollary 2.11, is a result by Vostrecov and Kreines [38] from 1961. However, we follow the line of the proof to the theorem from [21] and [29].

Recall that homogeneous polynomials are polynomials for which each term has the same degree. Let us denote the linear space of homogeneous polynomials of $n$ variables of degree $k$ by $H^n_k$. Note that the dimension of $H^n_k$ is $\binom{n+k-1}{k}$.

Theorem 2.4. Let $A$ be a subset of $\mathbb{R}^n$. The span of all ridge functions

$$\mathcal{R}(A) := \text{span}\{ f : \mathbb{R}^n \to \mathbb{R} \mid f(x) = g(a \cdot x); g \in C(\mathbb{R}), a \in A \}$$

is uniformly dense on compacta in $C(\mathbb{R}^n)$ if and only if the only homogeneous polynomial that vanishes on $A$ is the zero polynomial.
Proof. Given any direction \( a \in A \), we note that \( \mathcal{R}(A) \) also contains \( g(\lambda a \cdot x) \) for all \( \lambda \in \mathbb{R} \), since \( \mathcal{R}(A) \) allows all functions \( g \) in \( C(\mathbb{R}) \), so we can absorb \( \lambda \) in the function. This means that we are done by proving the result for sets \( A \subseteq S^{n-1} \), the unit sphere in \( \mathbb{R}^n \).

\textcircled{1} \implies \textcircled{'}: We will prove this by contradiction. Assume that \( \mathcal{R}(A) \) is dense in \( C(\mathbb{R}^n) \) and there exists a nontrivial homogeneous polynomial \( p \) of some degree \( k \) that vanishes on \( A \).

By assumption, \( p \) is of the form
\[
p(x) = \sum_{j=1}^{d} c_j x_1^{m_{j,1}} \cdots x_n^{m_{j,n}} = \sum_{j=1}^{d} c_j x_{m_j},
\]
for some \( d \in \mathbb{N} \), coefficients \( c_j \) and \( m_{j,i} \in \mathbb{Z}_+ \) such that \( \sum_{i=1}^{n} m_{j,i} = k \) for all \( j \). We also denoted by \( m_j \) the vector \((m_{j,1}, \ldots, m_{j,n}) \in \mathbb{Z}^n_+ \) and introduced the useful notation \( x^m = x_1^{m_1} \cdots x_n^{m_n} \).

Now pick any \( \phi \in C^\infty_c(\mathbb{R}^n) \), the set of infinitely dimensional functions from \( \mathbb{R}^n \) to \( \mathbb{R} \) with compact support, such that \( \phi \) is not the zero function. For notational convenience, define for \( m \in \mathbb{Z}^n_+ \) with \( \sum_i m_i = k \) the operator
\[
D^m = \frac{\partial^k}{\partial x_1^{m_1} \cdots \partial x_n^{m_n}}.
\]
Then, define the function \( \psi \) as
\[
\psi(x) := \sum_{j=1}^{d} c_j D^{m_j} \phi(x).
\]
Note that \( \psi \in C^\infty_c(\mathbb{R}^n) \) and also \( \psi \neq 0 \). Using repeated partial integration and the fact that \( \phi \) has compact support, we see that the Fourier transform of \( \psi \) becomes
\[
\hat{\psi}(x) = \frac{1}{(2\pi)^{n/2}} \int_{\mathbb{R}^n} e^{-iy \cdot x} \psi(y) \, dy = \frac{1}{(2\pi)^{n/2}} \sum_{j=1}^{d} c_j \int_{\mathbb{R}^n} e^{-iy \cdot x} D^{m_j} \phi(y) \, dy
\]
\[
= \frac{i^k}{(2\pi)^{n/2}} \sum_{j=1}^{d} c_j x_1^{m_{j,1}} \cdots x_n^{m_{j,n}} \int_{\mathbb{R}^n} e^{-iy \cdot x} \phi(y) \, dy = i^k \hat{\phi}(x) p(x) .
\]
(2.1)

Because of the homogeneity of \( p \), we find that \( p(\lambda a) = \lambda^k p(a) = 0 \) for all \( a \in A \). By equation (2.1), this also implies that \( \hat{\psi}(\lambda a) = 0 \) for all \( a \in A \). For such an \( a \), we find
\[
0 = \hat{\psi}(\lambda a) = \frac{1}{(2\pi)^{n/2}} \int_{\mathbb{R}^n} \psi(y) e^{-i\lambda a \cdot y} \, dy
\]
\[
= \frac{1}{(2\pi)^{n/2}} \int_{-\infty}^{\infty} \int_{a \cdot y = t} \psi(y) \, dy e^{-i\lambda t} \, dt.
\]
Since the equality holds for all $\lambda \in \mathbb{R}$, it follows that
\[
\int_{a \cdot x = t} \psi(x) \, dx = 0, \quad \forall t \in \mathbb{R}.
\]
Hence, for all $g \in C(\mathbb{R})$ and $a \in A$,
\[
\int_{\mathbb{R}^n} g(a \cdot x) \psi(x) \, dx = \int_{-\infty}^{\infty} \left( \int_{a \cdot x = t} \psi(x) \, dx \right) g(t) \, dt = 0.
\]
Thus, the positive linear functional defined by
\[
F(f) := \int_{\mathbb{R}^n} f(x) \psi(x) \, dx,
\]
for $f \in C_c(\mathbb{R}^n)$, annihilates $\mathcal{R}(A)$.
Let $K \subset \mathbb{R}^n$ be compact such that $K$ contains the support of $\psi$. On $C(K)$ we can now apply the Riesz-Markov-Kakutani Representation Theorem (Theorem A.3 in Appendix) to find that there exists a unique Borel measure $\mu$ on $K$ such that
\[
F(f) = \int_K f(x) \, d\mu(x),
\]
for all $f \in C_c(K)$. By the denseness of $\mathcal{R}(A)$ in $C(K)$, we must have that $\mu$ is the zero measure on $K$, making $F$ an operator mapping everything to zero. This however is a contradiction, since $F(\psi) > 0$. Because this holds for any $k \in \mathbb{N}$, we conclude that there exists no nontrivial homogeneous polynomial $p$ that vanishes on $A$.

\[\leftarrow \text{ Select any } k \in \mathbb{Z}_+. \text{ By assumption no nontrivial homogeneous polynomial of degree } k \text{ is identically zero on } A.\]

For any $a \in A$, define
\[
g(a \cdot x) := (a \cdot x)^k = \sum_{j=1}^{N} \binom{k}{m_j,1,\ldots,m_j,n} a^{m_j} x^{m_j} = \sum_{j=1}^{N} \frac{k!}{m_j,1! \cdots m_j,n!} a^{m_j} x^{m_j},
\]
where $N = \binom{n+k-1}{k}$. Then $g(a \cdot x)$ is in both $\mathcal{R}(A)$ and $H^n_k$. The linear space $H^n_k$ has dimension $N$ and a basis consisting of $x^{m_j}$, for $m_j$ such that $\sum_{i=1}^{n} m_{j,i} = k$ and $j = 1, \ldots, N$. Therefore, its dual space $(H^n_k)'$ has a basis of functionals $f_j$, $j = 1, \ldots, N$, for which $f_j(x^{m_j}) = \delta_{ji}$, where $\delta$ is the Kronecker delta. See e.g. [34], Theorem A.4 in the Appendix for this result.

Now note that
\[
D^{m_j} x^{m_j} = \delta_{ji} m_{j,1}! \cdots m_{j,n}!.
\]
Hence, for a linear functional $T$ on $H^n_k$, there exists a polynomial $q \in H^n_k$ such that $T(p) = q(D)p$, for each $p \in H^n_k$. Also, for any $q \in H^n_k$, we see $q(x) = \sum_{j=1}^{N} b_j x^{m_j}$, for
some coefficients $b_j \in \mathbb{R}$, $j = 1, \ldots, N$. So

$$q(D)g(a \cdot x) = \sum_{j=1}^{N} b_j D^{m_j} \left( \sum_{i=1}^{N} \frac{k!}{m_{i,1}! \cdots m_{i,n}!} a_{m_j}^{m_i} x_{m_i} \right)$$

$$= k! \sum_{j=1}^{N} b_j a_{m_j}$$

$$= k! q(a).$$

Thus, if the linear operator $T$ annihilates $g(a \cdot x)$ for all $a \in A$, then its representing polynomial $q \in H^n_k$ vanishes on $A$. By assumption, this means that $q$ is the zero polynomial, hence $T$ maps every element in $H^n_k$ to zero. This shows that no nontrivial linear functional working on $H^n_k$ can map $g(a \cdot x)$ to zero for all $a \in A$. Therefore we see that $H^n_k = \text{span}\{ f : \mathbb{R}^n \to \mathbb{R} | f(x) = g(a \cdot x); a \in A \} \subseteq \mathcal{R}(A)$.

Since this holds for any $k \in \mathbb{Z}_+$, we note that $\mathcal{R}(A)$ contains all homogeneous polynomials of any degree, and therefore all polynomials. The Weierstrass Approximation Theorem, or the more general Stone-Weierstrass Theorem (Theorem B.3 in Appendix B), now yields that $\mathcal{R}(A)$ is uniformly dense on compacta in $C(\mathbb{R}^n)$.

To give an example of a set $A$ to which we can apply Theorem 2.4, note that if $A$ contains an open subset of $S^{n-1}$ (in the relative topology), then the only homogeneous polynomial vanishing on $A$ is the zero polynomial.

For a subset $\Lambda \subset \mathbb{R}$ and a subset $A \subseteq S^{n-1}$, we denote the set $\{ \lambda a | \lambda \in \Lambda, a \in A \}$ by $\Lambda \land A$. Then, to reduce the dimension of our problem, we have the following simple proposition.

**Proposition 2.5.** Assume $\Lambda, \Theta \subseteq \mathbb{R}$ such that

$$\mathcal{N}(\sigma; \Lambda, \Theta) := \text{span}\{ g : \mathbb{R} \to \mathbb{R} | g(x) = \sigma(\lambda x + b); \lambda \in \Lambda, b \in \Theta \}$$

is uniformly dense on compacta in $C(\mathbb{R})$. Furthermore, assume that $A \subseteq S^{n-1}$ is such that $\mathcal{R}(A)$ is uniformly dense on compacta in $C(\mathbb{R}^n)$. Then

$$\Sigma^n(\sigma; \Lambda \land A, \Theta) := \text{span}\{ g : \mathbb{R}^n \to \mathbb{R} | g(x) = \sigma(w \cdot x + b); w \in \Lambda \land A, b \in \Theta \}$$

is uniformly dense on compacta in $C(\mathbb{R}^n)$.

**Proof.** Let $f \in C(\mathbb{R}^n)$ and $K \subseteq \mathbb{R}^n$ be compact. Let $\varepsilon > 0$. By denseness on compacta of $\mathcal{R}(A)$ in $C(\mathbb{R}^n)$, there exists a $d \in \mathbb{N}$, functions $g_i \in C(\mathbb{R})$ and $a^i \in A$ for $i = 1, \ldots, d$ for which

$$|f(x) - \sum_{i=1}^{d} g_i(a^i \cdot x)| < \frac{\varepsilon}{2}, \quad \forall x \in K.$$
Now, because $K$ is compact, and therefore bounded, there exist finite (compact) intervals $[\alpha_i, \beta_i]$, for $i = 1, \ldots, d$, such that $\{a^i \cdot x \mid x \in K\} \subseteq [\alpha_i, \beta_i]$. Furthermore, by the assumed denseness of $\mathcal{N}(\sigma; \Lambda, \Theta)$ in $C([\alpha_i, \beta_i])$, for $i = 1, \ldots, d$, we can find constants $c_{ij} \in \mathbb{R}$, $\lambda_{ij} \in \Lambda$ and $b_{ij} \in \Theta$, for $j = 1, \ldots, m_i$ and some $m_i \in \mathbb{N}$, such that
\[
|g_i(x) - \sum_{j=1}^{m_i} c_{ij} \sigma(\lambda_{ij} x + b_{ij})| < \frac{\varepsilon}{2d},
\]
for all $x \in [\alpha_i, \beta_i]$ and $i = 1, \ldots, d$. Hence, by the triangle inequality,
\[
|f(x) - \sum_{i=1}^{d} \sum_{j=1}^{m_i} c_{ij} \sigma(\lambda_{ij} a^i \cdot x + b_{ij})| < \varepsilon, \quad \forall x \in K.
\]
This shows the result. \( \square \)

Now, being allowed to do the work on $\mathbb{R}$, we will show a density result for activation functions in $C^\infty(\mathbb{R})$, the set of infinitely differentiable functions. We first need an adaptation of a lemma from [8].

**Lemma 2.6.** Let $f$ be in $C^\infty(\mathbb{R})$ such that for every point $x \in \mathbb{R}$ there exists a $k_x \in \mathbb{N}$ for which $f^{(k_x)}(x) = 0$. Then $f(x)$ is a polynomial.

**Proof.** Define $G$ as the open set of all points for which there exists an open neighbourhood on which $f(x)$ equals a polynomial. In other words, $G$ is the set of points for which there exists an open neighbourhood on which $f^{(k)}(x) = 0$ for some $k$ (and hence for all $k' > k$ as well). Define the closed set $F := G^c$. We assume $F$ is not empty and work towards a contradiction.

First note that $F$ cannot have isolated points. To see this, suppose $x_0$ is an isolated point of $F$. Then there are $a, b \in \mathbb{R}$ such that $(a, x_0) \subset G$ and $(x_0, b) \subset G$. On $(a, x_0)$, $f$ coincides with a polynomial which can be found by the Taylor expansion of $f$ around $x_0$. The same holds for the polynomial $f$ coincides with on the interval $(x_0, b)$. Together, this means that $f$ coincides with this polynomial on the whole interval $(a, b)$, showing that $x_0$ cannot be in $F$.

We will next define the closed sets $E_n$ as the subsets of $F$ on which $f^{(n)}$ vanishes identically. Note that $F = \cup_n E_n$, by the assumed property of the derivatives of $f$. Clearly, being a closed subset of a complete metric space, $F$ is complete as well and we can apply the Baire Category Theorem (Theorem A.6 in Appendix A). This theorem states that $F$ cannot be the countable union of nowhere dense sets, implying that the interior of $E_N$ in $F$ must be nonempty for some $N$. In other words, it contains an open ball in $F$: an $x_0 \in F$, such that for $\varepsilon > 0$ small enough, all points $y \in F$ with $d(x_0, y) < \varepsilon$ are also in $E_n$. Now take $I$ to be a closed interval around $x_0$, small enough to have $F \cap I \subseteq E_N$. Then clearly $F^{(N)}(x) = 0$ for all $x \in F \cap I$. Because $F$ does not have isolated points and $f^{(N)}(x) - f^{(N)}(y) = 0$ for all $x, y \in F \cap I$, it follows that $f^{(N+1)}$ also vanishes identically on $F \cap I$. Repeating the argument show that $f^{(m)}(x) = 0$ for all $m \geq N$ and $x \in F \cap I$. 

31
Furthermore, we realize that \( F \cap I \) cannot contain an interval, since that interval would then belong to \( G \). Hence, \( G \cap I \) is not empty. Because \( G \) is open and \( I \) is an interval, there must be a (small) interval \((a,b)\) in \( G \cap I \) with endpoints \( a,b \in F \cap I \). By the nature of \( G \), \( f \) is equal to a polynomial on \((a,b)\), hence, as before, this polynomial can be obtained from its Taylor expansion around either of the endpoints. Since all derivatives \( f^{(k)}(a) = f^{(k)}(b) = 0 \) for all \( k \geq N \), we see that \( f^{(N)}(x) = 0 \) on the whole of \((a,b)\). Since this holds for any arbitrary interval in \( G \cap I \) and also on \( F \cap I \), we can conclude that \( f^{(k)}(x) = 0 \) on all of \( I \) for all \( k \geq N \). This contradicts \( x_0 \in F \), since we have shown that at least the interior of \( I \) is in \( G \). We conclude that \( F = \emptyset \), thus \( f \) is a polynomial. \( \square \)

The proof of the following proposition uses the converse of the lemma. For now we take \( \Lambda = \Theta = \mathbb{R} \). Denote by \( \mathcal{P} \) the set of all polynomials.

**Proposition 2.7.** Let \( \sigma \in C^\infty(\mathbb{R}) \) and \( \sigma \notin \mathcal{P} \). Then \( \mathcal{N}(\sigma; \mathbb{R}, \mathbb{R}) \) is dense in \( C(\mathbb{R}) \) uniformly on compacta.

**Proof.** By the converse of Lemma 2.6, there exists a point \( b_0 \) such that \( \sigma^{(k)}(b_0) \neq 0 \) for all \( k \in \mathbb{N} \).

Calculating the derivative of \( \sigma(\lambda x + b_0) \) in \( \lambda = 0 \), we find
\[
\frac{d}{d\lambda} \sigma(\lambda x + b_0) \bigg|_{\lambda=0} = \lim_{h \to 0} \frac{\sigma((\lambda + h)x + b_0) - \sigma(\lambda x + b_0)}{h} \bigg|_{\lambda=0} = x \sigma'(b_0). \tag{2.2}
\]

So \( x \sigma'(b_0) \in \mathcal{N}(\sigma; \mathbb{R}, \mathbb{R}) \), since it is the limit of elements in \( \mathcal{N}(\sigma; \mathbb{R}, \mathbb{R}) \). Similarly
\[
\frac{d^k}{d\lambda^k} \sigma(\lambda x + b_0) \bigg|_{\lambda=0} = x^k \sigma^{(k)}(b_0), \tag{2.3}
\]

and \( x^k \sigma^{(k)}(b_0) \in \mathcal{N}(\sigma; \mathbb{R}, \mathbb{R}) \) for all \( k \in \mathbb{N} \). Because \( \sigma^{(k)}(b_0) \neq 0 \) for all \( k \in \mathbb{N} \), we have shown that \( \mathcal{N}(\sigma; \mathbb{R}, \mathbb{R}) \) contains all the monomials of all degrees and therefore all polynomials. Let \( K \subseteq \mathbb{R} \) compact. Then from the Weierstrass Approximation Theorem it follows that \( \mathcal{N}(\sigma; \mathbb{R}, \mathbb{R}) \) is dense in \( C(K) \). \( \square \)

We can easily combine the above results into the following Corollary.

**Corollary 2.8.** Let \( \sigma \in C^\infty(\mathbb{R}) \) and \( \sigma \notin \mathcal{P} \). Then \( \Sigma^n(\sigma) \) is uniformly dense on compacta in \( C(\mathbb{R}^n) \).

**Proof.** By Proposition 2.7, it follows that \( \mathcal{N}(\sigma; \mathbb{R}, \mathbb{R}) \) is uniformly dense on compacta in \( C(\mathbb{R}) \). Taking \( A \) equal to the unit sphere \( S^{n-1} \), \( \mathcal{R}(A) \) is uniformly dense on compacta in \( C(\mathbb{R}^n) \) by Theorem 2.4. Now, by Proposition 2.5 we conclude that \( \Sigma^n(\sigma; \mathbb{R}^n, \mathbb{R}) \) is dense in \( C(\mathbb{R}^n) \), uniformly on compacta. \( \square \)

When working with \( C^\infty \) activation functions to approximate continuous functions, this result is sufficient. But we can even take our weights and biases from a smaller set than \( \mathbb{R} \) and still have the uniform approximation result. Looking at the proof of Proposition
2.7, for Lemma 2.6 to hold, we only need $\Theta$ to be an open interval on which $\sigma$ is not a polynomial. Further on in the proof, we see that the only constraint on $\Lambda$ is that we can evaluate equations (2.2) and (2.3). We can therefore give a more general corollary to the result.

**Corollary 2.9.** Let $\Lambda, \Theta \subseteq \mathbb{R}$ be such that $\Lambda$ contains a sequence tending to zero and $\Theta$ is an open interval. Furthermore let $A \subseteq S^{n-1}$ such that $A$ contains an open set in the relative topology and take $\sigma \in C^\infty(\Theta)$ such that $\sigma$ is no polynomial on $\Theta$. Then $N(\sigma; \Lambda, \Theta)$ is dense in $C(\mathbb{R})$ and $\Sigma^n(\sigma; \Lambda \land A, \Theta)$ is dense in $C(\mathbb{R}^n)$, both uniformly on compacta.

We will now generalize the result of Proposition 2.7 to continuous activation functions.

**Proposition 2.10.** Let $\sigma \in C(\mathbb{R})$ and $\sigma \notin \mathcal{P}$. Then $N(\sigma; \mathbb{R}, \mathbb{R})$ is dense in $C(\mathbb{R})$ uniformly on compacta.

**Proof.** For any $\phi \in C_\infty^\infty(\mathbb{R})$, the class of infinitely differentiable functions with compact support, define 

$$\sigma\phi(x) := \sigma * \phi(x) = \int_{-\infty}^{\infty} \sigma(x-y)\phi(y) \, dy,$$

the convolution of $\sigma$ and $\phi$. Because $\phi$ has compact support and $\sigma$ and $\phi$ are continuous, $\sigma\phi$ is well-defined for every $t$ (the integral converges everywhere) and $\sigma\phi \in C_\infty^\infty(\mathbb{R})$. If we view the integral as a limit of Riemann sums, we see that $\sigma\phi \in N(\sigma; \{1\}, \mathbb{R})$. Since $\sigma\phi(\lambda x + b) = \int_{-\infty}^{\infty} \sigma(\lambda x + b - y)\phi(y) \, dy$,

we then have $N(\sigma\phi; \mathbb{R}, \mathbb{R}) \subseteq N(\sigma; \mathbb{R}, \mathbb{R})$. As in the proof of Proposition 2.7, we find that $x^k\sigma^{(k)}\phi(b) \in N(\sigma\phi; \mathbb{R}, \mathbb{R})$ for every $b \in \mathbb{R}$ and all $k \in \mathbb{N}$.

We will finish the proof by contradiction. Suppose $N(\sigma; \mathbb{R}, \mathbb{R})$ is not dense in $C(\mathbb{R})$. Then, it must be that $x^k \notin N(\sigma; \mathbb{R}, \mathbb{R})$ for some $k$. This implies that $x^k \notin N(\sigma\phi; \mathbb{R}, \mathbb{R})$, hence $\sigma^{(k)}\phi(b) = 0$ for all $b \in \mathbb{R}$. By Lemma 2.6 it now follows that $\sigma\phi$ must be a polynomial. Because its $k$'th derivative is identically zero, we also see that the degree of $\sigma\phi$ is at most $k - 1$. Note that the above holds for any $\phi \in C_\infty^\infty(\mathbb{R})$.

The next step is to find a sequence of $\phi_n \in C_\infty^\infty(\mathbb{R})$ such that $\sigma\phi_n \rightarrow \sigma$ uniformly on compacta. Taking the $\phi_n$ to be mollifiers, see Definition A.7 and Theorem A.9 in Appendix A.2, the desired convergence holds. Since all $\sigma\phi_n$ are polynomials of degree at most $k - 1$ and the space of polynomials of a certain degree is a closed linear space, we conclude that $\sigma$ must also be a polynomial of degree at most $k - 1$. This contradiction shows the result. 

We are now ready to state the equivalent of Corollary 2.8 for $\sigma \in C(\mathbb{R})$, the main result of this section. The proof is very similar, only with $\sigma \in C(\mathbb{R})$ and Proposition 2.10 instead of Proposition 2.7, and is therefore omitted.
Corollary 2.11. Let $\sigma \in C(\mathbb{R})$ and $\sigma \not\in \mathcal{P}$. Then $\Sigma^n(\sigma)$ is uniformly dense on compacta in $C(\mathbb{R}^n)$.

This already is an important result showing the capabilities of artificial neural networks, and in many cases this is enough. However, we can even show that neural networks are dense in some way in $\mathcal{M}^n$, the set of measurable functions from $\mathbb{R}^n$ to $\mathbb{R}$. To prove this, we will first have to define the right metric on $\mathcal{M}^n$.

**Definition 2.12 (Ky Fan metric).** Let $\mu$ be a probability measure on $(\mathbb{R}^n, \mathcal{B}(\mathbb{R}^n))$. We define the metric $\rho_\mu$ on $\mathcal{M}^n$ by

$$\rho_\mu(f, g) = \inf \{\varepsilon > 0 : \mu(|f - g| > \varepsilon) < \varepsilon\}.$$ 

From the definition of this metric, we see that two functions are close when the probability that they differ significantly is small. The relevance of this metric in approximation with neural networks is when we take $\mu$ as the probability of occurrence of certain inputs. If a neural network is close to the target function $f$ with respect to $\rho_\mu$, it can only differ significantly from $f$ on input sets that occur with small probability.

In Lemma 2.13 we show that convergence with respect to $\rho_\mu$ is equivalent to convergence in probability. We will formulate and prove three lemmas on the properties of this newly defined metric.

**Lemma 2.13.** Let $\mu$ be a probability measure on $(\mathbb{R}^n, \mathcal{B}(\mathbb{R}^n))$. For $f, f_1, f_2, \ldots \in \mathcal{M}^n$, the following statements are equivalent.

1. $\rho_\mu(f_m, f) \to 0$.
2. For every $\varepsilon > 0$ we have $\mu(|f_m - f| > \varepsilon) \to 0$ (convergence in probability).
3. $\int \min\{|f_m(x) - f(x)|, 1\} \, d\mu(x) \to 0$.

**Proof.** (i) $\implies$ (ii): Take $\varepsilon > 0$. Want to show that $\mu(|f_m - f| > \varepsilon) \to 0$. So let $\varepsilon' > 0$. Now by (i) we see that for $n$ big enough $\mu(|f_n - f| > \varepsilon') < \varepsilon$. So assume $\varepsilon' < \varepsilon$ (otherwise we are done). In this case, again by (i) we can find a large $m$ such that $\mu(|f_m - f| > \varepsilon) < \mu(|f_m - f| > \varepsilon') < \varepsilon'$.

(ii) $\implies$ (i): Let $\varepsilon > 0$. By (ii) we now that there exists a $N$ such that $m > N$ implies $\mu(|f_m - f| > \varepsilon) < \varepsilon$. But this immediately implies that $\rho_\mu(f_m, f) = \inf \{\varepsilon > 0 : \mu(|f_m - f| > \varepsilon) \leq \varepsilon\}$ for all $m > N$.

(iii) $\implies$ (ii): Let $\varepsilon > 0$. By virtue of (iii) there exists an $N$ such that $m > N$ implies $\mu(|f_m - f| > \varepsilon/2) < \varepsilon/2$. Hence

$$\int \min\{|f_m(x) - f(x)|, 1\} \, d\mu(x) \leq \mu(|f_m - f| > \varepsilon/2) + \varepsilon/2 < \varepsilon.$$

(iii) $\implies$ (ii): Fix $\varepsilon > 0$. We see that

$$\int \min\{|f_m(x) - f(x)|, 1\} \, d\mu(x) = \mu(|f_m - f| > 1) + \int |f_m(x) - f(x)|1_{\{|f_m - f| \leq 1\}} \, d\mu(x).$$
Since both terms are positive, by (iii) both go to 0 as $m \to \infty$. For the second term, we note that

\[
\int |f_m(x) - f(x)| 1_{\{|f_m - f| \leq 1\}} \, d\mu(x) \geq \int |f_m(x) - f(x)| 1_{\{\varepsilon < |f_m - f| \leq 1\}} \, d\mu(x) \geq \varepsilon \mu(\varepsilon < |f_m - f| \leq 1),
\]

thus $\mu(\varepsilon < |f_m - f| \leq 1) \to 0$. Then, as $m \to \infty$,

\[
\mu(|f_m - f| > \varepsilon) = \mu(|f_m - f| > 1) + \mu(\varepsilon < |f_m - f| \leq 1) \to 0.
\]

The next lemma, Lemma 2.14, relates uniform convergence on compacta to $\rho_\mu$ convergence.

**Lemma 2.14.** Let $\mu$ be a probability measure on $(\mathbb{R}^n, B(\mathbb{R}^n))$. Let $f, f_1, f_2, \ldots \in \mathcal{M}_n$. If $f_m \to f$ uniformly on compacta, then $\rho_\mu(f_m, f) \to 0$.

**Proof.** We will show that (iii) from Lemma 2.13 holds. This then implies $\rho_\mu$ convergence. Let $\varepsilon > 0$. Because $\mu$ is a probability measure on a complete separable metric space, it is tight and we can take a closed and bounded (hence compact) ball $K$ big enough such that $\mu(K) > 1 - \varepsilon/2$. Since $f_m \to f$ uniformly on compacta, we can find an $N$ such that $m \geq N$ implies $\sup_{x \in K} |f_m(x) - f(x)| < \varepsilon/2$. Combining these we find that for all $m \geq N$

\[
\int \min\{|f_m(x) - f(x)|, 1\} \, d\mu(x) = \int 1_K \min\{|f_m(x) - f(x)|, 1\} \, d\mu(x) + \int 1_{K^c} \min\{|f_m(x) - f(x)|, 1\} \, d\mu(x) < \frac{\varepsilon}{2} + \frac{\varepsilon}{2} = \varepsilon.
\]

Before we can state and prove the third lemma on the metric $\rho_\mu$, we need to prove a lemma on the denseness of $C(\mathbb{R}^n)$ in the set of integrable functions. Therefore, we need to introduce the concept of a regular measure.

**Definition 2.15.** Let $(X, T)$ be a topological space with $\mathcal{F} = \sigma(T)$ a $\sigma$-algebra on $X$. Let $\mu$ be a measure on $(X, \mathcal{F})$. A measurable subset $A$ of $X$ is called **inner regular** if

\[
\mu(A) = \sup\{\mu(F) \mid F \subseteq A, F \text{ closed}\}
\]

and **outer regular** if

\[
\mu(A) = \inf\{\mu(G) \mid G \supseteq A, G \text{ open}\}.
\]

A measure is called inner regular (outer regular) if every measurable set is inner regular (outer regular). If a measure is both outer regular and inner regular, it is called a **regular measure**.
We will call a measure defined on all Borel sets a *Borel measure*. We will now show that every finite Borel measure is regular.

**Proposition 2.16.** Let \( X \) be a metric space endowed with the Borel \( \sigma \)-algebra. A finite Borel measure \( \mu \) on \( X \) is regular.

**Proof.** Define \( S \) as the collection of measurable regular sets \( A \), i.e. the sets for which

\[
\mu(A) = \inf\{\mu(G) | G \supseteq A, G \text{ open}\}
\]

\[
= \sup\{\mu(F) | F \subseteq A, F \text{ closed}\}.
\]

We will show that \( S \) contains the Borel \( \sigma \)-algebra, which proves the proposition.

Firstly, note that \( S \) contains all measurable open sets. Clearly for \( A \) open, we have \( \mu(A) \geq \inf\{\mu(G) | G \supseteq A \text{ open}\} \) since \( A \) itself is open. Also \( \mu(A) \leq \inf\{\mu(G) : G \supseteq A \text{ open}\} \) since for every \( G \supseteq A \) it holds that \( \mu(G) \geq \mu(A) \). Furthermore, since in a metric space every open set is a countable union of closed sets, we can conclude that \( A \in S \). We will finish the proof by showing that \( S \) is a \( \sigma \)-algebra.

Clearly \( \emptyset \in S \). Take \( A \in S \) and let \( \varepsilon > 0 \). Since \( A \in S \) we can find sets \( F \subseteq A \subseteq G \) with \( F \) closed, \( G \) open and \( \mu(A) < \mu(F) + \varepsilon, \mu(A) > \mu(G) - \varepsilon \). Then \( G^c \subseteq A^c \subseteq F^c \) and \( G^c \) closed, \( F^c \) open. Also

\[
\mu(A^c) = \mu(X) - \mu(A) > \mu(X) - \mu(F) - \varepsilon = \mu(F^c) - \varepsilon,
\]

and

\[
\mu(A^c) = \mu(X) - \mu(A) < \mu(X) - \mu(G) + \varepsilon = \mu(G^c) + \varepsilon.
\]

This shows \( A^c \in S \).

Now let \( A_1, A_2, \ldots \in S \). We want to show that the countable union of these sets is again in \( S \). Let \( \varepsilon > 0 \). For \( n \in \mathbb{N} \) choose \( G_n \supseteq A_n \) open, such that \( \mu(G_n \setminus A_n) < \varepsilon / 2^n \). Then \( \bigcup_{n=1}^\infty G_n \) is open and \( \mu(\bigcup_{n=1}^\infty G_n \setminus \bigcup_{n=1}^\infty A_n) \leq \mu(\bigcup_{n=1}^\infty G_n \setminus A_n) < \varepsilon \).

Since \( \mu \) is finite, there exists an \( N \) such that \( \mu(\bigcup_{n=1}^N A_n) - \mu(\bigcup_{n=1}^N A_n) < \varepsilon / 2 \). We can also find \( F_n \subseteq A_n \) closed for \( n = 1, \ldots, N \) such that \( \mu(A_n \setminus F_n) < \frac{\varepsilon}{2N} \). Now \( \bigcup_{n=1}^N F_n \) is closed and contained in \( \bigcup_{n=1}^\infty A_n \). Furthermore

\[
\mu(\bigcup_{n=1}^\infty A_n) - \mu(\bigcup_{n=1}^N F_n) = \mu(\bigcup_{n=1}^\infty A_n) - \mu(\bigcup_{n=1}^N A_n) + \mu(\bigcup_{n=1}^N A_n) - \mu(\bigcup_{n=1}^N F_n)
\]

\[
< \frac{\varepsilon}{2} + \mu(\bigcup_{n=1}^N A_n \setminus F_n)
\]

\[
< \frac{\varepsilon}{2} + N \frac{\varepsilon}{2N} = \varepsilon.
\]

This shows \( S \) is a \( \sigma \)-algebra containing the open sets, hence \( B \subseteq S \). We conclude that \( \mu \) is a regular measure. \( \square \)

Using this result we can prove the lemma on the denseness of \( C(X) \) in the set of integrable functions.
Lemma 2.17. Let \((X, d)\) be a metric space endowed with the Borel \(\sigma\)-algebra and let \(\mu\) be a finite Borel measure on \(X\). Then \(C(X)\) is dense in \(L^1(X, \mu)\), the space of (equivalence classes of) integrable functions on \(X\).

Proof. Let \(A \in \mathcal{B}(X)\). Since \(\mu\) is a finite measure on a metric space, by Proposition 2.16 \(\mu\) is regular. Therefore, we can find sequences of open sets \(G_n \supseteq A\) and closed sets \(F_n \subseteq A\) such that \(\mu(A \setminus F_n) \to 0\) and \(\mu(G_n \setminus A) \to 0\) as \(n \to \infty\).

For any nonempty closed \(C \subset X\) we will define the (continuous) function \(d_C\) on \(X\) measuring the distance to \(C\) by

\[
d_C(x) := \inf_{y \in C} d(x, y).
\]

Next define \(f_n(x) := \frac{d_{G_n}(x)}{d_{G_n}(x) + d_{F_n}(x)}\).

Then \(f_n\) is continuous so \(f_n \in C(X)\) and \(f_n : X \to [0, 1]\), such that \(f_n = 0\) on the closed set \(G_n\) and \(f_n = 1\) on \(F_n\). Hence \(\mathbb{1}_{F_n} \leq f_n \leq \mathbb{1}_{G_n}\). But this implies that \(f_n \to \mathbb{1}_A\) in \(L^1\) by virtue of the squeeze lemma. This can be done for any \(A \in \mathcal{B}(X)\).

Now we have shown that we are able to approximate the indicator of any measurable set, we see that the indicator functions lie in \(C(X)\). By linearity this means that also the simple functions are in \(C(X)\). Since we can approximate functions in \(L^1(X, \mu)\) by simple functions, we can conclude that \(L^1(X, \mu)\) is also contained in \(C(X)\) with respect to the \(L^1\) norm. \(\square\)

We will use the above result to show the third lemma on the \(\rho_\mu\)-metric.

Lemma 2.18. For any probability measure \(\mu\) on \(\mathbb{R}^n\), \(C(\mathbb{R}^n)\) is dense in \(\mathcal{M}^n\), the set of measurable functions from \(\mathbb{R}^n\) to \(\mathbb{R}\), with respect to the \(\rho_\mu\)-metric.

Proof. Let \(f \in \mathcal{M}^n\) and let \(\varepsilon > 0\). By the finiteness of the measure \(\mu\), we can find a large \(M\) such that \(\mu(|f| > M) < \varepsilon/2\). But this implies that

\[
\int \min\{|f|_{|f| < M}, 1\} \, d\mu = \int \mathbb{1}_{|f| > M} \min\{|f|, 1\} \, d\mu < \mu(|f| > M) < \frac{\varepsilon}{2}.
\]

Using Lemma 2.17, we can find a function \(g \in C(\mathbb{R}^n)\) such that

\[
\int |f|_{|f| < M} - g| \, d\mu < \frac{\varepsilon}{2}.
\]

Application of the triangle inequality now yields

\[
\int \min\{|g - f|, 1\} \, d\mu \leq \int \min\{|g - f|_{|f| < M}| + |f|_{|f| < M} - f|, 1\} \, d\mu
\]

\[
\leq \int \min\{|g - f|_{|f| < M}|, 1\} \, d\mu + \int \min\{|f|_{|f| < M} - f|, 1\} \, d\mu
\]

\[
< \frac{\varepsilon}{2} + \frac{\varepsilon}{2} = \varepsilon.
\]

By Lemma 2.13 we can now conclude that \(C(\mathbb{R}^n)\) is dense in \(\mathcal{M}^n\) with respect to \(\rho_\mu\). \(\square\)
Finally we are ready to prove the denseness of $\Sigma^n(\sigma)$ in $M^n$. Fortunately, we have already done most of the work.

**Corollary 2.19.** Let $\mu$ be a probability measure on $(\mathbb{R}^n, B(\mathbb{R}^n))$ and let $\sigma \in C(\mathbb{R})$ such that $\sigma \not\in P$. Then $\Sigma^n(\sigma)$ is dense in $M^n$ with respect to the metric $\rho_\mu$.

**Proof.** By Theorem 2.11, $\Sigma^n(\sigma)$ is uniformly dense on compacta in $C(\mathbb{R}^n)$. By Lemma 2.14, this means that $\Sigma^n(\sigma)$ is also dense in $C(\mathbb{R}^n)$ with respect to the $\rho_\mu$-metric. In the same metric, $C(\mathbb{R}^n)$ is dense in $M^n$, by Lemma 2.18. By the triangle inequality we may now conclude that $\Sigma^n(\sigma)$ is dense in $M^n$ with respect to $\rho_\mu$. \qed

In the following corollary, we will extend the result of Corollaries 2.11 and 2.19 to multi-output networks.

**Definition 2.20.** For any activation function $\sigma \in C(\mathbb{R})$ such that $\sigma \not\in P$ and $n, m \in \mathbb{N}$ we define the class of functions

$$\Sigma^{n,m}(\sigma) := \left\{ f : \mathbb{R}^n \to \mathbb{R}^m \mid f(x) = \sum_{i=1}^d b_i \sigma(A_i(x)); b_i \in \mathbb{R}^m, A_i \in \Lambda^n, d \in \mathbb{N}\right\}.$$ 

Denote by $M^{n,m}$ the class of measurable functions from $\mathbb{R}^n$ to $\mathbb{R}^m$. We also need to expand the definition of the $\rho_\mu$ to define a metric on $M^{n,m}$.

**Definition 2.21.** Let $\mu$ be a probability measure on $(\mathbb{R}^n, B(\mathbb{R}^n))$. We define the metric $\rho_{\mu}^{m}$ on $M^{n,m}$ by

$$\rho_{\mu}^{m}(f, g) = \sum_{i=1}^m \rho_\mu(f_i, g_i),$$

where $f_i, g_i$ denote the entries of $f$ respectively $g$ and $\rho_\mu$ is the metric on $M^n$ as defined in Definition 2.12.

The approximation result for for a one-layer neural network with $m$ linear output neurons now follows easily.

**Corollary 2.22.** Let $\mu$ be a probability measure on $(\mathbb{R}^n, B(\mathbb{R}^n))$ and let $\sigma \in C(\mathbb{R})$ such that $\sigma \not\in P$. Then $\Sigma^{n,m}(\sigma)$ is uniformly dense on compacta in $C(\mathbb{R}^n, \mathbb{R}^m)$ and dense in $M^{n,m}$ with respect to the metric $\rho_{\mu}^{m}$.

**Proof.** Let $f \in C(\mathbb{R}^n, \mathbb{R}^m)$ and $\varepsilon > 0$. We will reduce the dimension of the problem by approximating the vector elements of $f$ separately. By Corollary 2.11 we can approximate elements $f_j$ for each $j \in \{1, \ldots, m\}$ uniformly on compacta. So take a compact $K \subset \mathbb{R}^n$. For $j = 1, \ldots, m$ we can find functions $g_j \in \Sigma^n(\sigma)$ such that we have $\sup_{x \in K} |g_j(x) - f_j(x)| < \varepsilon/m$. By replacing the weights $b_i^{(j)}$ in each of the approximations $g_j$ by vectors in $\mathbb{R}^m$ with zero entries except in the $j$'th position (where we put $b_i^{(j)}$) we get functions $g_j^{(j)} \in \Sigma^{n,m}(\sigma)$. Defining $g = \sum_{j=1}^m g_j^{(j)}$, we see $g \in \Sigma^{n,m}(\sigma)$ and by the triangle inequality the uniform difference on $K$ is smaller than $\varepsilon$. Since we can do
this for every compact $K \subset \mathbb{R}^n$, we conclude $\Sigma^{n,m}(\sigma)$ is uniformly dense on compacta in $C(\mathbb{R}^n, \mathbb{R}^m)$.

Exactly the same argument, but with $f \in \mathcal{M}^{n,m}$, using Corollary 2.19 and the metric $\rho_\mu$, will yield the denseness result in $\mathcal{M}^{n,m}$. \hfill \Box

We now want to move to neural networks with $L$ layers (note again that we do not count the input layer, but do count the output layer). We will denote this class of neural networks by $\Sigma^{n,m}_{L}(\sigma)$.

Before we can state the result on these networks, we need a lemma.

**Lemma 2.23.** Let $\mathcal{F}$ be a class of functions from $\mathbb{R}$ to $\mathbb{R}$ that is uniformly dense on compacta in $C(\mathbb{R})$ and $\mathcal{G}$ be a class of functions from $\mathbb{R}^n$ to $\mathbb{R}$ that is uniformly dense on compacta in $C(\mathbb{R}^n)$. The class $\mathcal{H} := \{ f \circ g : f \in \mathcal{F}, g \in \mathcal{G} \}$ is also uniformly dense on compacta in $C(\mathbb{R}^n)$.

**Proof.** Let $z \in C(\mathbb{R}^n)$, let $K \subset \mathbb{R}^n$ be compact and $\varepsilon > 0$. Since $\mathcal{G}$ is dense on compacta in $C(\mathbb{R}^n)$, we can find a $g \in \mathcal{G}$ such that $\sup_{x \in K} |g(x) - z(x)| < \varepsilon/2$. Since $z$ is a continuous function, its image of $K$, $\{z(x) : x \in K\}$, is compact, hence closed and bounded. Now because $g$ is uniformly close to $z$ on $K$, we see that $\{g(x) : x \in K\}$ is also bounded.

Define $G_K := \{g(x) : x \in K\}$. $G_K$ is closed and bounded, thus compact. Therefore, we can find an $f \in \mathcal{F}$ such that $\sup_{x \in G_K} |f(x) - x| < \varepsilon/2$. But this implies that

$$
\sup_{x \in K} |f(g(x)) - z(x)| \leq \sup_{x \in K} |f(g(x)) - g(x)| + \sup_{x \in K} |g(x) - z(x)|
$$

$$
= \sup_{x \in G_K} |f(x) - x| + \sup_{x \in K} |g(x) - z(x)|
$$

$$
< \varepsilon/2 + \varepsilon/2 = \varepsilon.
$$

We conclude $f \circ g \in \mathcal{H}$ approximates $z$ uniformly on $K$. The result follows. \hfill \Box

This last corollary will prove that multi-layer, multi-output neural networks with linear output neurons are also uniform approximators.

**Corollary 2.24.** Let $\mu$ be a probability measure on $(\mathbb{R}^n, \mathcal{B}(\mathbb{R}^n))$ and let $\sigma \in C(\mathbb{R})$ such that $\sigma \notin \mathcal{P}$. For every $L \in \mathbb{N}$ with $L \geq 2$, $\Sigma^{n,1}_L(\sigma)$ is uniformly dense on compacta in $C(\mathbb{R}^n, \mathbb{R}^m)$ and dense in $\mathcal{M}^{n,m}$ with respect to the metric $\rho_\mu$.

**Proof.** Assume $m = 1$. Since the activation of one layer in the network is the input of the next layer, we will proceed by induction to the number of layers. Suppose the number of layers $L = 2$ (one hidden layer). In Corollary 2.11 we have already shown that $\Sigma^{n,1}_2(\sigma)$ is uniformly dense on compacta in $C(\mathbb{R}^n)$. This shows the induction hypothesis.

Next suppose that $\Sigma^{n,1}_L(\sigma)$ is uniformly dense in compacta in $C(\mathbb{R}^n)$. A network with $L + 1$ layers will just be a $\Sigma^{n,1}_{L+1}(\sigma)$ function with as input the output of a $\Sigma^{n,1}_L$ network. By Corollary 2.11, $\Sigma^{n,1}_1(\sigma)$ is uniformly dense on compacta in $C(\mathbb{R})$. Hence application of Lemma 2.23 yields that the $L + 1$ layer network, $\Sigma^{n,1}_{L+1}(\sigma)$, is uniformly dense on compacta. This concludes the induction.
Repetition of the proof of Corollary 2.19 yields the result of denseness in $\mathcal{M}^{n,1}$ with respect to the $\rho_{\mu}$-metric. To extend the result to $m > 1$, apply exactly the same technique as in Corollary 2.22.

These powerful results show that artificial neural networks with any number of hidden layers are in theory capable of approximating any measurable function from $\mathbb{R}^n$ to $\mathbb{R}^m$, for any $n, m \in \mathbb{N}$, with any desired precision. That is, if the activation function is continuous and not polynomial. With a strengthened trust in artificial neural networks, we can proceed with the discussion started in Section 2.1.

2.3. Optimization methods

In this section we will discuss several iterative optimization methods that may find the optimal values for the weights and biases given the input and target values. Before we can optimize the parameters, we have to define with respect to what metric we want to minimize the difference between the targets $t$ and network outputs $y$. The first subsection will therefore introduce the cost function and give two examples of commonly used cost functions. Training of neural networks is mainly done with gradient descent optimization algorithms. Methods that involve the calculation of second-order derivatives, such as Newton’s methods, are often computationally not feasible due to the large number of parameters in a neural network. In Subsection 2.3.2, a short introduction to gradient descent optimization algorithms is given and in Subsection 2.3.3 the Adam optimization algorithm is explained.

2.3.1. Cost functions

To measure how well the output of a neural network approximates the target values in the training stage, we need to define a cost (or error) function. These cost functions are often the mean of the individual loss functions per training data point. Consider a training data set $\mathcal{X} = \{x_1, \ldots, x_N\}$ with $N$ data points and $\mathcal{T} = \{t_1, \ldots, t_N\}$ the corresponding targets (note that each data point and target is a $d_0$- respectively $d_L$-dimensional vector). Given these inputs and targets, the cost function becomes a function of the weights and biases of the network

$$C(w, b \mid \mathcal{X}, \mathcal{T}) = \frac{1}{N} \sum_{n=1}^{N} C_n(w, b \mid x_n, t_n),$$

(2.4)

where $C_n(w, b)$ is loss function of the $n$’th input. A commonly used cost function in regression problems is the mean squared error (MSE) in which the loss functions take the form $C_n(w, b \mid x, t) = \frac{1}{2} \| y - t \|^2$, where by $\| \cdot \|$ we denote the Euclidean norm and $y = y(w, b \mid x)$ is the output of the neural network.

In classification problems, where each sample can be in one class, the mean squared error may not be the best cost function to use. A loss function that is often used in
classification problems is the cross-entropy (CE) error. For example, in [35] simulations show that using the cross-entropy error outperforms the mean squared error in classification problems and even leads to faster training and improved generalization. This is also in accordance with our own observations while doing the simulations in Chapter 3. The cross-entropy loss function is defined as

\[ C_n(w, b | x, t) = -\sum_{i=1}^{d_L} t_i \log(y_i), \]

where \( y_1, \ldots, y_{d_L} \) and \( t_1, \ldots, t_{d_L} \) are the elements of the output vector \( y \) and the target vector \( t \) and represent a probability distribution.

### 2.3.2. Gradient descent

One of the most well-known optimization algorithms is the iterative method known as gradient descent. For this we need a differentiable cost function and a starting point of the algorithm. Each iteration is a step in the direction opposite to the gradient. In the case of neural networks, the weights and biases are set to their initial values and the input vectors are fed through the network to generate the output vectors. These are then compared to the target values \( t \) by applying the cost function. As mentioned above, given the input and the target values for that input, the cost function \( C \) is in fact a function of the weights \( w \) and biases \( b \) of the network (since those fully determine the output). To find a minimum of the cost function, the weights and biases are changed in the direction of steepest descent

\[
w \rightarrow w - \eta \nabla_w C, \quad b \rightarrow b - \eta \nabla_b C,
\]

where \( \eta \) is the step size, called the learning rate. When the learning rate is too big, there is a possibility of overshooting the minimum. On the other hand, a too small learning rate will result in slow convergence.

A straightforward application of this optimization method is called batch gradient descent. Output is generated for all training samples and gradient descent is applied using the average cost of all the data. This means that we only update the weights and biases once, after evaluating the network on all of the training data. This might be too computationally intensive when dealing with larger datasets, resulting in very slow learning. The training data is therefore often split into so-called mini-batches of smaller size. The weights and biases are then updated according to the gradient on a mini-batch. By choosing the mini-batches large enough, the gradient on this subset of the training data will be approximately the same as the gradient of the full training data. Therefore, the weights and biases will be updated roughly in the same direction, but only the samples in the mini-batch have to be evaluated. A full cycle through all the mini-batches is called an epoch. After one epoch the weights and biases will have been updated several times with the same number of computations that are necessary to update the weights and biases one time in batch gradient descent.
To prevent the algorithm from getting stuck in cycles and to reduce the impact of accidentally choosing unrepresentative mini-batches and as a consequence getting the gradient wrong, it is advisable to randomly select the mini-batches in each epoch. This method is sometimes referred to as stochastic gradient descent (SGD).

Given the right mini-batch size and learning rate, the stochastic gradient descent algorithm might perform well. However, there are situations where the learning progress can be slow. As is summarized in [31], one of them is where the SGD algorithm has to pass through a narrow trough with steep walls. Instead of moving along the bottom towards the local minimum, the SGD algorithm might oscillate up and down the walls. This effect is depicted in Figure 2.2. A heuristic solution to this problem is adding a momentum term when updating the weights. Instead of only updating the weights based on the current gradient, also a step is made in the direction of the gradient of the last update. This way the momentum of the movement towards the local minimum is weighted in the optimization.

![Figure 2.2:](image)

**Figure 2.2:** The left picture depicts the SGD algorithm trying to navigate through a narrow trough, oscillating of the walls. In the right picture a momentum term is added, forcing the algorithm to take the previous direction(s) into account. (Image by Sebastian Ruder [31])

### 2.3.3. The Adam optimizer

An example of an optimization method exploiting the benefits of the momentum method is the Adam algorithm introduced in [16]. Besides taking into account the mean direction in which past updates were done, it also uses the second moment of the updates. Let us denote by $\theta$ the set of weights and biases, i.e. all parameters that can be updated to minimize the cost function. It is important to realize that when applying a mini-batch algorithm, the cost function will be evaluated on a randomly selected subset of the training data. In each iteration step from $t - 1$ to $t$, the algorithm will update the parameters $\theta_{t-1}$ to $\theta_t$. To do this, a mini-batch $\mathcal{X}_t \subset \mathcal{X}$ with its corresponding set of target vectors $\mathcal{T}_t \subset \mathcal{T}$ is chosen randomly. Therefore, the cost as function of the parameters $\theta_{t-1}$ will be stochastic. We will use the shorthand $C_t(\theta_{t-1}) = C(\theta_{t-1} \mid \mathcal{X}_t, \mathcal{T}_t)$ to simplify notation.

Given the input and targets, the algorithm calculates the exponential moving average of
the gradient and the gradient squared:

\[ m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla_\theta C_t(\theta_{t-1}) \]

\[ v_t = \beta_2 v_{t-1} + (1 - \beta_2) (\nabla_\theta C_t(\theta_{t-1}))^2, \]

where the meta-parameters \( \beta_1 \) and \( \beta_2 \) are the exponential decay rates of the moving averages and \((\nabla_\theta C_t(\theta_{t-1}))^2\) is the element-wise square. The values for \( m_0 \) and \( v_0 \) are set to zero. Because of this initialization, the authors of [16] note that \( m_t \) and \( v_t \) are biased towards zero. To correct for this effect they introduce, for \( t > 0 \),

\[ \hat{m}_t = \frac{m_t}{1 - (\beta_1)^t}, \quad \hat{v}_t = \frac{v_t}{1 - (\beta_2)^t}, \]

where again all operations on vectors are element-wise. The parameters will then be updated according to the rule

\[ \theta_t = \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon}, \quad (2.5) \]

where \( \alpha \) is the learning rate and \( \varepsilon \) prevents division by zero. The creators of Adam suggest default parameters of \( \alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999 \) and \( \varepsilon = 10^{-8} \).

In [16], a proof is given of the convergence of the algorithm if the cost functions \( C_t \) are convex and have bounded gradients. Unfortunately, in most neural network applications, the cost function is not convex. However, the article shows empirically that the Adam algorithm outperforms other often used optimization algorithms even when the cost function is not convex, and has therefore become widely used.

2.4. Backpropagation

The gradient descent optimization and the Adam optimizer from Section 2.3 both involved the calculation of the derivative of the cost function with respect to every weight and bias in the network. As networks grow larger, this would mean calculating many derivatives each training cycle. Luckily, the backpropagation algorithm makes this process computationally feasible. This method is a fairly straightforward application of the multivariate chain rule, but it is easy to get lost in all the indices required for the notation. The efficiency of the backpropagation algorithm made the training of deeper neural networks possible. In this section we again follow the explanation by Nielsen [26] and Bishop [2] and adapt their notation slightly. We will arrive at a useful expression for the derivatives of the cost function with respect to the weights and biases in the network.

2.4.1. The backpropagation algorithm

Recall from (2.4) that the the cost function is the sum of the loss functions of individual data points. Therefore, if we are calculating the derivative for \( N \) data points with respect
to a weight \( w_{ji}^{(l)} \) or bias \( b_j^{(l)} \) in the network, we get

\[
\frac{\partial C}{\partial w_{ji}^{(l)}} = \sum_{n=1}^{N} \frac{\partial C_n}{\partial w_{ji}^{(l)}}, \quad \frac{\partial C}{\partial b_j^{(l)}} = \sum_{n=1}^{N} \frac{\partial C_n}{\partial b_j^{(l)}}.
\] (2.6)

To calculate these partial derivatives, we will first introduce the error of node \( j \) in layer \( l \) as

\[
\delta_j^{(l)} = \frac{\partial C_n}{\partial z_j^{(l)}},
\]

the partial derivative of the loss function with respect to the weighted input of the node \( j \) in layer \( l \). Note that we dropped the subscript \( n \) in the \( \delta \) for notational simplicity.

If we look at the error in the output layer \( L \), we find that

\[
\delta_j^{(L)} = \frac{\partial C_n}{\partial z_j^{(L)}} = \frac{\partial C_n}{\partial y_j} \frac{\partial y_j}{\partial z_j^{(L)}} = \frac{\partial C_n}{\partial y_j} h'(z_j^{(L)}),
\] (2.7)

since \( C_n \) depends on weighted input \( z_j^{(L)} \) only through its activation \( y_j = a_j^{(L)} = h(z_j^{(L)}) \). To find the error in the other layers, recall that for each \( j \) and \( l \) the weighted input is given by

\[ z_j^{(l)} = \sum_i w_{ki}^{(l)} a_i^{(l-1)} + b_j^{(l)} \]

and for \( l = 1, \ldots, L - 1 \) the activation is \( a_i^{(l)} = \sigma(z_i^{(l)}) \).

Application of the multivariate chain rule now yields

\[
\delta_j^{(l)} = \frac{\partial C_n}{\partial z_j^{(l)}} = \sum_k \frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}} \frac{\partial C_n}{\partial z_k^{(l+1)}} = \sigma'(z_j^{(l)}) \sum_k w_{kj}^{(l+1)} \delta_k^{(l+1)},
\] (2.8)

where \( \sigma' \) needs to be replaced by the derivative of the output function if \( l = L - 1 \). Since we are able to calculate the value for \( \delta_L \) by equation (2.7) we can calculate each \( \delta_j^{(l)} \) in the other layers by working backwards.

Knowing the expressions for the errors, we can calculate the partial derivatives in (2.6) by again applying the multivariate chain rule. Noting that \( C_n \) only depends on weight \( w_{ji}^{(l)} \) through the weighted input \( z_j^{(l)} \) we get

\[
\frac{\partial C_n}{\partial w_{ji}^{(l)}} = \frac{\partial C_n}{\partial z_j^{(l)}} \frac{\partial z_j^{(l)}}{\partial w_{ji}^{(l)}} = \delta_j^{(l)} a_i^{(l-1)}. \]
(2.9)

Similarly, for the derivative with respect to the biases we get

\[
\frac{\partial C_n}{\partial b_j^{(l)}} = \frac{\partial C_n}{\partial z_j^{(l)}} \frac{\partial z_j^{(l)}}{\partial b_j^{(l)}} = \delta_j^{(l)}. \]
(2.10)

Putting everything together yields the backpropagation algorithm: First forward propagate the data \( x_1, \ldots, x_N \) through the network calculating the weighted inputs and
activations of all nodes and calculating the output of the network. Then, knowing the expression for the chosen loss function, calculate the output error $\delta^{(L)}$ using equation (2.7). Backpropagate calculating the other values of $\delta^{(l)}$ for $l = L - 1, \ldots, 1$ using equation (2.8). Lastly use equations (2.9) and (2.10) and sum them according to equation (2.6) to find the derivatives of the cost function with respect to every weight and bias. Having found these derivatives, we can apply the optimization algorithm discussed in Section 2.3.

2.4.2. Learning slowdown

As we have seen above, the derivatives of the cost function with respect to the weights and biases in the network are a product of the activation and the delta of the involved node. These delta’s in turn, are products of the derivative of the activation function and the delta’s of other layers. Because the amount that the weights and biases change in each iteration step depends on the size of the gradient, a small derivative of the activation function will mean a small change in the weights. This especially becomes a problem for the first layers in deeper neural networks, where the delta’s are products of many of these derivatives. When the weighted inputs of the nodes are such that the derivatives of the activation functions are very small, the network may hardly learn anything each epoch. This effect is called learning slowdown. A smart choice of the activation function might reduce this effect, this is discussed in Section 2.5. Another way to prevent learning slowdown is by initializing the weights and biases in such a way that the algorithm starts learning in regions where derivatives of the activation functions are large, see Section 2.6.

2.5. Activation and output functions

This section will shortly discuss the choice of activation and output functions. We will start with the latter. For a more thorough discussion of output functions and their combination with the choice of cost function, see [2].

2.5.1. Output functions

The output function of a neural network is the activation function of the output layer. It is important to choose one that is well-suited for the application. In regression problems, the most common choice is a linear output function in combination with a MSE cost function. For classification problems, this is the softmax output function in combination with the cross-entropy cost function. The output of softmax output node $j$ is defined as

$$y_j = \frac{e^{z_j(L)}}{\sum_{i=1}^{d_L} e^{z_i(L)}}.$$

Note that, by design, a neural network with softmax output produces a probability distribution, since the outputs of the nodes add up to one. This makes it very suited
for classification problems (output can be interpreted as the probability of belonging to a certain class) and allows for the use of the cross-entropy cost function.

2.5.2. Activation functions

This subsection will give an overview of several activation functions, their benefits and drawbacks. See Figure 2.3 for a graph of the discussed activation functions.

Sigmoid

Historically one of the most widely used activation functions was the standard logistic function, in this context known as the sigmoid function,

$$\sigma(x) = \frac{1}{1 + e^{-x}}.$$  

Note that this function satisfies Definition C.3 of a sigmoidal function. For these sigmoidal functions, early universal approximation results were shown by Hornik, Stinchcombe and White [14] (see Section C), and Cybenko [7]. However, the sigmoid function has two major drawbacks. Because the gradients are close to zero at the tails, it suffers a lot from the learning slowdown problem discussed in Subsection 2.4.2. If the weights and biases are such that the weighted input is very negative or positive, the neuron is said to be ‘saturated’, since learning is very difficult because of the small gradient. This makes proper initialization of the weights and biases vital to networks with sigmoid neurons.

Hyperbolic tangent

Another activation function that gained popularity was the hyperbolic tangent (note that this is just rescaled sigmoid: \(\tanh(x) = 2\sigma(2x) - 1\)). The benefit of tanh over sigmoid is that its activations are centered around zero. Allowing negative activations gives the network more flexibility and improves the learning speed compared to the sigmoid activation which is always positive. On top of that, Glorot and Bengio [10] note that, especially when using softmax output, the output may initially rely more on the biases in the last layer, which have a direct effect on the output and are learned quickly, than on the activations. In the notation of Section 2.1, the output of the network is of the form softmax\((W^a_{(L-1)} + b^L)\), so if the network learns by focusing first on the biases, it will make \(W^a_{(L-1)}\) small. This can be realized by pushing the activations \(a^{(L-1)}\) to zero. In the case of the sigmoid, this poses a problem, because the zero output is in the left tail where the gradient is small. This makes it difficult to recover after the biases have converged to a stable value. In the case of the tanh activation function, an output of zero is a good place for learning, because the gradient is biggest. This makes the tanh almost always preferable over the sigmoid activation function.

However, the tanh activation function still has rapidly diminishing gradients in its tails.
Rectified Linear Unit

One relatively recent innovation solving this diminishing gradient problem is the so-called Rectified Linear Unit (ReLU). The ReLU was introduced as an activation function by Krizhevsky, Sutskever and Hinton in 2012 [17] and is currently the most popular activation function [19]. It is a simple activation function, $f(x) = \max\{0, x\}$, but is shown to accelerate convergence of the gradient descent algorithm compared to sigmoid or tanh neurons. This is likely to be the consequence of the constant gradient for positive input, which is especially useful in deep neural networks. Furthermore, the simple form of the ReLU makes it computationally more attractive than e.g. the tanh, which involves the calculation of exponentials. Because neurons with negative weighted input have zero activation, the ReLU may lead to calculations with sparse matrices. This in contrast to a sigmoid activation function, which will always give a slightly positive value, thus no computational benefit from neurons with virtually zero activation.

The fact that the gradient is zero for negative weighted input is also a downside of the ReLU. Normally, the activation of a ReLU neuron will be zero on a part of the training data and positive on the rest. The zero gradient on some data points does not harm the learning process too much, because the gradient is 1 on the other data points. However, once a neuron gets to output zero on most of the training data, it ‘dies’ and is not able to recover because of the zero gradient. Finding that a large fraction of the ReLU activations is zero during training, can indicate that the learning rate is too high. Instead of gentle updates in the learnable domain, the neuron can then jump to zero in one update step.

Despite the dying neuron problem, many of the recent achievements in applications with neural networks, such as image and speed recognition, are made with ReLU networks.

Leaky ReLU and parametric ReLU

In an effort to reduce the impact of dying neurons, the Leaky ReLU (LReLU) was introduced in 2013 ([23]). It allows a small gradient $\alpha$ where the ReLU outputs zero,

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{if } x \leq 0 \end{cases}$$

The value of $\alpha$ typically lies between 0.01 and 0.3, but the optimal value should be determined for each application. In the case of the parametric ReLU (PReLU), introduced in 2015 ([13]), the negative slope $\alpha$ is made into a learnable parameter for each neuron. For both the Leaky ReLU and PReLU there is a lot of discussion whether they are a significant improvement to ReLU. The results tend to vary per application.
Exponential Linear Unit

A very recent attempt to improve on the ReLU is the Exponential Linear Unit (ELU) introduced in 2016 ([6]). The ELU,

$$f(x) = \begin{cases} 
  x & \text{if } x > 0 \\
  \alpha(e^x - 1) & \text{if } x \leq 0 
\end{cases},$$

allows for negative values of the activation. But unlike the PReLU and Leaky ReLU it saturates for negative values. The authors of [6] argue that the saturation in the negative domain increases noise-robustness. Again, the benefits of the ELU are still under discussion.

Figure 2.3.: The sigmoid (dotted line), tanh (solid), ReLU (dashdot), leaky ReLU with $\alpha = 0.2$ (dashed) and ELU with $\alpha = 1$ (dashdot) activation functions.
2.6. Weight initialization

Because an artificial neural network learns by adapting the weights and biases, the initialization of these parameters is very important. A wrong initialization may lead to very slow learning or no convergence at all, e.g. in the case of ReLU activation if the majority of neurons is dead after initializing or for sigmoidal functions if many neurons initialize in the tails.

For a long time, people working with artificial neural networks used heuristic initialization methods. Most commonly initializing the biases at zero and drawing each weight independently from a normal distribution. By drawing them this way, they are all different and centered around zero, improving the learning speed in the case of a tanh activation function. The normal distribution was sometimes truncated to prevent large weights, which can drive the neuron into saturation upon initialization. Because the variance of the activation of layer \( l \) scales with \( d_{l-1} \), the number of neurons in the layer before it (see derivation below, equation (2.13)), the standard deviation of the the distribution from which the weights are drawn is often taken to be inversely proportional to the square root of \( d_{l-1} \). A similar scaling, but with \( d_l \), the number of neurons in layer \( l \), happens when updating the weights during backpropagation. For artificial neural networks with symmetric activation function, Glorot and Bengio [10] introduced the ‘Xavier’ initialization method in 2010. Compromising between reducing the variance of the output and the variance of the weights when backpropagating, their method comes down to drawing the weights independently from a Uniform\([-\frac{\sqrt{6}}{\sqrt{d_{l-1}+d_l}}, \frac{\sqrt{6}}{\sqrt{d_{l-1}+d_l}}]\) distribution. The Xavier method is still widely used, but more recently He et al. (2015 [13]) used a reasoning similar to Glorot and Bengio to find an initialization method especially for ReLU, LReLU and PReLU networks. We will explain the derivation of their method below.

2.6.1. ReLU initialization

Just as the Xavier method, the ReLU weight initialization of [13] aims to prevent the scaling of the variance of the output when forward propagating, and the variance in the weights when backpropagating. This is especially important when working with deep and large network structures. We will do the derivation for a Leaky ReLU network (2.11), because setting \( \alpha = 0 \) will cover the case of the ReLU.

Variance when forward propagating

Recall from Section 2.1 that the weighted input of layer \( l \) with \( d_l \) nodes in a neural network is given by
\[
z^{(l)} = W^{(l)}a^{(l-1)} + b^{(l)},
\]
where \( z^{(l)}, b^{(l)} \in \mathbb{R}^{d_l}, \ a^{(l-1)} \in \mathbb{R}^{d_{l-1}} \) and \( W^{(l)} \in \mathbb{R}^{d_l \times d_{l-1}} \). The activation of this layer was defined as \( a^{(l)} = \sigma(z^{(l)}) \), where \( \sigma \) is the activation function that is applied element-wise.

We start with initializing the biases to zero, \( b^{(l)} = 0 \), and for now take the weights, the entries of \( W^{(l)} \), iid symmetrically around zero. In [13] the assumption is made that the entries of \( a^{(l-1)} \) are also iid and \( a^{(l-1)} \) and \( w^{(l)} \) are independent. Because we are in
the initialization phase and no learning has taken place yet, the second assumption is 
reasonable. However, the independence of the elements in \( a^{(l-1)} \) seems to be invalid in a 
nearl network where every node in layer \( l - 1 \) is connected to every node in layer \( l - 2 \). 
Fortunately, they are identically distributed and because of the zero mean independently 
initiated weights, the elements \( w_{ji}^{(l)} a_i^{(l-1)} \), \( i = 1, \ldots, d_{l-1} \), have zero covariance among 
each other and we can still split the variance of a sum into the sum of variances.

The variance of weighted input \( j \) of layer \( l \) is given by

\[
\text{Var}(z_j^{(l)}) = \text{Var} \left( \sum_{i=1}^{d_{l-1}} w_{ji}^{(l)} a_i^{(l-1)} \right) \\
= \sum_{i=1}^{d_{l-1}} \text{Var}(w_{ji}^{(l)} a_i^{(l-1)}) \\
= d_{l-1} \text{Var}(w_{j1}^{(l)} a_1^{(l-1)}) \\
= d_{l-1} \left( \text{Var}(w_{j1}^{(l)}) \text{Var}(a_1^{(l-1)}) + \text{Var}(w_{j1}^{(l)}) \left( \mathbb{E}a_1^{(l-1)} \right)^2 + \text{Var}(a_1^{(l-1)}) \left( \mathbb{E}w_{j1}^{(l)} \right)^2 \right) \\
= d_{l-1} \text{Var}(w_{j1}^{(l)}) \mathbb{E} \left[ (a_1^{(l-1)})^2 \right] \\
(2.12)
\]

To calculate the expectation of the square of the activation, we need to realize that 
\( a_1^{(l-1)} = \sigma(z_1^{(l-1)}) \). Because the biases initialization at 0 and the zero centered symmetrical 
weights, we see that \( z_1^{(l-1)} \) is also symmetrically distributed around zero. Let us 
denote its density function by \( f_z \). Then by symmetry

\[
\mathbb{E} \left[ (a_1^{(l-1)})^2 \right] = \mathbb{E} \left[ \sigma(z_1^{(l-1)})^2 \right] \\
= \int_{-\infty}^{0} f_z(x) \alpha^2 x^2 \, dx + \int_{0}^{\infty} f_z(x) x^2 \, dx \\
= \frac{1}{2} (1 + \alpha^2) \mathbb{E} \left[ (z_1^{(l-1)})^2 \right] \\
= \frac{1}{2} (1 + \alpha^2) \text{Var}(z_1^{(l-1)}).
\]

Plugging this into equation (2.12) gives

\[
\text{Var}(z_j^{(l)}) = \frac{1}{2} (1 + \alpha^2) d_{l-1} \text{Var}(w_{j1}^{(l)}) \text{Var}(z_1^{(l-1)}). \\
(2.13)
\]

In an artificial neural network with \( L \) layers and linear output function, we therefore 
find that the variance of the output is

\[
\text{Var}(y_j) = \text{Var}(z_j^{(L)}) = \text{Var}(z_1^{(L)}) \prod_{l=2}^{L} \left( \frac{1}{2} (1 + \alpha^2) d_{l-1} \text{Var}(w_{j1}^{(l)}) \right) \\
= d_0 \text{Var}(x_1) \text{Var}(w_{j1}^{(1)}) \prod_{l=2}^{L} \left( \frac{1}{2} (1 + \alpha^2) d_{l-1} \text{Var}(w_{j1}^{(l)}) \right),
\]

50
where \( d_0 \) is the number of inputs of the network. The last equality is because the input layer does not apply any activation function, hence \( z_j^{(1)} \) is just the sum of the weights times network inputs. Because the aim of the initialization method is to make sure that the variance of the input is not magnified or diminished exponentially in the output, the weights should be initialized such that

\[
\frac{1}{2} \alpha^2 d_{l-1} \text{Var}(w_{ij}^{(l)}) = 1, \tag{2.14}
\]

for all \( l, j, i \). The method therefore suggests all weights to be initialized independently from a normal distribution with zero mean and variance \( \frac{2}{d_{l-1}(1 + \alpha^2)} \) to reduce the effect of the number of neurons on the variance of the output. To be precise, we should have chosen the variance for the weights in the first layer to be \( \frac{1}{d_0} \), however, the authors of [13] argue that dividing by a factor \( \frac{1}{2}(1 + \alpha^2) \) will not be a big problem, since it only appears in one of the layers.

**Variance when backpropagating**

To avoid diminishing or magnification of the gradients when doing backpropagation, we argue along the same lines, but deviate a little from [13]. Recall from equation (2.8) that

\[
\delta_j^{(l)} = \sigma'(z_j^{(l)}) \sum_{i=1}^{d_{l+1}} w_{ij}^{(l+1)} \delta_i^{(l+1)}. \tag{2.15}
\]

In our case, \( \sigma'(x) = 1 \) if \( x \geq 0 \) and \( \sigma'(x) = \alpha \) when \( x < 0 \). Again we make assumptions of independence between the weights \( w_{ij}^{(l+1)} \) and \( \delta_j^{(l+1)} \) and \( \sigma'(z_j^{(l)}) \). As in the case of forward propagation, due to the symmetrical mean-zero initialization of the weights, \( z_j^{(l)} \) is symmetrically distributed around 0. This means that \( \mathbb{E}[\sigma'(z_j^{(l)})^2] = \frac{1}{2}(1 + \alpha^2) \), the same factor as showed up before. By the iid assumptions, taking the expectation of equation (2.15) results in

\[
\mathbb{E}[\delta_j^{(l)}] = \mathbb{E}[\sigma'(z_j^{(l)})] \sum_{i=1}^{d_{l+1}} \mathbb{E}[w_{ij}^{(l+1)}] \mathbb{E}[\delta_i^{(l+1)}] = 0.
\]

Therefore, the variance becomes

\[
\text{Var}(\delta_j^{(l)}) = \mathbb{E}[\delta_j^{(l)}]^2 = \mathbb{E}[\sigma'(z_j^{(l)})^2] \mathbb{E}\left( \sum_{i=1}^{d_{l+1}} w_{ij}^{(l+1)} \delta_i^{(l+1)} \right)^2
\]

\[
= \frac{1}{2}(1 + \alpha^2) d_{l+1} \mathbb{E}\left( w_{ij}^{(l+1)} \right)^2 \mathbb{E}[\delta_i^{(l+1)}]^2
\]

\[
= \frac{1}{2}(1 + \alpha^2) d_{l+1} \text{Var}(w_{ij}^{(l+1)}) \text{Var}(\delta_i^{(l+1)}),
\]
a similar kind of iterative expression as in the forward propagation case! Now, for a network with $L$ layers, we find in the first layer

$$\text{Var}(\delta^1_j) = \text{Var}(\delta^{(L)}_1) \prod_{l=2}^{L} \left( \frac{1}{2} (1 + \alpha^2) d_l \text{Var}(w^{(l)}_{ij}) \right),$$  \hspace{1cm} (2.16)

To mitigate effects on the variance of the error due to the number of neurons and layers, we can choose the weights such that

$$\frac{1}{2} (1 + \alpha^2) d_l \text{Var}(w^{(l)}_{ij}) = 1,$$  \hspace{1cm} (2.17)

for all $l, j, i$. The difference between this case and the forward propagation case is the factor $d_l$ instead of $d_{l-1}$. In this case it would result in a normal distribution with mean zero and variance $2/(d_l(1 + \alpha^2))$.

The authors of [13] note that it does not matter much whether one uses equation (2.14) or equation (2.17) to choose the variance of the weights. This is especially true when the layers are designed with the same number of nodes. In case one uses equation (2.14) and initializes the weights from a normal distribution with zero mean and variance $2/(d_{l-1}(1 + \alpha^2))$, the product term in equation (2.16) becomes $d_L/d_1$. Because this term is not dependent on the number of layers in the network, making the network deeper will not have exponential effects on the variance. In [13] it is also empirically verified that in both cases the neural networks converge to the correct target function. In practice, most networks are chosen to initialize according to equation (2.14).

### 2.7. Reduction of overfitting

Because every node in layer $l$ of a neural network is connected to every other node in layer $l-1$, the number of trainable parameters in even a relatively small network is fairly large. To illustrate this, a neural network with four hidden layers with fifty neurons in each of them has a total of 9000 weights and 224 biases that have to be optimized to match the targets on the data. And there are many networks that are much larger. In image recognition tasks, neural networks with 5 to 60 million learnable parameters are not uncommon [37]. This means that there is a serious risk of overfitting if the amount of training data is small. In that case, the network might learn noise in the data instead of just the underlying trends. Such an overfitted network will perform very well on the training data, but poorly on unseen data (with different noise) and is therefore likely to have a big error when used on a new data set. This makes it very important to keep track of possible overfitting during training. When overfitting does occur, the easiest way to combat it, is to reduce the size of the network (less layers and hidden neurons, thus less weights and biases), or to collect more training data. However, sometimes smaller networks fail to capture complex relationships and more data is not available. In these cases, there are many other methods to reduce or signal overfitting. In this section we will only discuss the methods used or considered in the simulations in Chapter 3.
Separating training, validation and test data

Splitting the available data set is probably the method to signal overfitting that is most easy to implement. From the initial data set, roughly two-thirds is used as the training data. The remaining data is split in half: one part validation data, one part test data. The network is trained on the training data. After each epoch, the network will be run on the validation data. Because the network did not train on the validation data, the error on this data set is likely to be bigger than on the training data. However if the difference is too large, it is probable that the model is overfitting the training data. Another indication that overfitting is happening is when the error on the training data decreases, but the error on the validation data starts to increase.

When determining the optimal network structure, there are more parameters to choose than just the weights and biases. These are the so-called meta-parameters. Possible meta-parameters are the number of hidden layers, the number of hidden nodes per layer, the learning rate, the slope on the negative axis for Leaky ReLU activation functions, and so on. Other choices that have to be made when designing a network structure are the activation functions, the output function and the optimization and initialization methods. Because looking at a network’s performance on the validation data and adapting the meta-parameters to increase performance is also a form of ‘learning’, even in this process there is a risk of overfitting. To see if this indeed is the case, the test data was left untouched. Only in the end, after the final network architecture has been selected, the network is run on the test data. This data set should therefore only be used once, and just to signal if there has been ‘manual’ overfitting on the validation data.

A problem with this method of separating data arises when the data set is too small to split the data and train the network properly on the training data. Splitting on small datasets also increases the risk of ending up with a validation or training set that is not representative of the whole data set. When selecting a network model with small datasets, an alternative method is k-fold cross-validation. The dataset is split into k parts of equal size. Then, k − 1 of these sets will be selected and used as training data. The remaining set is used as validation set. This is done k times, each time with a different set used as validation data. The validation error is taken as the average over the k validation errors. This way, all of the data is used to train on and select a network structure. The size of k depends on the application and the amount of data, but typically k is taken to be ten.

Early stopping

Closely related to the method of separating training, test and validation data, is the method of early stopping. Splitting the data will help signal overfitting. Early stopping simply means stopping the training when there are indications that overfitting might occur. This way the network will stop training when it has learned the most general features, but before it starts learning peculiarities or noise in the training data.
Regularization

A different approach to combat overfitting, is weight regularization. This method adds an extra term to the cost function, reducing the size of the weights. A common regularization term is the sum of the squared weights. In the case of cross-entropy, the cost function becomes

\[
C = - \sum_{i=1}^{d_L} t_i \log(y_i) + \frac{\lambda}{2} \sum_w w^2,
\]

where \( \lambda \) is called the regularization parameter. Since the training algorithm aims to minimize the cost function, adding this extra term will force the weights to be smaller. If the weights are smaller, the network output will change less when the input values change. Hence, fluctuations due to noise will have a smaller effect on the output. The algorithm will increase the size of a weight only when it significantly reduces the original cost function. Note that the regularization parameter \( \lambda \) is also a meta-parameter that has to be selected. Often regularization is used together with the other methods discussed in this section.
3. Simulations

The goal of this project was to use artificial neural networks to create a new prepayment model to estimate prepayment probabilities for groups of clients. The first section of this chapter describes what data was used to predict mortgage prepayments and how an optimal network architecture was chosen. In the second section, the performance of the selected model is stated. All calculations were done in Python. To be able to work with artificial neural networks, Google’s open-source software library TensorFlow 0.12 was used.

3.1. Method

We will first describe the data collection and pre-processing. We will explain shortly how to price the prepayment option and then a short description of the model selection is given.

3.1.1. Data collection

The mortgage data used in the simulations is a small selection of the mortgage data from ABN AMRO. Because client data is very sensitive information, clients with similar mortgages are grouped into so-called buckets which are entirely anonymized. Again due to data sensitivity, we only had access to 12,000 buckets to create our ‘portfolio’ when writing this thesis. The mortgage start dates for the buckets range from 1970 to 2016 and the interest reset dates from 2017 up to 2046. Besides the start and end dates, each bucket contains information about the interest rate on the loan (coupon), the amortization type, whether the client is an employee of ABN AMRO or not, if the mortgage was ABN AMRO main brand or not, if the mortgage has a reconsider option, the length of the fixed interest period (term) and the amount of money of all clients in the buckets combined (face amount). The face amount was not used to train the network, only to price the prepayment option (Subsection 3.1.3).

The components of the input data concerning the classification of a bucket into more than two classes, being amortization type (bullet, level, linear) and season, were presented in so-called ‘one-hot encoding’. For the amortization type, this means that it is transformed into an array of length three, with a one on the location of its type and zeros elsewhere. For example for a bullet bucket, the amortization type is [1, 0, 0]. If the classes were given as a scalar with different values, say 1=bullet, 2=level, 3=linear, then level would be the average of bullet and linear, which is not the case. To remove these wrong ‘closeness’ implications in the data, one-hot encoding is used. Every element of the
amortization type and season has its own input neuron, and the network can therefore be trained without wrong preconceptions in the data. This brings the total number of input variables to 24: ten explanatory variables from which two in one-hot encoding and an indicator such that the network knows when a mortgage has a reconsider option. To calculate the HPI ratio, the HPI for each month from 1995 to 2016 in the Netherlands was taken from Statistics Netherlands (CBS)\(^1\). For the 20 buckets with start dates before 1995, the HPI of January 1995 was used to calculate the HPI ratio. From 2012 onward, the ABN AMRO HPI forecast was used. This forecasts predicts the HPI for each month up to the end of 2044.

To be able to progress the portfolio through time and calculate the interest incentives, short rate scenarios were run. As mentioned in Section 1.3, the mortgage rates are swap rates plus a service spread. To find the size of the service spread, we took the current mortgage rates from the ABN AMRO website and compared them to the ICE swap rates\(^2\) with the same maturities. The mortgage maturities offered by ABN AMRO are 1, 2, 3, 5, 6, 7, 10, 12, 15, 17, 20, 25 and 30 years.

The Nelson-Siegel-Svensson parameters were taken from the ECB website\(^3\) for 1 December 2016, from which the initial instantaneous forward curve and the zero coupon bond prices were constructed as described in Subsection 1.3.3. The one-factor Hull-White model was then chosen to match the initial term structure in the way described in Subsection 1.3.2.

As a sanity check for the simulated short rates, a large amount of sample paths was generated. As described in Subsection 1.3.1 about the Hull-White model, the short rate is normally distributed with mean and variance at time \(t\) given by

\[
E[r(t)] = \alpha(t), \quad \text{Var}(r(t)) = \frac{\sigma^2}{2a} \left(1 - e^{-2at}\right).
\]

It turns out that the sample mean and sample variance are very close to the theoretical mean and variance in all combinations of parameters tried. Plotting the empirical distribution function against the theoretical cumulative distribution function (Matlab \texttt{normplot}), Figure 3.1, also shows normality of the generated sample paths.

Confident that the generated short rate sample paths are correct, we generated 100 sample paths of monthly swap rates up to 10 years in the future, with maturities that are offered by ABN AMRO for mortgage loans. Using these and the HPI forecast, more data was generated from the initial sample by simply progressing the portfolio through time with time steps of 7 months on each sample path (thus updating the remaining interest period, interest incentive, loan age, penalty proxy, HPI ratio and season each time step). We chose time steps of 7 months to keep the size of the data manageable and make sure every month appears in the sampled data, which is important, since the season is an explanatory variable. To reduce the amount of samples that are very alike,

\[^1\text{Base year 2010, } \text{http://statline.cbs.nl/Statweb/?LA=en}\]
\[^2\text{https://www.theice.com/iba/historical-data}\]
we only used the last 5 years of the generated data, such that the swap sample paths had time to move away from each other.

This way in total 2,567,100 data points was created. To generate the corresponding targets, the traditional multinomial prepayment model was used (Section 1.2). The thus generated data was split into a training, validation and test dataset, as described in Section 2.7. Around two-thirds was selected as training data. One-sixth was used as validation data to select meta-parameters (see Subsection 3.1.2). This resulted in a training dataset of 1,711,100 samples and a validation set of 427,600 samples. The remaining one-sixth of the data was left untouched until the model had been selected and was just used to test the final model on. The results on this test data set, consisting of 428,400 samples, can be found in Section 3.2.

There were relatively few mortgages with a reconsider option in the portfolio, and many of the reconsider options were not executed. This prepayment class was therefore underrepresented in the training data, making it difficult for a neural network to predict the prepayment ratio for this class correctly. To combat this, extra training data can be generated using a synthetic minority over-sampling technique (SMOTE, [5]). In short, this algorithm selects a random fraction of the mortgages with the reconsider option. For each of the selected mortgages, the method finds the $k$ nearest neighbours that also have the reconsider option in the space of explanatory variables. From those nearest

![Normal Probability Plot](image)

Figure 3.1.: The empirical distribution function of generated short rates at $t = 397$ months with parameters $a = 0.02$, $\sigma = 0.001$. The diagonal line is the theoretical value of the cumulative distribution function.
neighbours, a sample of size $N$ is drawn. For each of these $N$ selected points, a new reconsider sample is created with explanatory variables equal to a random spot on the line between the original point and the selected neighbour.

It turned out that this was beneficial for the performance of the neural network when only a subset of the data was used to train on. When training on a set of 2400 buckets and the data points corresponding to the time evolution on 20 short rate sample paths, using the SMOTE algorithm on 50% of the reconsider mortgages with $k = 1$ and $N = 1$ improved the prediction performance of the reconsider class. This smaller dataset was used to roughly select the form of the final model.

In the final model with the full data set derived from all 12 000 buckets, the network performed best without SMOTE data. Apparently there were already sufficient reconsider options in the larger dataset to take this class into account.

The magnitude of the different components of the data was very different. For example, the interest period start data is in the MATLAB date format, which is a number of the order $10^5$, while interest incentive is a percentage. To prevent the network from focusing the learning process on the very large input values, it is recommended to normalize the different inputs (e.g. in [11]). Therefore, we calculated the mean and standard deviation in the training set of each input type. Then, for the training, validation and test data, we used these to normalize the inputs. It is important to use the training data mean and standard deviation to scale the test and validation data, because the validation data, test data and possible real-life datasets when applying the model may not contain the same kind of samples in the same amount as the training data. After scaling, all input types are of similar size such that the network is not biased towards the input with the largest magnitude.

3.1.2. Model selection

Since the network aims to predict the prepayment probabilities over 6 classes, the network needs to have 6 output neurons. The total number of different inputs per buckets is 24, so the network has 24 input neurons. We first select the output function and cost function. Then number of hidden layers and the number of neurons per layer. After that, the optimal activation function and learning rate are chosen. All decisions involved a lot of trial and error on the smaller data set with 2400 buckets (with extra SMOTE reconsider buckets). On this smaller set, the most promising network structures were selected. It is convenient to do this on the smaller set, because of the smaller training time (approximately 15 minutes). However, we can only judge the performance of any network structure on the full data set. On a small dataset, it is difficult to measure the performance of larger networks, because they are more likely to be overfitted. Therefore, to select the final model, the most promising candidates were also tried on the full dataset of 12 000 buckets. Of course, the networks were trained on the training data and performance was measured on the validation data. Training on this large dataset took slightly less than two hours, but the exact training time depended on the size of the network. The test data was left untouched. Surprisingly, on the full dataset there were no signs of overfitting. Therefore, we did not apply regularization by penalizing
large weights (as described in Section 2.7).

Output and cost function

Even though estimating the prepayment probabilities per prepayment class is in fact a regression problem, we noticed that using the cross-entropy cost function in combination with softmax output function performed significantly better than minimizing the mean squared error (both with linear output as with softmax output function). This is most likely due to the fact that there is a big difference in magnitude of the six prepayment classes. By the nature of the MSE, minimizing it will focus on absolute differences between the network output and the targets,

\[ \frac{\partial}{\partial y_j} \left( \frac{1}{2} \| y - t \|^2 \right) = y_j - t_j. \]

This means that a big difference in magnitude of the output neurons will lead to less optimization to minimize the error of the smaller output nodes. In our case, for example, the prepayment probabilities for the category relocation are roughly between $10^{-4}$ and $10^{-2}$, while the magnitude of the reconsider probability may exceed 0.15. This difference makes it difficult for the MSE to converge on the relocation class.

On the other hand, the cross-entropy error will take into account the relative error,

\[ \frac{\partial}{\partial y_j} \left( \sum_{i=1}^{d_L} t_i \log(y_i) \right) = \frac{t_i}{y_j}, \]

thus paying more attention to the smaller output neurons. In an attempt to fix this problem for the MSE, we tried dividing each component of the squared error by the average of the target $\bar{t}_i$, resulting in a derivative of

\[ \frac{\partial}{\partial y_j} \left( \frac{1}{2} \sum_{i=1}^{d_L} \frac{(y_i - t_i)^2}{t_i} \right) = \frac{y_i - t_i}{t_i}. \]

Minimizing this weighted MSE outperformed the normal MSE in predicting the smaller prepayment classes correctly. However, because the cross-entropy error still resulted in faster convergence we chose to work with cross-entropy and softmax output. However, we only used the cross-entropy error in the optimization. To document the prediction error, we will still use the MSE and also look at the relative error per prepayment class, because it is more insightful.

Optimization method and number of hidden layers and hidden neurons

On the smaller data set, it was immediately clear that the Adam optimization method, discussed in Subsection 2.3.3, yielded the best network performance. We therefore chose to work with this optimizer. Even though the article introducing Adam is still in preprint, this optimization method is already widely used throughout the field of neural networks.
The networks that showed the most promising results on the smaller dataset, consisted of either 1 hidden layer with around 125 neurons or 4 hidden layers with 50 neurons each. Therefore, we explored several network architectures close to these two shapes on the large dataset. In the tests, a Leaky ReLU activation function ($\alpha = 0.2$), softmax output and CE cost function were used. The number of epochs may vary due to early stopping (Section 2.7), but was set to 200 if no stopping occurred. To reduce the impact of the initialization and random effects, the average over 5 training sessions is taken. Some of the results are listed in Table 3.1.

<table>
<thead>
<tr>
<th>Results on validation data</th>
<th>hidden layers x nodes</th>
<th>MSE ($\times 10^{-8}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1x125</td>
<td>5.683</td>
</tr>
<tr>
<td></td>
<td>1x130</td>
<td>7.615</td>
</tr>
<tr>
<td></td>
<td>4x50</td>
<td>2.708</td>
</tr>
<tr>
<td></td>
<td>4x60</td>
<td>23.89</td>
</tr>
<tr>
<td></td>
<td>4x100</td>
<td>2.327</td>
</tr>
<tr>
<td></td>
<td>5x50</td>
<td>5.929</td>
</tr>
<tr>
<td></td>
<td>7x50</td>
<td>5.204</td>
</tr>
<tr>
<td></td>
<td>10x35</td>
<td>8.085</td>
</tr>
</tbody>
</table>

Table 3.1.: The MSE on the validation data of the full data set for networks with Leaky ReLU activation function, learning rate 0.001, softmax output and CE cost function, averaged over 5 training sessions.

**Activation function**

To find the best performing activation function, we used a network with 4 hidden layers of 50 hidden neurons each. Some results on the validation data are listed in Table 3.2.

<table>
<thead>
<tr>
<th>Results on validation data</th>
<th>Activation function</th>
<th>MSE ($\times 10^{-8}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LReLU $\alpha = 0.01$</td>
<td>4.219</td>
</tr>
<tr>
<td></td>
<td>LReLU $\alpha = 0.1$</td>
<td>9.345</td>
</tr>
<tr>
<td></td>
<td>LReLU $\alpha = 0.2$</td>
<td>2.708</td>
</tr>
<tr>
<td></td>
<td>LReLU $\alpha = 0.3$</td>
<td>3.483</td>
</tr>
<tr>
<td></td>
<td>PReLU</td>
<td>3.896</td>
</tr>
<tr>
<td></td>
<td>ReLU</td>
<td>5.780</td>
</tr>
<tr>
<td></td>
<td>ELU</td>
<td>9.100</td>
</tr>
</tbody>
</table>

Table 3.2.: The MSE on the validation data of the full data set for networks with 4 hidden layers with 50 nodes, learning rate 0.001, softmax output and CE cost function, averaged over 5 training sessions.
Other meta-parameters

There are many other parameters that can be varied to influence the performance of a neural network. Several tests with the exponential decay rates from the Adam optimizer (Subsection 2.3.3), showed that the optimization was best with the default parameter values of $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\varepsilon = 10^{-8}$. However, we did note that a lower learning rate could lead to less fluctuations in the MSE during training. These fluctuations are caused by the random shuffling of mini-batches at each epoch. A smaller learning rate will decrease the step size in each iteration and therefore smoothen the learning process. Of course, the size of the mini-batches is also important. We noticed that a mini-batch size of 1000 worked well. Smaller mini-batch sizes lead to a lot of fluctuations in the prediction error and larger sizes resulted in slower improvements. We did several tests varying the learning rate, with a network of 4 hidden layers with 50 neurons each. The results are listed in Table 3.3.

<table>
<thead>
<tr>
<th>Learning rate</th>
<th>MSE ($\times 10^{-8}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0001</td>
<td>4.073</td>
</tr>
<tr>
<td>0.001</td>
<td>2.708</td>
</tr>
<tr>
<td>0.01</td>
<td>8.648</td>
</tr>
</tbody>
</table>

Table 3.3.: The MSE on the validation data of the full data set for networks with 4 hidden layers with 50 nodes, Leaky ReLU activation function, softmax output en CE cost function, averaged over 5 training sessions.

Selection of the final network

From Table 3.3, it becomes clear that a learning rate of 0.001 outperformed other learning rates. In Table 3.2 we see that a Leaky ReLU with $\alpha = 0.2$ was the best choice for the activation function. When varying the number of hidden layers and hidden neurons in Table 3.1, we see that two networks showed better results than the others. Because the MSE of the network with 4 hidden layers with 50 neurons and the network with 4 hidden layers with 100 neurons were comparable, we decided to try both network structures. For each network shape, we trained 5 networks and selected the best performing network on the validation data as our final model. This turned out to be a network with 50 hidden neurons per layer. The MSE of the selected network was $1.029 \times 10^{-8}$ on the validation data. Section 3.2 lists more results and plots of the output of the selected network on the validation and test data.

3.1.3. Pricing the prepayment option

In the end, the new prepayment model based on artificial neural networks will be used to price the prepayment option on a mortgage. Simply put, this is the value of having the possibility to prepay. This is an important quantity, because it needs to be hedged and
is used in the calculations of the penalty on the prepayment classes curtailment high and refinancing. We can calculate the price of the prepayment option by calculating the difference between the value of the interest payments (coupon times face amount at the specific time) on a bucket with prepayments and without prepayments. All future interest payments have to be discounted to determine the present value of the mortgage loan with and without prepayments.

To quantify the performance of the neural network compared to the multinomial prepayment model, we calculate the value of the interest payments for mortgage loans without prepayment, with prepayment as predicted by the multinomial model and as predicted by the neural network. To do this, scenarios are generated as described in Subsection 3.1.1. Portfolios are progressed through time and the discounted interest cash flows are calculated for the three cases. Because the outstanding loan in a bucket is gigantic and also differs per bucket, we will look at the price of the prepayment option relative to the value of the mortgage without prepayments.

3.2. Results

In this section we will state and discuss the results of the model selected in Subsection 3.1.2 on the test data. This dataset has not been used in any way in earlier stages. The expectation is therefore that the network will perform slightly worse on this data set compared to its performance on the validation data.

The best performing network achieved a mean squared error on the validation data of $1.029 \times 10^{-8}$, while the error on the training data was $9.333 \times 10^{-9}$. Plotting the MSE on both training and validation data during training on a logarithmic scale, results in Figure 3.2.

As we can see, the error fluctuates a lot due to the stochastic shuffling of mini-batches. However, by applying the early stopping method we are able to stop training when the MSE is at a minimum. The large fluctuations could point to a mini-batch size that is too small or a learning rate that is too large. However as the experiments in Subsection 3.1.2 describe, varying these did not improve the approximations. Another way to track the effectivity of a network is to look at the values of the activations per layer. If too many activations are negative, the network will learn slowly from that point on due to the smaller gradient of the Leaky ReLU activation function in the negative domain. Histograms of the average activations per neuron in each layer are shown in Figure 3.3.

As we can see, the first two layers show many activations in the positive domain, thus the learnable part of the Leaky ReLU. In the last two layers, activations tend to accumulate around 0. This is not optimal, but because we are using a Leaky ReLU neuron, it is not a real obstacle if we want to continue learning, since we chose the negative slope to be $\alpha = 0.2$; sufficient to recover from dying neurons.

To visualize the performance of the network on the validation data, we can plot the network predictions for a couple of buckets and compare them to the targets. This is depicted in Figure 3.4.

These plots show that the prepayment probabilities are predicted well in most cases.
Figure 3.2.: The MSE on logarithmic scale of the prepayment predictions of a neural network with 4 hidden layers with 50 hidden neurons each, Leaky ReLU ($\alpha = 0.2$) activation function and softmax output. The green line is the MSE on the training data, the blue line on the validation data.

This also shows from the relative error per prepayment class on the validation data, presented in Table 3.4.

Another way to visualize the results is by using scatter plots. This is done in Figure 3.5. As we can see, the predictions of the neural network agree very well with the targets. Especially if taken into account that there are 1 711 100 data points in the training data plot and 427 600 in the validation data plot.

Finally we are allowed to use the test data. It turned out that the model did indeed perform worse on the test data, having a MSE of $3.995 \times 10^{-8}$. The relative errors per prepayment class are given in Table 3.5.

As we can see, the average relative errors per prepayment class are the same as those on the validation data. The scatter plot on the test data is shown in Figure 3.6.

As we can see, there are slightly more outliers in the scatter plot of the test data, but again, taking into account that there are 428 400 data points in the scatter plot, the
result is satisfying since the neural network seems to be able to capture the general relation between the explanatory variables and the target prepayment predictions. When using the model to calculate the price of the prepayment options on mortgages, we generated another 200 short rate sample paths and evaluated 200 buckets. We found an average difference of 0.42% compared to the prepayment options priced with the multinomial model.
Table 3.4.: The average relative error per prepayment class of the neural network on the validation data.

<table>
<thead>
<tr>
<th>prepayment class</th>
<th>average relative error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>refinancing</td>
<td>0.51%</td>
</tr>
<tr>
<td>relocation</td>
<td>0.58%</td>
</tr>
<tr>
<td>reconsider</td>
<td>1.00%</td>
</tr>
<tr>
<td>curt. high</td>
<td>0.56%</td>
</tr>
<tr>
<td>curt. low</td>
<td>1.02%</td>
</tr>
<tr>
<td>no event</td>
<td>0.0098%</td>
</tr>
</tbody>
</table>

Table 3.5.: The average relative error per prepayment class of the neural network on the test data.

<table>
<thead>
<tr>
<th>prepayment class</th>
<th>average relative error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>refinancing</td>
<td>0.51%</td>
</tr>
<tr>
<td>relocation</td>
<td>0.58%</td>
</tr>
<tr>
<td>reconsider</td>
<td>1.00%</td>
</tr>
<tr>
<td>curt. high</td>
<td>0.56%</td>
</tr>
<tr>
<td>curt. low</td>
<td>1.02%</td>
</tr>
<tr>
<td>no event</td>
<td>0.0098%</td>
</tr>
</tbody>
</table>

The estimated prepayment probabilities of the neural network compared to the targets

Figure 3.4.: Four plots of the predicted prepayment probability per prepayment class (blue) compared to the target prepayment probabilities (red) on random buckets. On the y-axis we see the prepayment probabilities, on the x-axis the prepayment categories number as numbered in Section 1.1 (category 6: no event is left out since it is determined by the other 5 and is a lot larger than the other probabilities). The lines coincide almost perfectly.
Figure 3.5.: Scatter plots of the predictions per class of the neural network ($x$-axis) and the target prepayments ($y$-axis). The top figure shows the scatter plot for the training data (1 711 100 points), the bottom figure for the validation data (427 600 points).
Figure 3.6.: Scatter plot of the predictions per class of the neural network ($x$-axis) and the target prepayments ($y$-axis) on the test data (428 400 points).
4. Conclusion

It has become clear from the results in Section 3.2, that the selected artificial neural network with 4 hidden layers of 50 neurons each was able to capture the underlying relation between the explanatory variables and the generated prepayment targets. This does vouch for the use of these kind of networks in prepayment modelling. However, we must note that the errors were still large.

We stress the importance of using enough data during the training phase of a network, as the errors might have been smaller if a larger number of different short rate sample paths had been present in the training data. Another important thing to keep in mind is that if a network is supposed to perform in extreme situations, also a number of data points in this extreme domain should be added to the training data. This is vital, because networks are poor extrapolators and it is especially difficult to predict their outcome on data that is very different from the data they were trained on. One option that could be considered is using a neural network based model in combination with a traditional multinomial prepayment model. The neural network will function on the normal domain, but in extreme scenarios that were not included in the training data the multinomial model could take over. This also has the benefit that it is easier to understand the influence of input data on the output of the multinomial model.

Although the estimation errors on the prepayment probabilities look small, even a small error can have big consequences because very large amounts of money are involved. For that reason, it is important to keep improving the model and keep training it on large and recent datasets. For some other suggested directions of further research to improve the models performance, see Section 5.2.

Unfortunately, we did not have access to the real prepayment data. This means that we cannot draw any conclusions about the performance and applicability of the neural network based prepayment model on the real dataset. However, the estimation of the prepayment probabilities was accurate on the generated targets. This is promising. The actual relation between the explanatory variables and the prepayment probabilities is likely to be a lot more complex than the one described by the multinomial prepayment model. Because of the theoretical capacities of artificial neural networks to approximate very complex functions it is likely that neural networks have the potential to be better predictors of prepayment than the current multinomial prepayment model. Especially because large amounts of prepayment data have been collected and could be used for training in the future. Because of this potential of a model based on a neural network, we encourage further research in its applications to this and other models within ABN AMRO.

We will expatiate on these recommendations in Section 5.1.
5. Discussion and Further Research

In the first section of this chapter, we discuss the general and problem specific advantages and disadvantages of the use of neural networks in risk modelling. Section 5.2 suggests some improvements to the selected model that might be promising. In Section 5.3 we will shortly review the process of writing this thesis.

5.1. Advantages and disadvantages of artificial neural networks

There are several advantages and disadvantages in the use of artificial neural networks. Below, we will summarize a few of them.

Data-driven

One of the main characteristics of artificial neural networks is that they will learn solely from the data they are trained on. As we have shown in Section 2.2, they are universal approximators and therefore, if a function describing the relation between the input data and the targets exists, they are theoretically capable of approximating this function. To be able to train well, large quantities of data are necessary. This can be a restriction, but in this case of mortgage prepayments it is not. The selection of 12,000 buckets available in writing this thesis, is just a tiny fraction of the total amount of data on mortgages. During training, the network may discover relations that the modeller is not yet aware of and no restrictive assumptions on the data are necessary to be convinced that a neural network can capture the sought after relation. One only has to select a network architecture and have large amounts of data. The training will take care of the rest. Once trained, the network will be immediately applicable to new situations. However, it should be stressed that a neural network can only learn what it is shown. To be able to handle extreme situations such as stress tests, it should be trained on data with explanatory variables in this extreme domain. If there is no training data available in a certain domain, the network cannot be expected to extrapolate well as it is simply not taught how to do so.

That the network learns what it is shown, is also something one has to be careful with for another reason. Especially if neural network output is used to make decisions about (individual) clients, it is very important to consider what kind of relations the network is allowed to find. If you feed it data including the ethnicity of clients, the network is very likely to include this data in its decisions and might develop racist tendencies. In this case it is obviously not right to include ethnicity, but sometimes other apparently more
harmless input data may lead the network to similar results. If for example the address of the client is used as an input parameter, the network might favour one neighbourhood over the other. If one of these neighbourhoods has a high concentration of specific ethnic group, the results are virtually the same. Accidentally racist (or sexist) algorithms appeared before with purely data-driven algorithms (e.g. [22]), and should serve as a warning. Think carefully about what explanatory variables you want the network to base its decisions on and be aware of the possible consequences of feeding all available data to a neural network and let it figure out what to do by itself.

Rapidly developing field of research

New articles suggesting improvements on the shape of neural networks, the activation function, optimization method, weight initialization method and so on, seem to appear almost every day. Considering the achievements already made with the current techniques, this is an indication that the performance of neural networks will increase in the coming years. That is, if one keeps up with all the developments. In a field of research that moves this fast, the state-of-the-art models from this year may already be outdated next year. This will force the modeller to constantly stay up to date with the most recent articles and try the new methods in his models. This can make working with neural networks time consuming, despite the fact that the concept is easily understood. However, we strongly recommend doing so. Using the ReLU weight initialization [13], introduced in 2015, and the Adam optimizer [16], introduced in 2017, greatly improved the performance of our neural network compared to older methods.

Other observations

When working with neural networks in risk modelling, several other factors have to be taken into account. One of those is that a network needs to be retrained regularly as new data becomes available. The financial markets and client behaviour are dynamical, and their behaviour may change from year to year. Recent data should be added to the training data and old irrelevant data (e.g. through regulatory changes) should be removed.

Another property of neural networks that became clear in doing this research, is that there is some randomness in how well a network learns. Performance may vary even when training on the same data and using the same validation data. To mitigate the variance in network results, an easy solution is to train several networks and pick the one that performs best on the validation data. The selected network should then be tested on the test data to see whether it performs accordingly and did not just by chance match the validation data well. Another procedure one can consider is called ensemble averaging ([12]). In this method, multiple neural networks are trained on the same dataset. Then, instead of using only one of the networks, a weighted average of the outputs is used. The weights are determined by the performance of the individual networks on the validation data (or can even be optimized using another neural network). In this thesis, we chose to work with just one neural network. However, each of the discarded networks may
have learned a different aspect of the training data, so if the estimation problem is more complex or if there are different classes of input data, ensemble averaging may be worth investigating. The last thing we want to touch upon in this section is what some call the ‘black box’ property of neural networks. Input data goes in and the network gives output, but what happens in between is very difficult to decipher. This is because of the highly non-linear form of the transformation of input data and the large amounts of weights and biases (in our relatively small four hidden-layer network already 9224). This makes it difficult to track the impact of changing one of the input variables, meaning that a neural network trained for a specific problem can in most cases not tell the user general rules regarding its decision-making. One could evaluate the output of the network on a surface of explanatory variables to visualize the impact of different inputs, but in high dimensional settings this is not very insightful. The only way to comfort the user (and possible regulators) is to show the networks performance on test datasets.

5.2. Further research

In this section we will describe some possible improvements of the selected model, which we were not able to try due to time constraints. Because the data used to price the prepayment option was newly generated and therefore unseen by the network, the error on the prepayment option might have been reduced by a larger number of different short rate sample paths in the training data. This could have reduced the probability of ‘surprising’ sample paths in the pricing of the option. This stresses the importance of a large training data set, so that the network is even more able to capture the underlying general relation between explanatory variables and the prepayment probabilities. We are also aware of the fact that we trained the network on predicting the prepayment probabilities, and not on predicting the correct value of the prepayment option. Although these two are closely related, it would be interesting to see if a network is able to directly price the prepayment option, without first calculating the individual prepayment probabilities per category.

Another observation when looking at the results in Section 3.2, is that we barely see signs of overfitting. This is good, but it is also an indication that even larger networks (if necessary with regularization) can be applied to this problem. However, trying larger networks did not yield better approximation results with our dataset. This could either mean that the general relation between input and output is already captured by this network structure or that there is insufficient data to try larger networks. Whenever the training dataset will be taken bigger (which will be the case when working with the real prepayment data), we therefore encourage experimenting with larger network structures.

5.3. Review of the process

After choosing the subject of this research at ABN AMRO, I had a very pleasant time working there and being part of the team. I also got the possibility of doing the large
computations necessary to train neural networks on their server. I had a lot of freedom to determine my own working schedule and working hours. My daily supervisor, Bastian Wemmenhove, helped me to determine the direction of this research during our meetings and would point me in the right direction when I needed it. Therefore, I would like to thank ABN AMRO and Bastian in particular for the support and encouragements during this research. I also want to thank Peter Spreij for his feedback on earlier versions of this thesis. Below I will reflect on some of the difficulties I encountered in the process of writing and doing research.

One of the greatest difficulties was the unavailability of data. As mentioned in Subsection 3.1.1, client data is difficult to get by. Even when anonymized and grouped in buckets, it was not possible to get the real prepayment data for this research. Therefore, targets had to be created using the existing multinomial prepayment model. This means that the results only reflect the networks ability to replicate the other model. Even though it is encouraging that it could to a reasonable extent, before we can draw any conclusions about the applicability of this model, it should be trained and tested on real prepayment data.

Another challenge in writing this thesis was that I was forced to use an older version of TensorFlow (r0.12). Because this is such a rapidly evolving field, it is very useful to have the most recent version of the software (this currently is r1.12). Because of the older software, a functionality like early stopping was difficult to implement and saving trained networks was not possible at all due to a bug. Luckily this has been fixed in newer versions of TensorFlow and I recommend updating the version at ABN AMRO to facilitate further research in this topic.

Because of the many advantages and the potential a model based on a neural network has, I encourage further research in the application of these networks to this and other models within ABN AMRO. Hopefully this thesis can function as a guide to developing a new prepayment model and help modellers get up to speed in the development of other neural network models. With the large quantities of data the bank has collected, artificial neural networks definitively have the potential to add value to risk modelling.
Popular Summary

In this master thesis, we try to predict part of the risk for a bank when writing out a mortgage loan. When a customer takes a mortgage loan of thirty year, but decides to pay it back after only ten, the bank loses twenty years of interest payments. If you realize that ABN AMRO has billions of euros of outstanding mortgage loans, it is not hard to see why it is important for them to predict how many customers are going to pay back their loan before the end of the contract. There are many parameters that can influence these so-called mortgage prepayments. Among others, think of the difference between the interest rate the client is paying and the interest rate he can get on a new mortgage, the length of the contract or even whether the client is an employee of ABN AMRO or not. In our research, we take into account 24 different parameters influencing the probability of prepayment, among which the three mentioned above. We assume there exists a function $f$ that maps the data of the loan to the probability of prepayment. Because this is a real-world problem, we do not know what this function looks like or how complicated it is (it may even change each year!), but we do want to approximate it. This is where the neural networks come into play.

Artificial neural networks are booming. A lot of impressive recent developments, ranging from self-driving cars to Apple's Siri, are thanks to these networks. How these artificial neural networks work is actually not so difficult to understand. Artificial neural networks are used as approximators. They can for example approximate the function taking an image of a handwritten number as input and giving the number it represents as output. Or in this thesis, the function taking the mortgage prepayment parameters as input and the prepayment probability as output.

In Figure 2 below we see a schematic representation a neural network. The input enters from the left, and the signal flows to the right. Each node in the network takes a weighted sum of all the input it receives, applies a certain function to it and sends it on to all neurons in the next layer. The key to making this neural network approximate the target function, is to make it ‘learn’ the correct weights. It gets to see a lot of input values and makes predictions. If the prediction was incorrect, all weights are changed a little in the direction that will make the network give a better prediction next time. This way, the network learns by making mistakes. Much like our own brain! You can imagine that big neural networks with many layers and neurons are able to capture very complicated structures in the data. Indeed, it turns out that they are very good at recognizing patterns in data. In this thesis we explore how we can use this potential to predict prepayment behaviour.
Figure 2.: A schematic representation of an artificial neural network.
Bibliography


Appendices
A. Additional Theorems and Definitions

In this Appendix we will state theorems (if insightful with proofs) and definitions that are referenced throughout the text.

A.1. Theorems for Chapter 1

This section will state results that are needed in Chapter 1. We will start with a recap mainly summarizing Chapter 2 in Filipović ([9]) and Chapter 1 of Brigo and Mercurio ([3]). Throughout this section, we assume $(\Omega, \mathcal{F}, \mathbb{P})$ to be a probability space with filtration $\mathcal{F} = \{\mathcal{F}_t\}_{t \geq 0}$ satisfying the usual conditions.

Bank account, short rate, discount factor, zero-coupon bond

The most basic definition in interest rates is the bank account. The bank account or money-market account satisfies the differential equation

$$dB(t) = r(t)B(t)\,dt, \quad B(0) = 1,$$

such that

$$B(t) = e^{\int_0^t r(s)\,ds}.$$

Hence $B(t)$ changes risk-free with the (stochastic) short rate $r(t)$. The money-market account is used to define the discount factor, determining the value at time $t$ of money available at a time $T > t$. To discount an amount at time $T$ to its value at time $t$ we multiply by the discount factor

$$D(t,T) = \frac{B(t)}{B(T)} = e^{-\int_t^T r(s)\,ds}.$$

A definition closely related to the discount factor is that of the zero-coupon bond (to stress the maturity time $T$ of a zero-coupon bond, it is also called a $T$-bond). A zero-coupon bond with maturity time $T$ is a contract that pays 1 unit of currency at time $T$. The price of this contract at time $t < T$ is denoted by $P(t,T)$. The difference between the discount factor $D(t,T)$ and the price of a zero-coupon bond $P(t,T)$ is that $P(t,T)$ has to be known at time $t$, since it is the price of a product, hence $P(t,T)$ is $\mathcal{F}_T$-measurable. While $D(t,T)$ is $\mathcal{F}_T$-measurable, since the future values of the short rate process $r$ are not known at time $t$.

If we also assume there exists an equivalent martingale measure $\mathbb{Q}$, meaning that $\mathbb{Q} \sim \mathbb{P}$ and all discounted price processes are martingales under $\mathbb{Q}$, the market is arbitrage free.
(Section 4.3) and the relation between the discount factor and the zero-coupon bond price becomes
\[ P(t, T) = \mathbb{E}_Q \left[ e^{-\int_t^T r(s) \, ds} \mid \mathcal{F}_t \right]. \] (A.1)

**Forward rate agreement and derived rates**

Another basic financial product is the so-called *forward rate agreement (FRA)*. At time \( t \), the buyer of this contract sells one \( T \)-bond and buys an amount of \( \frac{P(t, T)}{P(t, S)} \) \( S \)-bonds. The net investment is therefore 0. At time \( T \) he has to pay 1 unit of currency according to the sold \( T \)-bond. At time \( S \) he receives \( \frac{P(t, T)}{P(t, S)} \) units of currency.

The definition of this instruments leads to the definition of the simply compounded and the continuously compounded forward rate. The *simply compounded forward rate* is defined as
\[ F(t; T, S) = \frac{1}{S-T} \left( \frac{P(t, T)}{P(t, S)} - 1 \right). \] (A.2)

This is exactly the simple interest rate which makes a forward rate agreement fair. If we invest one unit of currency at time \( T \) and we receive simple interest on the investment, at time \( S \) we have
\[ 1 + (S - T)F(t; T, S) = \frac{P(t, T)}{P(t, S)}, \]
exactly the cash flow of the FRA. Simply compounded forward rates are widely used in finance. An important example of a simple forward rate is the London Interbank Offered Rate, in short LIBOR, which is the rate at which banks borrow money from each other and the rate at which swaps between banks are traded.

If we use continuously compounded interest rates, the rate to make the forward rate agreement fair is the rate \( R(t; T, S) \) such that
\[ e^{R(t; T, S)(S-T)} = \frac{P(t, T)}{P(t, S)}, \]
leading to the definition of the *continuously compounded forward rate*:
\[ R(t; T, S) = -\log \frac{P(t, S)}{S-T}. \]

If we remove the ‘forward’ from this rate, we get the definition of the *continuously compounded spot rate* for the time interval \([t, T]e:
\[ R(t, T) = R(t; t, T) = -\log \frac{P(t, T)}{T-t}. \] (A.3)

Also taking \( T \downarrow t \) yields the *short rate*:
\[ r(t) = \lim_{T \downarrow t} R(t, T). \]
On the other hand, letting the time interval of the continuously compounded forward rate go to zero gives the *instantaneous forward rate* (also often called forward rate):

\[ f(t, T) = \lim_{S \downarrow T} R(t; T, S) = -\frac{\partial \log P(t, T)}{\partial T}. \] (A.4)

Note that this also means that \( r(t) = f(t, t) \) and \( P(t, T) = e^{-\int_t^T f(t, u) \, du}. \)

**Interest rate swaps**

An interest rate swap is a contract that makes it possible for the owner to pay a fixed interest rate and receive a floating rate or vice versa. Transactions take place at future dates \( T_0 < T_1 < \ldots < T_n \), where often the assumption of equidistant time points is made, such that \( T_i - T_{i-1} = \delta \).

A *payer interest rate swap* is an agreement in which the holder of the swap pays a fixed rate \( K \) over an agreed amount \( N \), called the principal, and receives a floating rate \( F \) over the principal. The floating rate is a forward rate often taken as the LIBOR rate. Thus at each payment date \( T_i \), a fixed amount (fixed coupon) \( \delta NK \) has to be paid, called the fixed leg of the swap. A floating rate \( \delta NF(T_{i-1}, T_i) \) is received, where \( F(T_{i-1}, T_i) = F(T_{i-1}, T_{i-1}, T_i) \) as defined in equation (A.2). This cash flow is called the floating leg of the swap. Note that the floating rates \( F(T_{i-1}, T_i) \) are known and fixed at time \( T_{i-1} \) but the floating leg is received at time \( T_i \).

The total cash flow of a payer swap at time \( T_i \) therefore is

\[ \delta N(F(T_{i-1}, T_i) - K). \]

To calculate the fair price of the swap at time \( t < T_0 \), we have to know the time value of the future cash flows. Using the prices of zero-coupon bonds, we find that the price of a payer swap at time \( t \) equals

\[ \Pi_p(t) = \delta N \sum_{i=1}^{n} P(t, T_i)(F(T_{i-1}, T_i) - K). \]

This expression can be rewritten to be totally expressed in terms of the prices of zero-coupon bonds. Using the definition of the forward rate (A.2), the floating leg of the swap has cash flow \( \delta NF(T_{i-1}, T_i) = N \left( \frac{1}{P(T_{i-1}, T_i)} - 1 \right) \) at time \( T_i \). The value at time \( t \) of one unit of currency at time \( T_i \) simply is the price of the \( T_i \)-bond \( P(t, T_i) \). The time value at \( t \) of \( \frac{1}{P(T_{i-1}, T_i)} \) at time \( T_i \) can be found by realizing that buying \( \frac{1}{P(T_{i-1}, T_i)} \) \( T_i \)-bonds at time \( T_{i-1} \) will yield the same payoff. To be able to buy these, we need to have one unit of currency available at time \( T_{i-1} \). But this is exactly the payoff of a \( P(t, T_{i-1}) \) zero-coupon bond. Hence an investment of \( P(t, T_{i-1}) \) at time \( t \) will yield a payoff of \( \frac{1}{P(T_{i-1}, T_i)} \) at time \( T_i \), so we see that the time value at \( t \) of \( \frac{1}{P(T_{i-1}, T_i)} \) at time \( T_i \) must be equal to \( P(t, T_{i-1}) \). We can therefore write the time value at time \( t \) of the floating leg

81
cash flow at time \( T_i \) as \( \delta NF(T_{i-1}, T_i) = N(P(t, T_{i-1}) - P(t, T_i)) \). Using this to rewrite the price of the payer swap yields

\[
\Pi_p(t) = N \left( P(t, T_0) - P(t, T_n) - K \delta \sum_{i=1}^{n} P(t, T_i) \right).
\]

In the case of a receiver interest rate swap, the owner of the contract pays the floating leg and receives the fixed leg. This clearly implies that

\[
\Pi_r(t) = -\Pi_p(t).
\]

A swap at time \( t \) is called at-the-money (ATM) when the swap rate \( K \) is equal to the ‘fair rate’: the rate above which \( \Pi_r(t) = \Pi_p(t) = 0 \). From the above derived swap prices, we see that the ATM swap rate is

\[
S_{T_n}(t) = \frac{P(t, T_0) - P(t, T_n) - \delta \sum_{i=1}^{n} P(t, T_i)}{\delta \sum_{i=1}^{n} P(t, T_i)}. \tag{A.5}
\]

In this thesis, swaps are important instruments because the interest rate paid on a mortgage loan (mortgage rate) with maturity \( T \) is taken as the ATM swap rate for a swap with maturity \( T \) plus a certain (service) spread, see equation (1.1).

**Feynman-Kac and Affine Term Structure**

Let \( Z \subset \mathbb{R} \) be a closed interval with nonempty interior. Consider the stochastic differential equation

\[
dr(t) = b(t_0 + t, r(t))dt + \sigma(t_0 + t, r(t))dW(t), \quad r(0) = r_0 \tag{A.6}
\]

where \( W \) is a Brownian motion, \( b, \sigma : \mathbb{R}_+ \times Z \to \mathbb{R} \) are continuous functions and \( (t_0, r_0) \in \mathbb{R}_+ \times Z \). As shorthand for the drift and diffusion terms we write \( b(t, r) \) and \( \sigma(t, r) \) respectively. Assume that (A.6) admits a strong solution \( r \). We will state a simple version of the Feynman-Kac formula.

**Theorem A.1 (Feynman-Kac).** Let \( T > 0 \) and \( \Phi \) be a continuous function on \( Z \). Assume that \( F(t, r) \in C^{1,2}([0, T] \times Z) \) is a solution to the boundary value problem

\[
\partial_t F(t, r) + b(t, r)\partial_r F(t, r) + \sigma^2(t, r)\partial_{rr} F(t, r) - rF(t, r) = 0, \quad F(T, r) = \Phi(r), \tag{A.7}
\]

on \([0, T] \times Z\). Then

\[
M(t) = F(t, r(t))e^{-\int_0^t r(u)du}, \quad t \leq T,
\]

is a local martingale. If furthermore either \( M \) is uniformly bounded, or

\[
\mathbb{E} \left[ \int_0^T |\partial_r F(t, r(t))e^{-\int_0^t r(u)du}\sigma(t, r(t))|^2 dt \right] < \infty,
\]

82
then $M$ is a martingale. In that case it also holds that

$$F(t, r(t)) = \mathbb{E} \left[ e^{-\int_t^T r(u) du} \Phi(r(T)) \mid \mathcal{F}_t \right], \quad t \leq T.$$  

**Proof.** See Lemma 5.1 in [9].

Short-rate models provide an affine term-structure, if the $T$-bond prices are of the form

$$P(t, T) = F(t, r; T) = e^{-A(t, T) - B(t, T)r(t)},$$

for smooth functions $A$ and $B$. The Feynman-Kac formula allows us to characterize these models.

**Corollary A.2.** A model with short-rate dynamics (A.6) provides an affine term-structure if and only if

$$b(t, r) = b(t) + \beta(t)r \quad \text{and} \quad \sigma^2(t, r) = a(t) + \alpha(t)r,$$

for continuous functions $b, \beta, a, \alpha$ and if $A$ and $B$ satisfy the differential equations

$$\partial_t A(t, T) = \frac{1}{2} a(t)B^2(t, T) - b(t)B(t, T), \quad A(T, T) = 0, \quad (A.9)$$

$$\partial_t B(t, T) = \frac{1}{2} \alpha(t)B^2(t, T) - \beta(t)B(t, T) - 1, \quad B(T, T) = 0, \quad (A.10)$$

for all $t \leq T$.

**Proof.** See Proposition 5.2 in [9].

---

### A.2. Theorems for Chapter 2

This section will state and sometimes prove results that are needed in Chapter 2.

**Riesz-Markov-Kakutani Representation Theorem**

We start by formulating the Riesz-Markov-Kakutani Representation Theorem (not to be mistaken with other versions of Riesz Representation Theorems).

**Theorem A.3 (Riesz-Markov-Kakutani).** Let $X$ be a locally compact Hausdorff space and $T$ a linear functional on $C_c(X)$, the space of continuous functions with compact support. Then there exists a $\sigma$-algebra $\mathcal{A}$ in $X$ which contains all Borel sets in $X$ and a unique positive Borel measure $\mu$ on $\mathcal{A}$ such that

$$T(f) = \int_X f \, d\mu,$$

for every $f \in C_c(X)$.

**Proof.** E.g. see Theorem 2.14 in [32].
The basis of a dual space

Below we formulate the theorem characterizing a basis of the dual space of a linear space, necessary in the proof of Theorem 2.4. Note that $\mathbb{F}$ can be either $\mathbb{R}$ or $\mathbb{C}$.

**Theorem A.4.** Let $X$ be an $n$-dimensional normed linear space with basis $\{v_1, \ldots, v_n\}$. Then its dual space $X'$, the space of linear transformations of $X$ to $\mathbb{F}$, has a basis $\{f_1, \ldots, f_n\}$ such that $f_j(v_k) = \delta_{jk}$, for $1 \leq j, k \leq n$, where $\delta_{jk}$ is the Kronecker delta. In particular, $\dim X' = \dim X$.

**Proof.** See Theorem 5.1 from [34]. □

The Baire Category Theorem

Below we will state the Baire Category Theorem, necessary in proving Lemma 2.6. We first define the notion of a nowhere dense set.

**Definition A.5.** Let $X$ be a topological space. We say a set $E \subseteq X$ is nowhere dense if its closure has empty interior.

**Theorem A.6 (Baire Category Theorem).** Let $X$ be a nonempty complete metric space. Then $X$ is not the countable union of nowhere dense sets.

**Proof.** E.g. see Rudin [32]. □

A countable union of nowhere dense sets is sometimes called a set of the *first category*. All other subsets of the topological space $X$ are of the *second category*. This explains the name of Theorem A.6, which can be reformulated as: no complete metric space is of the first category.

Mollifiers

There are several definitions of mollifiers. Their essential use is to create sequences of smooth functions that approximate a general function by mollification. We give the definition from [1].

**Definition A.7.** Let $\phi \in C^\infty_c(\mathbb{R}^n)$ with properties

(i) $\phi \geq 0$,

(ii) $\phi(x) = 0$ if $|x| \geq 1$,

(iii) $\int_{\mathbb{R}^n} \phi(x) \, dx = 1$.

For $\varepsilon > 0$, the function $\phi_\varepsilon(x) := \varepsilon^{-n} \phi(x/\varepsilon)$ is called a *mollifier*. For functions $f$ for which the integral converges, the convolution

$$\phi_\varepsilon * f(x) = \int_{\mathbb{R}^n} \phi_\varepsilon(x - y) f(y) \, dy,$$

is called the *mollification* of $f$. 
It is easy to check that $\phi \in C^\infty_c(\mathbb{R}^n)$ and is also nonnegative. Furthermore, it satisfies $\phi(x) = 0$ if $|x| \geq \varepsilon$ and $\int_{\mathbb{R}^n} \phi(x) \, dx = 1$. If $f$ is integrable we see that $\phi \ast f$ is in $C^\infty(\mathbb{R}^n)$.

A common function $\phi$ to define a mollifier is presented in the next example.

Example A.8. The function

$$\phi(x) = \begin{cases} I_n^{-1} e^{-1/(1-|x|^2)} & \text{for } |x| < 1 \\ 0 & \text{for } |x| \geq 1, \end{cases}$$

where $I_n$ is a normalization factor such that $\phi$ satisfies property (iii) in Definition A.7 is a commonly used function in mollification.

The approximation result for mollifications used in Proposition 2.10, is Theorem A.9 below ([1]).

Theorem A.9. For all $\varepsilon > 0$, let $\phi$ and $\phi_\varepsilon$ be as in Definition A.7. For $f \in C(\mathbb{R}^n)$,

$$\phi_\varepsilon \ast f(x) \to f(x),$$

uniformly on compacta as $\varepsilon \to 0^+$. 

Proof. Let $K \subset \mathbb{R}^n$ compact and let $\varepsilon' > 0$. Then, because $f$ is uniformly continuous on $K$, we can find an $\varepsilon > 0$ such that for all $x \in K$,

$$|\phi_\varepsilon \ast f(x) - f(x)| = \left| \int_{\mathbb{R}^n} \phi_\varepsilon(x-y)(f(y) - f(x)) \, dy \right| \leq \sup_{|x-y| < \varepsilon} |f(y) - f(x)| < \varepsilon',$$

where we used that $\phi_\varepsilon$ integrates to unity. \qed
B. The Stone-Weierstrass Theorem

The formulation of the Stone-Weierstrass theorem and its proof are based on a note by Brosowski and Deutsch [4]. They present a proof that only uses elementary properties of compact sets, continuous functions and the Bernoulli inequality, which is stated below for completeness.

Proposition B.1 (Bernoulli inequality). If $x \geq -1$, then for every $n \in \mathbb{N}$ we have
\[ (1 + x)^n \geq 1 + nx. \] (B.1)

We also mention the definition of an algebra of functions.

Definition B.2. Let $A$ be a family of real valued functions on a set $X$. We say $A$ is an algebra of functions if $A$ is closed under multiplication, addition and scalar multiplication.

If $A$ contains the function that is constant 1 (and thus any other constant function by the properties of an algebra), we say that $A$ contains constants.

If $A$ has the property that for every $x, y \in X$ with $x \neq y$ there exists a function $f \in A$ such that $f(x) \neq f(y)$, then we say that $A$ separates points on $X$.

Theorem B.3 (Stone Weierstrass). Let $(X, T)$ be a compact topological space. If $A \subset C(X)$ is an algebra that contains constants and separates points, then $A$ is uniformly dense in $C(X)$.

As in [4], we will split the proof into three parts and write $\| \cdot \|$ for the supremum norm on $C(X)$. We will first prove two lemma’s

Lemma B.4. Consider the setup of the Stone-Weierstrass theorem. Let $x_0 \in X$ and let $U$ be an open neighbourhood of $x_0$. Then there exists an open neighbourhood $V \subseteq U$ of $x_0$, with the property that for all $\varepsilon > 0$, there exists a $f \in A$ such that

(i) $0 \leq f(x) \leq 1$, for $x \in X$.
(ii) $f(x) < \varepsilon$, for $x \in V$.
(iii) $f(x) > 1 - \varepsilon$, for $x \in X \setminus U$.

Proof. Since $A$ separates points, for every $x \in X \setminus U$ we can find a $g_x \in A$ such that $g_x(x) \neq g_x(x_0)$. Next define the function $h_x = g_x - g_x(x_0) \cdot 1$. Then $h_x(x) \neq h_x(x_0) = 0$. Note that $h_x \in A$ since $A$ is an algebra and contains constants. Now define $p_x \in A$ by $p_x := (1/\| h_x^2 \|) \cdot h_x^2$, such that $p_x(x) > 0$, $p_x(x_0) = 0$ and $0 \leq p_x \leq 1$ on $X$.  

86
Define $U(x) := \{y \in X \mid p_x(y) > 0\}$. Then $U(x)$ is an open neighbourhood of $x$ and by compactness of $X \setminus U$ (closed subset of compact set is compact) we can find a finite number of points $\{x_1, \ldots, x_m\}$ in $X \setminus U$ such that the above defined open neighbourhoods $U(x_i)$ form a finite open cover of $X \setminus U$. Let $p = (1/m)\sum_{i=1}^{m} p_{x_i}$. Then again $p \in A$, $p(x_0) = 0$, $0 \leq p \leq 1$ and $p > 0$ on $X \setminus U$.

Since $p$ is a linear combination of continuous functions on $X$ ($A \subset C(X)$), it attains a minimum on the compact set $X \setminus U$. This means that there exists a $0 < \delta < 1$ such that $p \geq \delta$ on $X \setminus U$. Now define $V := \{x \in X \mid p(x) < \delta/2\}$. Then $V \subset U$ and $V$ is an open neighbourhood of $x_0$.

If we take $k$ to be the smallest integer greater that $1/\delta$, then $k - 1 \leq 1/\delta$ such that $k \leq (1 + \delta)/\delta < 2/\delta$. But this means that $1 < k\delta < 2$. Now, for $n = 1, 2, \ldots$, define the functions $q_n(x) = (1 - p^n(x))^{k^n}$. Then $q_n \in A$, $q_n(x_0) = 1$ and surely $0 \leq q_n \leq 1$.

By definition of the set $V$, we have for every $x \in V$ that $kp(x) < k\delta/2 < 1$. Hence application of Bernoulli’s inequality we find for $x \in V$ that

$$q_n(x) = (1 - p^n(x))^{k^n} \geq 1 - (kp(x))^n \geq 1 - (k\delta/2)^n \to 1.$$ 

Implying that $q_n(x) \to 1$ uniformly on $V$. Similarly, noting that for $x \in X \setminus U$ we have $kp(x) \geq k\delta > 1$, we find

$$q_n(x) = \frac{1}{(kp(x))^n} (1 - p^n(x))^{k^n} k^n p^n(x) \leq \frac{1}{(kp(x))^n} (1 - p^n(x))^{k^n} (1 + k^n p^n(x)) \leq \frac{1}{(kp(x))^n} (1 - p^n(x))^{k^n} (1 + p^n(x))^{k^n} \leq \frac{1}{(k\delta)^n} (1 - p^{2n}(x))^{k^n} \leq \frac{1}{(k\delta)^n} \to 0,$$

where we used Bernoulli’s inequality at the second inequality. This in turn implies that $q_n \to 0$ uniformly on $X \setminus U$. To conclude, let $\varepsilon > 0$. By the above we can take $n$ large enough to ensure that $q_n > 1 - \varepsilon$ on $V$ and $q_n < \varepsilon$ on $X \setminus U$. Taking $f = 1 - q_n$ now yields the result of the lemma.$\square$

The next lemma builds on this result and proves a similar result on two disjoint closed sets.

**Lemma B.5.** Consider the setup of the Stone-Weierstrass theorem. Let $A$ and $B$ be two disjoint closed subsets of $X$. Then for each $0 < \varepsilon < 1$, there exists an $f \in A$ such that

(i) $0 \leq f(x) \leq 1$, for $x \in X$.

(ii) $f(x) < \varepsilon$, for $x \in A$.
(iii) \( f(x) > 1 - \varepsilon, \) for \( x \in B. \)

**Proof.** Define the open set \( U = X \setminus B. \) Since \( A \) and \( B \) are disjoint, so \( A \subset U, \) we can invoke Lemma B.4 for each point \( x \in A \) to find an open neighbourhood \( V(x) \) of \( x. \) Since a closed subset of a compact set is compact, we can find a finite set of points \( \{x_1, \ldots, x_m\} \) in \( A \) such that the corresponding \( V(x_i) \) form an open cover of \( A. \) For each \( i = 1, \ldots, m, \) by the properties \( V(x_i), \) there exist functions \( f_i \in A \) with the property that \( f_i < \varepsilon/m \) on \( V(x_i), \) \( f_i > 1 - \varepsilon/m \) on \( B = X \setminus U \) and \( 0 \leq f_i \leq 1 \) on the whole of \( X. \) Multiplying all these functions, we define \( f := \prod_{i=1}^{m} f_i. \) Clearly \( f \in A \) and still \( 0 \leq f \leq 1. \) Furthermore on \( \bigcup_{i=1}^{m} V(x_i) \) we have \( f < \varepsilon/m \leq \varepsilon, \) and since \( A \subset \bigcup_{i=1}^{m} V(x_i) \) this property also holds on the whole of \( A. \) To show the right behaviour of \( f \) on \( B, \) we again need Bernoulli's inequality. On \( B \) we have

\[
f = \prod_{i=1}^{m} f_i > (1 - \frac{\varepsilon}{m})^m \geq 1 - \varepsilon.
\]

We conclude that the function \( f \) satisfies all the required properties. \( \square \)

Now we are ready to prove the main theorem of this section.

**Proof of the Stone-Weierstrass theorem.** Take \( f \in C(X) \) and let \( \varepsilon > 0. \) We will have to find a \( g \in A \) such that

\[
\|f - g\| = \sup_{x \in X} |f(x) - g(x)| < \varepsilon. \tag{B.2}
\]

Without loss of generality we may assume that \( f \geq 0. \) Since if this was not the case, we could find a \( g \) for the function \( f + \|f\| \) and then subtract the constant \( \|f\| \) (which is finite by compactness of \( X \) and continuity of \( f \)) again from both functions. Furthermore, assume \( \varepsilon/2 < \frac{1}{3}. \)

Pick \( n \in \mathbb{N} \) such that \( (n - 1)\varepsilon/2 \geq \|f\| \) and define for \( j = 0, 1, \ldots, n \) the sets

\[
A_j := \{x \in X \mid f(x) \leq \frac{(j - 1)}{3} \cdot \varepsilon\}, \quad B_j := \{x \in X \mid f(x) \geq \frac{(j + 1)}{3} \cdot \varepsilon\}.
\]

Note that \( A_0 = \emptyset = B_n \) and \( A_n = X. \) For all \( j \) the sets \( A_j \) and \( B_j \) are disjoint and closed and the sets are nested in the sense that \( A_0 \subset A_1 \subset \ldots \subset A_n \) and \( B_0 \supset \ldots \supset B_n. \)

Using Lemma B.5 we can find an \( g_j \in A \) for each \( j = 0, \ldots, n \) such that \( 0 \leq g_j \leq 1 \) on \( X, \) \( g_j < \varepsilon/(2n) \) on \( A_j \) and \( g_j > 1 - \varepsilon/(2n) \) on \( B_j. \) We define the function

\[
g(x) := \frac{\varepsilon}{2} \sum_{j=1}^{n} g_j(x).
\]

Note that \( g \) is a linear combination of functions in \( g_i \in A \) hence \( g \in A. \)

Pick any \( x \in X. \) By the nested structure of the \( A_j \) and the continuity of \( f, \) there is a \( j \geq 1 \) such that \( x \in A_j \setminus A_{j-1}, \) so

\[
\left( j - \frac{4}{3} \right) \cdot \frac{\varepsilon}{2} < f(x) \leq \left( j - \frac{1}{3} \right) \cdot \frac{\varepsilon}{2}. \tag{B.3}
\]

88
Furthermore, for every $i \geq j$ we have

$$g_i(x) < \frac{\varepsilon}{2n}. \quad \text{(B.4)}$$

Note that equation (B.3) also implies that $x \in B_i$ for every $i \leq j - 2$, meaning that for all $i \leq j - 2$ we have

$$g_i(x) > 1 - \frac{\varepsilon}{2n}. \quad \text{(B.5)}$$

Using that $g_i(x) \leq 1$ for all $i$, that $j \geq 1$, $\varepsilon/2 < \frac{1}{3}$ and applying equation (B.4) yields

$$g(x) = \frac{\varepsilon}{2} \sum_{i=0}^{j-1} g_i(x) + \frac{\varepsilon}{2} \sum_{i=j}^{n} g_i(x)$$

$$\leq \frac{j \varepsilon}{2} + \frac{\varepsilon}{2} \frac{\varepsilon}{2n} (n - j + 1)$$

$$\leq \frac{j \varepsilon}{2} + \left( \frac{\varepsilon}{2} \right)^2$$

$$\leq \left( j + \frac{1}{3} \right) \cdot \frac{\varepsilon}{2}.$$

Since $g(x) \geq 0$, for $j = 1$ we trivially find $g(x) > \left( j - \frac{4}{3} \right) \cdot \frac{\varepsilon}{2}$. For $j \geq 2$ using equation (B.5) we get

$$g(x) \geq \frac{\varepsilon}{2} \sum_{i=0}^{j-2} g_i(x)$$

$$> (j - 1) \frac{\varepsilon}{2} \left( 1 - \frac{\varepsilon}{2n} \right)$$

$$> (j - 1) \frac{\varepsilon}{2} - \left( \frac{\varepsilon}{2} \right)^2$$

$$> \left( j - \frac{4}{3} \right) \cdot \frac{\varepsilon}{2}.$$

Taking into account equation (B.3) we see that

$$|f(x) - g(x)| \leq \frac{\varepsilon}{2} \left( \left( j + \frac{1}{3} \right) - \left( j - \frac{4}{3} \right) \right) < \varepsilon.$$

Since this holds for every $x \in X$, we conclude that equation (B.2) holds true for this $g$. This proves the theorem.
C. Historical Proof of the Universal Approximation Theorem

This chapter will treat the Universal Approximation Theorem from the historically very relevant article by Hornik, Stinchcombe and White [14], but we will sometimes deviate from and expand on results from the article. Following the approach of Hornik et al., we will first prove the approximation result for neural networks with one hidden layer and one-dimensional linear output in Theorem C.8, then use corollaries from Section 2.2 to extend the result to general neural networks with linear output neurons.

A lot of work has already been done in Section 2.2, to which we will refer regularly. Recall Definition 2.1, introducing the class of one-layer neural networks with one-dimensional linear output. Because we have to make a strict distinction between activation functions that are sigmoidal (Definition C.3) and those that are not, we will use the notation $G$ for non-sigmoidal activation functions in this section. We will need an auxiliary class of functions.

**Definition C.1.** For any Borel measurable function $G : \mathbb{R} \to \mathbb{R}$ and $n \in \mathbb{N}$ we define the class of functions $\Sigma\Pi^n(G)$ as

$$\left\{ f : \mathbb{R}^n \to \mathbb{R} \mid f(x) = \sum_{i=1}^{d} b_i \prod_{k=1}^{L_i} G(A_{ik}(x)); b_i \in \mathbb{R}, A_{ik} \in \mathbb{R}^n, L_i, d \in \mathbb{N} \right\}.$$  

(C.1)

The first main result of this section will show that $\Sigma\Pi^n(G)$, for continuous non-constant $G$, is uniformly dense on compacta in $C(\mathbb{R}^n)$, the set of continuous function from $\mathbb{R}^n$ to $\mathbb{R}$. It also shows denseness in $M^n$, the set of measurable functions from $\mathbb{R}^n$ to $\mathbb{R}$, with respect to the Ky Fan metric $\rho_\mu$, defined in Definition 2.12. This is Theorem C.2 below. The proof of this theorem relies heavily on the Stone-Weierstrass theorem, which can be found in Appendix B.

**Theorem C.2.** Let $n \in \mathbb{N}$ and $G : \mathbb{R} \to \mathbb{R}$ be any continuous non-constant function. Then $\Sigma\Pi^n(G)$ is uniformly dense on compacta in $C(\mathbb{R}^n)$ and dense in $M^n$ with respect to the metric $\rho_\mu$.

**Proof.** Let $K$ be a compact set in $\mathbb{R}^n$. From the definition of an algebra of functions, Definition B.2, it is clear $\Sigma\Pi^n(G)$ is an algebra of functions on $K$. By continuity of $G$, we also have $\Sigma\Pi^n(G) \subseteq C(\mathbb{R}^n)$. To apply the Stone-Weierstrass theorem, we have to check that it also contains constants and separates points. We will start with the first. Since $G$ is non-constant, we can pick $b \in \mathbb{R}$ such that $G(b) \neq 0$. Set $A(x) = 0 \cdot x + b$. Then for all $x \in K$ we have $G(A(x)) = G(b)$. We now see that choosing all $A_{ik}$ in...
equation (C.1) as $A(x)$ yields a constant function that is in $\Sigma \Pi^n(G)$. Since $\Sigma \Pi^n(G)$ is an algebra, we conclude that it contains all constant functions.

To see that $\Sigma \Pi^n(G)$ separates points on $K$, select $x, y \in K$ such that $x \neq y$. Again, since $G$ is not a constant function we can find points $a, b \in \mathbb{R}$ such that $a \neq b$ and $G(a) \neq G(b)$. Now pick $A \in \mathbb{A}^n$ as the affine function with $A(x) = a$ and $A(y) = b$. Then clearly $G(A(x)) \neq G(A(y))$. Choosing all $A_{ik}$ in equation (C.1) as $A(x)$ yields a function that separates $x$ and $y$. This proves that $\Sigma \Pi^n(G)$ separates points on $K$.

Now by virtue of the Stone-Weierstrass Theorem, $\Sigma \Pi^n(G)$ is uniformly dense in the real continuous functions on $K$. Since this argument is valid for any compact $K \subset \mathbb{R}^n$, this shows denseness on compacta in $C(\mathbb{R}^n)$.

The exact same proof as that of Corollary 2.19 applied to $\Sigma \Pi^n(G)$, shows that $\Sigma \Pi^n(G)$ is dense in $\mathcal{M}^n$ with respect to $\rho_{\mu}$. \hfill $\square$

In the results above, we are restricted to continuous activation functions $G$. By assuming a certain type of functions defined below, we can get rid of the continuity restriction.

**Definition C.3.** A function $\sigma : \mathbb{R} \to [0, 1]$ is called a sigmoidal function if

(i) $\sigma$ is non-decreasing

(ii) $\lim_{x \to \infty} \sigma(x) = 1$

(iii) $\lim_{x \to -\infty} \sigma(x) = 0$

For a long time, sigmoidal functions were the most commonly used activation functions in neural networks.

We will now show that the result of Theorem C.2 also holds for $\Sigma \Pi^n(\sigma)$ functions, where $\sigma$ is a sigmoidal function. We first need another lemma.

**Lemma C.4.** Let $\sigma$ be an arbitrary sigmoidal function and let $F$ be a continuous sigmoidal function. Then, for every $\varepsilon > 0$ there exists an $H \in \Sigma^1(\sigma)$ such that $\sup_{x \in \mathbb{R}} |F(x) - H(x)| < \varepsilon$.

**Proof.** Let $\varepsilon > 0$ and without loss of generality assume $\varepsilon < 1$. We will construct the function $H$ below.

Choose $Q \in \mathbb{N}$ such that $1/Q < \varepsilon/2$. Set $b_i = 1/Q$ for $i = 1, \ldots, Q - 1$. By the limiting behaviour of sigmoidal functions, we can pick an $M > 0$ such that $\sigma(-M) < \varepsilon/(2Q)$ and $\sigma(M) > 1 - \varepsilon/(2Q)$. Furthermore, using the continuity and limiting properties of $F$, define $r_i = \sup\{x : F(x) = i/Q\}$ for $i = 1, \ldots, Q - 1$ and set $r_Q = \sup\{x : F(x) = 1 - 1/(2Q)\}$. Lastly, for $r < s$, choose $A_{r,s} \in A^1$ as the affine function for which $A_{r,s}(r) = -M$ and $A_{r,s}(s) = M$.

Now define

$$H(x) := \sum_{i=1}^{Q-1} b_i \sigma(A_{r_i, r_{i+1}}(x)).$$

By construction $H \in \Sigma^1(\sigma)$, and on each of the intervals $(-\infty, r_1], (r_1, r_2], \ldots, (r_Q, \infty)$ it holds that $|F(x) - H(x)| < \varepsilon$. \hfill $\square$
Theorem C.5. For every \( n \in \mathbb{N} \), every sigmoidal function \( \sigma \) and every probability measure \( \mu \) on \((\mathbb{R}^n,\mathcal{B}(\mathbb{R}^n))\), \( \Sigma \Pi^n(\sigma) \) is uniformly dense on compacta in \( C(\mathbb{R}^n) \) and dense in \( \mathcal{M}^n \) with respect to the metric \( \rho_\mu \).

Proof. We will first show that every function of the form \( \prod_{k=1}^L F(A_k(x)) \), where \( F \) is any continuous sigmoidal function and \( A_k \in \mathbb{A}^n \), can be uniformly approximated by functions in \( \Sigma \Pi^n(\sigma) \).

Let \( \varepsilon > 0 \). Since \( [0,1]^L \) is compact and the function from \( \mathbb{R}^L \) to \( \mathbb{R} \) that multiplies the entries of its input vectors is continuous, we can find a \( \delta > 0 \) such that \( |x_k - y_k| < \delta \) for \( k = 1, \ldots, L \) implies \( \prod_{k=1}^L x_k - \prod_{k=1}^L y_k| < \varepsilon \). By virtue of Lemma C.4 there exists a function \( H \in \Sigma^3(\sigma) \) such that \( \sup_{x \in \mathbb{R}} |F(x) - H(x)| < \delta \). Therefore, by the above uniform continuity argument

\[
\sup_{x \in \mathbb{R}^n} |\prod_{k=1}^L F(A_k(x)) - \prod_{k=1}^L H(A_k(x))| < \varepsilon.
\]

Recall that this \( H \) will be of the form \( H(x) = \sum_{i=1}^d b_i \sigma(A_i^1(x)) \) for some \( d \in \mathbb{N} \), weights \( b_i \) and the \( A_i^1 \in \mathbb{A}^l \). Because \( A_i^1 \circ A_k \in \mathbb{A}^n \), we see by multiplying out the sums that \( \prod_{k=1}^L H(A_k(x)) \in \Sigma \Pi^n(\sigma) \). We can thus uniformly approximate functions of the form \( \prod_{k=1}^L F(A_k(x)) \) by functions in \( \Sigma \Pi^n(\sigma) \).

Since \( \Sigma \Pi^n(F) \) functions are linear combinations of functions of the form \( \prod_{k=1}^L F(A_k(x)) \), the above result implies that we can also uniformly approximate \( \Sigma \Pi^n(F) \) functions with functions in \( \Sigma \Pi^n(\sigma) \). Thus \( \Sigma \Pi^n(\sigma) \) is uniformly dense in \( \Sigma \Pi^n(F) \), which in turn by Theorem C.2 is uniformly dense on compacta in \( C(\mathbb{R}^n) \). The triangle inequality now shows that \( \Sigma \Pi^n(\sigma) \) is also uniformly dense on compacta in \( \mathcal{M}^n \).

By Lemma 2.14 \( \Sigma \Pi^n(\sigma) \) is dense in \( \Sigma \Pi^n(F) \) with respect to the \( \rho_\mu \) metric. Applying Theorem C.2 to \( \Sigma \Pi^n(F) \) and applying the triangle inequality again yields that \( \Sigma \Pi^n(\sigma) \) is also \( \rho_\mu \)-dense in \( \mathcal{M}^n \).

To extend the result of this theorem to artificial neural networks, we need two more lemmas.

Lemma C.6. For every sigmoidal function \( \sigma \), every \( \varepsilon > 0 \) and \( M > 0 \) there exists a \( h_M^\varepsilon \in \Sigma^3(\sigma) \) such that

\[
\sup_{x \in [-M,M]} |h_M^\varepsilon(x) - \cos(x)| < \varepsilon. \tag{C.2}
\]

Proof. Define the continuous sigmoidal function

\[
F(x) = \frac{1 + \cos(x + 3\pi/2)}{2} \mathbb{1}_{\{-\pi/2 \leq x \leq \pi/2\}} + \mathbb{1}_{\{x > \pi/2\}}.
\]

We see that \( F \) is 0 for small values of \( x \), for \( x \in (-\pi/2, \pi/2) \) it follows the slope of a cosine up to the value 1. Note that by taking linear combinations of the functions \( F \) with shifted arguments, we can precisely get the cosine function on \([-M,M]\). In this, we
need that we can make \( F \) a constant function by shifting the indicator on \((−\pi/2, \pi/2)\) out of the interval \([-M,M]\). This is illustrated in Figure C.1.

By Lemma C.4 we can approximate \( F \) uniformly arbitrarily well by elements of \( \Sigma^1(\sigma) \). Hence we can also approximate linear combinations of \( F \) uniformly since \( \Sigma^1(\sigma) \) is closed under taking linear combinations. By application of the triangle inequality we now see that we can find \( h_{\delta M} \in \Sigma^1(\sigma) \) such that equation (C.2) holds.

\[
\text{Figure C.1.:} \text{ An illustration of the reconstruction of the cosine function on } [-5, 5] \text{ by linear combinations of } \sigma \text{ with shifted arguments. The first graph shows the function } F. \text{ The second graph defines the function } G(x) := F(x + \pi/2) - F(x - \pi/2). \text{ The third graph shows the cosine shape by adding three functions } G \text{ and the fourth graph compares the cosine function (red dotted line) with an affine transformation of the function in graph three. Note that } F(x + 10) \text{ is the constant function with value 1.}
\]

Lemma C.7. For any sigmoidal function \( \sigma \), any compact \( K \subset \mathbb{R}^n \), any function \( g : \mathbb{R}^n \to \mathbb{R} \) of the form

\[
g(x) = \sum_{i=1}^{d} b_i \cos(A_i(x)),
\]

with \( d \in \mathbb{N} \), \( b_i \in \mathbb{R} \), \( A_i \in \mathbb{A}^n \), and for any \( \varepsilon > 0 \), there exists an \( f \in \Sigma^n(\sigma) \) such that \( \sup_{x \in K} |g(x) - f(x)| < \varepsilon \).
Proof. Since continuous functions assume there minimum and maximum on a compact set, we can choose an $M > 0$ such that $-M \leq A_i \leq M$ on $K$ for each $i = 1, \ldots, m$. Define $\varepsilon' = \varepsilon \left( \sum_{i=1}^{d} |b_i| \right)^{-1}$. Then by Lemma C.6, we can find a function $h_M' \in \Sigma^1(\sigma)$ such that $\sup_{x \in [-M,M]} |h_M'(x) - \cos(x)| < \varepsilon'$. Define $f(x) = \sum_{i=1}^{d} b_i h_M'(A_i(x))$. Then for $x \in K$ by the triangle inequality
\[
|f(x) - g(x)| \leq \sum_{i=1}^{d} |b_i||h_M'(A_i(x)) - \cos(A_i(x))| < \varepsilon' \sum_{i=1}^{d} |b_i| = \varepsilon.
\]
Since $h_M' \in \Sigma^1(\sigma)$ and $A_i \in \mathbb{A}^n$, we see $h_M' \circ A_i \in \Sigma^n(\sigma)$. Because $f$ is a linear combination of these functions we conclude $f \in \Sigma^n(\sigma)$. \hfill \Box

The choice to approximate the cosine function in Lemmas C.6 and C.7 seems arbitrary. In the proof of Theorem C.8 below, we shall see that this choice useful because it is possible to write a product of two cosines as a sum of two cosines. This allows us to move from $\Sigma\Pi^n$ functions to the class of neural networks $\Sigma^n$. Of course, this product-to-sum property also holds for the sine function, showing that there are other possible ways to arrive at Theorem C.8, the main result of this section for one-layer single linear output neural network.

**Theorem C.8.** For every $n \in \mathbb{N}$, every sigmoidal function $\sigma$ and every probability measure $\mu$ on $(\mathbb{R}^n, \mathcal{B}(\mathbb{R}^n))$, $\Sigma^n(\sigma)$ is uniformly dense on compacta in $C(\mathbb{R}^n)$ and dense in $\mathcal{M}^n$ with respect to the metric $\rho_{\mu}$.

Proof. We will first show the denseness on compacta in $C(\mathbb{R}^n)$. By Theorem C.2 the class of functions $\Sigma\Pi^n(\cos)$, is uniformly dense on compacta in $C(\mathbb{R}^n)$. Functions of this class are of the form $\sum_{i=1}^{d} b_i \prod_{i=1}^{L_i} \cos(A_{ik}(x))$, for $b_i \in \mathbb{R}$, $A_{ik} \in \mathbb{A}^n$ and $L_i, d \in \mathbb{N}$. Recall the trigonometric identity $2\cos(x)\cos(y) = \cos(a + b) + \cos(a - b)$. Using this identity repeatedly, all functions in $\Sigma\Pi^n(\cos)$ can be written as functions of the form $\sum_{i=1}^{d} b'_i \cos(A'_i(x))$ for some $b'_i \in \mathbb{R}$, $d' \in \mathbb{N}$ and $A'_i \in \mathbb{A}^n$. Application of Lemma C.7 shows that $\Sigma^n(\sigma)$ is uniformly dense on compacta in $\Sigma\Pi^n(\cos)$. The triangle inequality now yields the result.

Proving that $\Sigma^n(\sigma)$ is $\rho_{\mu}$-dense in $\mathcal{M}^n$ follows easily from the other results. Given that $\Sigma^n(\sigma)$ is uniformly dense on compacta in $C(\mathbb{R}^n)$, application of Lemma 2.14 yields that $\Sigma^n(\sigma)$ is also $\rho_{\mu}$ dense in $C(\mathbb{R}^n)$. Since by Lemma 2.18 $C(\mathbb{R}^n)$ is dense in $\mathcal{M}^n$, simple use of the triangle inequality now shows that $\Sigma^n(\sigma)$ is dense in $\mathcal{M}^n$ with respect to the $\rho_{\mu}$-metric. \hfill \Box

Corollaries 2.22 and 2.24 extend the result of Theorem C.8 to neural networks $\Sigma^{n,m}_L(\sigma)$ from $\mathbb{R}^n \to \mathbb{R}^m$ with $L$ layers.

These results show that artificial neural networks with sigmoidal activation functions are capable of approximating any measurable function from $\mathbb{R}^n$ to $\mathbb{R}^m$, with any desired precision. Early proofs like this one of the Universal Approximation Theorem for artificial neural networks were very important for the acceptance of neural networks in fields beyond experimental computer science.