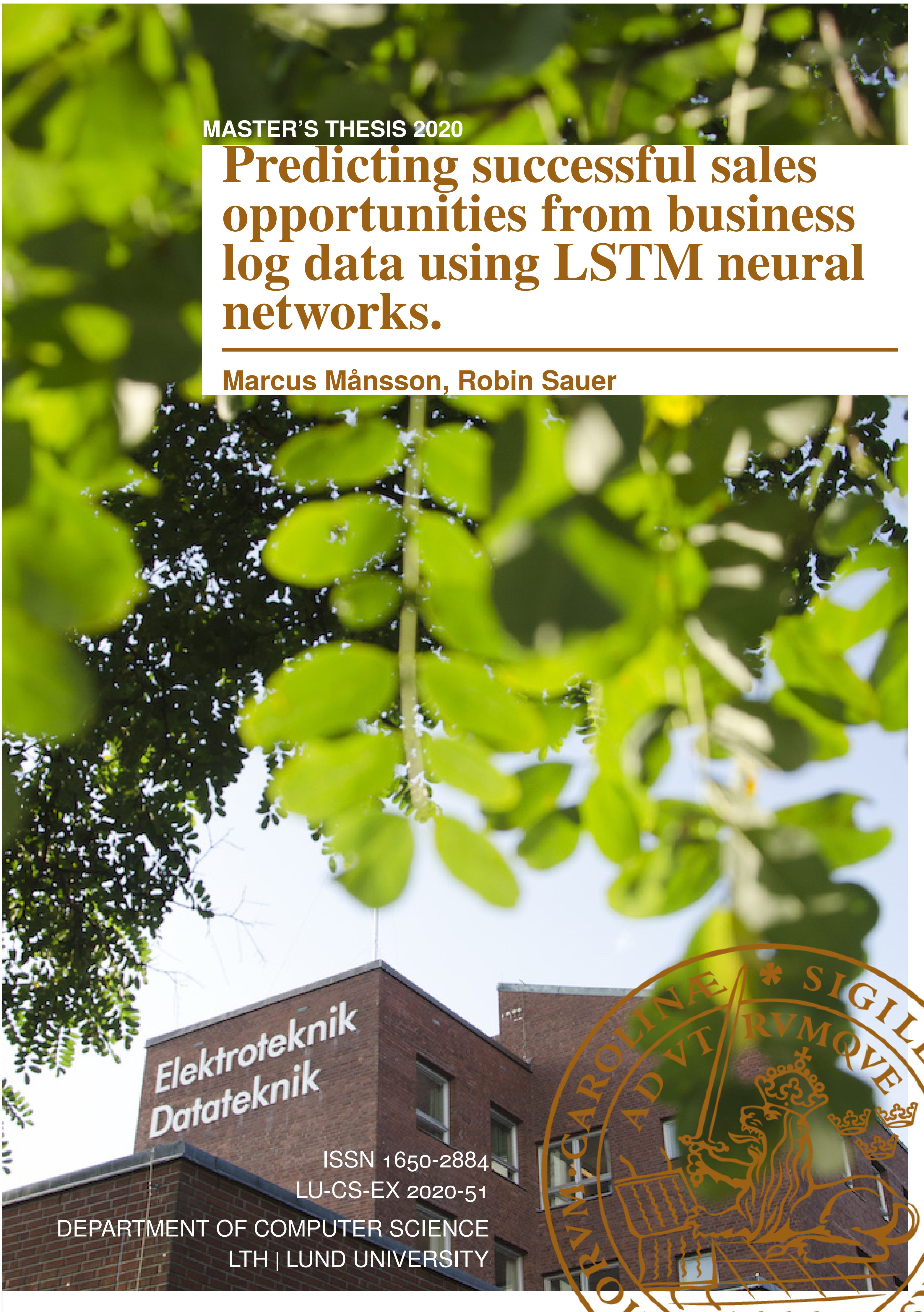


MASTER'S THESIS 2020

Predicting successful sales opportunities from business log data using LSTM neural networks.

Marcus Månsson, Robin Sauer

ISSN 1650-2884
LU-CS-EX 2020-51

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

LU-CS-EX 2020-51

Predicting successful sales opportunities from business log data using LSTM neural networks.

Marcus Månsson, Robin Sauer

Predicting successful sales opportunities from business log data using LSTM neural networks.

Marcus Månsson
mob12mma@student.lu.se

Robin Sauer
dat14rsa@student.lu.se

August 31, 2020

Master's thesis work carried out at Axis Communications AB.

Supervisors: Robin Gustafsson, robin.gustafsson@axis.com
Martin Höst, martin.host@cs.lth.se

Examiner: Björn Regnell, bjorn.regnell@cs.lth.se

Abstract

Axis Communications is a leading manufacturer of network cameras. Axis sells its products to distributors in a project oriented process during which information is recorded in the project as it progresses. Projects contain an estimate of the likelihood that the project will succeed, set by the sales person in charge of the project. In this paper we investigate the possibilities of improving this estimate using machine learning and how this can benefit Axis.

The life cycle of a project can be represented as sequential time series data and we exploit this by exploring Long Short-Term Memory (LSTM) neural networks. We found that project log data can be used to predict project success rate to a meaningful degree and could serve as a useful starting point for the demand planning of products. The LSTM network provides a classification accuracy of 83.1% and outperforms the other baseline machine learning models, logistic regression, MLP and random forest, used in this report.

Keywords: Machine Learning, Feature extraction, Recurrent Neural Networks, LSTM, Prediction, Business Data, Log Data

Acknowledgements

First we would like to thank Axis for giving us the opportunity to do our thesis with them and providing us with equipment and workspace.

A special thanks to Simon Steiner at the BI department whose enthusiasm for our project made sure it happened, and to the Axis tools team for all the encouragement and support. We would also like to thank Alexandra Wikström and Anton Gustavsson for the interview.

Finally, a big thank you to our supervisors Martin Höst at LTH for the invaluable input and Robin Gustafsson at Axis without whom this thesis wouldn't even have been possible.

Robin Sauer and Marcus Månsson
Lund, Sweden - August 2019

Contents

Chapter 1

Introduction

Machine learning (ML), neural networks in particular, has had a huge upswing in popularity since the beginning of the millennium and has become a common buzzword in business contexts as well as in daily life, promising many amazing things spanning everything from simple spam filters [15] to advanced image recognition for self-driving cars [11] and eventually general AI. However, there are two sides to every coin and many of the advancements made in the field of ML has already seen applications with nefarious intent such as the spreading of deep-fakes, a very convincing face-swapping technology, of famous people doing or saying things they normally would not. Another ethically dubious use case of ML is identification technologies such as facial recognition for identifying and surveilling individuals or certain groups of people. In [19] for example, it was reported that the Chinese government uses facial recognition for tracking and controlling Uighurs, a largely Muslim minority group concentrated in the western parts of China. Not only is the persecution of a certain minority group highly unethical but this technology, used in other scenarios as well, is not perfect and misidentification is a very real problem with potentially devastating consequences to affected people. Going forward, as machine learning continues to evolve, great consideration of the potential impacts of this mostly amazing technology needs to be taken so as to avoid and mitigate unethical usage.

Even though most breakthroughs in ML were quite recent it is an old discipline within the world of computer science and information theory stemming from the field of statistics. Bayes' Theorem, named after the 18th century statistician Thomas Bayes [1] but defined by the famous french scholar Pierre-Simon Laplace in 1812 [2], is one of the important corner stones of modern ML. Neural networks were first mentioned in a 1943 paper [17] and the first implementations of one were made in 1951 by Marvin Lee Minsky [3], and another in 1958 by Frank Rosenblatt called the perceptron [4]. Unfortunately the 60s and 70s did not see much development in the field, in part due to the exaggerated hype of the perceptron which did not deliver on its promises leading to diminished funding. Not until the eighties came by did ML receive a boost in popularity and funding again and in 1997 ML using neural networks really took off when IBM's computer Deep Blue beat

Garry Kasparov, regarded by many as the best chess player ever, at his own game. Since then, as computer hardware and software has developed and become more powerful and widespread, the use of ML algorithms has proliferated quickly and now permeate most, if not all, digital infrastructure in one way or another.

1.1 Background

Axis is a leading network video surveillance company that is part of the canon group and has offices all over the world with their HQ located at Ideon science park in Lund, Sweden. Axis was founded in 1984 and quickly became a top contender in the field of network connected printers. In 1995 Axis released a network solution for storing optical data and became the world leader in this space. The technology was named 'ThinServer' and was a precursor to the field of technology that we today know as 'the internet of things' or 'IoT' for short. In 1996 Axis produced the world's first network camera which revolutionized the industry by shifting video surveillance from an analog technology into a digital one. Since then Axis has been a leader in the field of network video surveillance and has further expanded into other 'IoT' security solutions and now offers solutions spanning network video, access control, audio and more.

Like most other modern companies, Axis sees the potential benefits of machine learning and already is, or is looking at, incorporating machine learning into different business areas. One such area is sales which Axis denotes 'projects'. A project encompasses the entire sales process during which information is gathered over time as it becomes available up until a point when the project is closed after having either succeeded or failed. Initially a project contains limited amounts of information like in which country the sale will take place, some customer information and maybe an estimated sales value. As the project develops, more and more information is added by the sales person in charge of the sale like products, information regarding different stakeholders etc. The sales person updates the information in a project using Axis' CRM (customer relationship management) tool which updates concerned tables in a database we will denote 'Live'. The 'Live' database therefore holds the current state of all projects. For the sake of traceability all updates to 'Live' are also stored in an ever expanding separate system log table stored in a separate database which holds the complete update history of every sale ever conducted at Axis that has gone through the CRM tool. With the advent of machine learning in a business context, Axis has realized the potential value of the data in the log and has been investigating different alternatives for applying machine learning to it. This thesis is one of those alternatives.

In the CRM tool the sales person can set a 'probability of success' estimate for a project i.e. the likelihood that the project will be successful as perceived by the sales person. Unsurprisingly, this metric has been quite poor since it is very difficult to manually predict how a project will fare due to the large amount of parameters to consider, a consequence of which is that different people might give very similar projects very different success rates. Fortunately this is exactly the type of task that machine learning excels at.

1.2 Problem formulation

In this thesis we seek to improve on a project's estimated probability of success currently provided by the sales person in charge of the project by applying machine learning, specifically neural networks, to the historical data relating to projects stored in the system log. Providing a more reliable indication of the success chance of a project could aid Axis in project prioritization, revenue estimation, demand planning and more, in addition to providing a proof of concept regarding the application of machine learning on Axis' log data. Since the log data contains all historic records of all projects we realized that the data can be extracted as a time series on which Long short-term memory (abbreviated LSTM, see Section 2.2.4) neural networks perform well. We therefore formulate our overarching problem as:

> *Using machine learning, is it possible to provide an accurate estimate of the probability of a project being successful?*
>
> *Since the data can be extracted as a time series, potentially providing a temporal relationship within the data, are LSTM networks the superior choice for this task when compared to more mainstream machine learning models, specifically random forests and feed forward neural networks?*
>
> *To what extent can a successful implementation of such a model benefit Axis?*

1.3 Related work

LSTMs have been used to process sequences in several different domains. Examples of successful applications involves areas such as speech recognition [12], machine translation [14] and time series anomaly detection [16].

In the system log at Axis we are observing unevenly spaced time series representing projects. In [23] they exploit logs to predict next event and timestamp in business processes using LSTM. The data used in their experiments has properties similar to the data at Axis if one refers to the lengths of the sequences and the irregular time interval between events. In [23] they show that LSTM outperforms existing techniques for their use case. Furthermore they arrange their data in the same manner, i.e. when training or predicting at a certain timestamp, all previous events are utilized. As a last motivation of our thesis and approach they believe their framework can be extended to predict case outcome in a business process. This work supports our early hypothesis about how the problem could be approached and inspired us to pursue the idea.

1.4 Structure

This thesis is structured as follows: Chapter 2 Introduces the concept of machine learning and some underlying theory required for understanding the implementation specific parts of the report. In Chapter 3 we explain our approach to the project based on initial knowledge and assumptions. Chapter 4 describes the methods we use for the data processing, network implementation and evaluation and the choices we made as it relates to this. In Chapter 5 we present our results and in Chapter 6 we evaluate them and discuss improvements before providing our final conclusion in chapter 7 along with potential future work.

1.5 Work distribution

The thesis took place at Axis communication headquarters in Lund, Sweden during the spring term of 2019. The work was a collaboration to which both parties contributed equally.

Chapter 2

Theory

In this chapter we provide some basic theory regarding machine learning as it relates to the techniques used in this thesis. We start with a light introduction to the field as a whole with emphasis on the concepts used in our work before taking a closer look at these in the later sections. Some prior knowledge in multivariate calculus, linear algebra and statistics is assumed.

2.1 Machine learning

All cognitive beings only know basic motor functions and other reflex behavior when they are born and need to learn from experience to understand the world they populate. Using our senses, we humans collect information from many different scenarios and thereby accumulate experience which we store by remembering the situation, our response to it and the result of that response. Through repetition we then gradually hone our skills on different tasks and skills, such as walking, until eventually we master them.

In the field of machine learning we mimic this biological approach to learning to teach computers to perform tasks, typically classification and regression tasks, by training them on examples from the environment. Classification entails letting the machine predict the most likely class from a set of classes that the data it is looking at can belong to, for example classifying whether an image contains a car or a dog. Regression means finding the function for the curve that best fits the data belonging to a certain distribution such that new data can be produced belonging to that same distribution. For example, regression could be used to find a curve that fits stock price development and be used to predict future stock price [18].

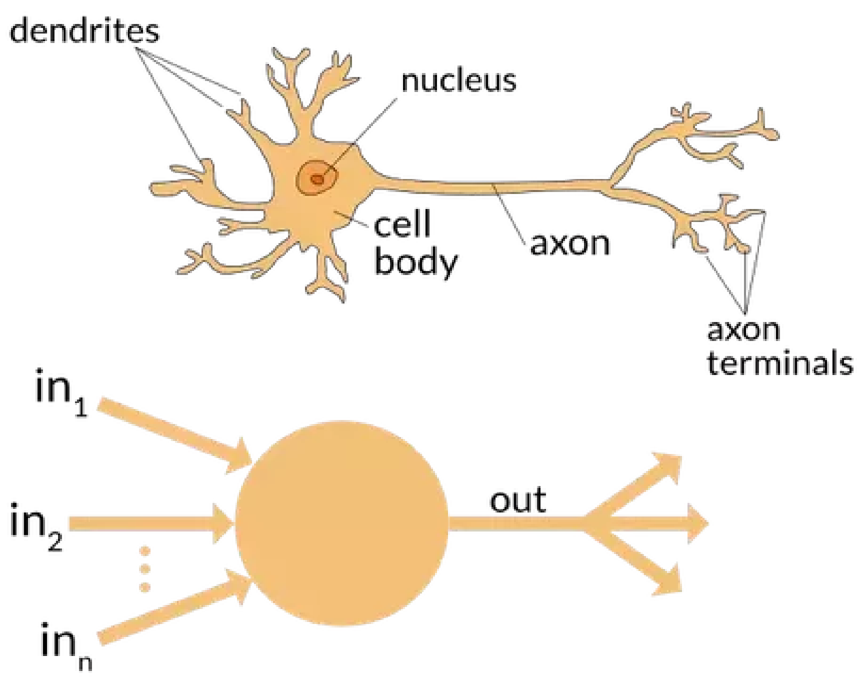


Figure 2.1: Comparison of the biological- and artificial neuron.[8]

2.1.1 Models

Classic machine learning utilizes models found in statistics such as Bayes' theorem and maximum likelihood estimation (MLE). These types of models are the foundations on which modern machine learning has been built upon. Bayes' model, defined as;

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{2.1}$$

often requires a considerable amount of data to get statistically significant estimates of the probabilities P in the model, save for the probability that we seek to find. MLE on the other hand requires knowledge about the underlying data distribution rather than a large data set. This is because MLE is commonly used to either estimate the mean and standard deviation of the distribution that produced the data, or to find the parameters for the polynomial of a certain degree that best fits the data. In either case, a good spread of just a few data points can still yield a good estimate of the underlying distribution.

The artificial neuron

Most modern machine learning models build upon the concepts so far discussed regarding taking inspiration from nature, of having lots of data available and finding parameters for a given model. The artificial neuron is arguably the most prominent such model, inspired by its biological counterpart, see Figure 2.1.

Very simplified the biological neuron consists of dendrites, soma (the cell body) and an axon. The dendrites receive inputs from axons of other neurons in the form of electrical signals and pass these along to the soma where signals from many dendrites (inputs) accumulate and are finally sent out to other neurons through the axon. In the artificial neuron the dendritic inputs are represented by one data sample with each dendrite corresponding to a feature of the data that is being analyzed. If the data is for example stock market data, then the features of that data could be ’today’s date’, ’share cost previous day’, ’total number of shares in stock’ etc. which together make up one data sample. Each input is associated with an updateable weight that decides the importance of the feature which in the biological neuron would equate to the strength of the signal received by each dendrite. The updating of the weights is the mechanism by which the artificial neuron ’learns’. All the weighted inputs are then summed up and the result passed through a special function called an activation function which produces the output of the artificial neuron corresponding to the function of the soma and the axon in the biological neuron.

Neural networks

There are many different implementations of the artificial neuron with the aforementioned perceptron being one of the earliest and most primitive of these consisting of just a single neuron. In later years, with algorithmic improvements to the neuron in addition to increased memory and computing power of computers, entire networks of connected neurons are built capable of learning much more complex problems, a model inspired by the brain. The simplest and most common of these ’neural networks’ is the multilayer perceptron (MLP) which consists of three or more fully connected layers of neurons, i.e. the output of every neuron in a layer connects to the input of all neurons in the next consecutive layer, see Figure 2.5. The MLP is a so called feed-forward network which means that information only propagates forward from layer to layer through the network. More complex networks, called recurrent neural networks (RNN), allow neurons to connect back to themselves or to neurons of previous layers. Due to this recurring propagation of data samples RNNs implement a simple version of a memory since they get to see the same data samples over and over again. The recurrent data is represented as a single feature separate from the data input features since it is the output of the same neuron or a neuron in a later layer, i.e. the weighted sum of all inputs passed through the activation function, see Figure 2.6a. The long short-term memory (LSTM) neural network used in this thesis is a special type of recurrent network that allows for more control over the memory aspect of the network.

Decision trees

The decision tree is another very common and simple machine learning model often used in an ensemble fashion called random forest. A decision tree is basically a one-directional tree-like flowchart of nodes starting with one ’root node’ and ending in several ’leaf nodes’ with each leaf containing an output of the tree. Decision trees are best illustrated with an example; let’s say we have a data set of people with the three features ’gender’, ’age’ and ’smoker’ and we want to classify the likelihood of a person getting cancer as either ’high’ or ’low’. In the root node of this tree we might split the data on ’smoker’, if the answer is yes then we might reach a leaf node with ’high’ and output this, see Figure 2.2. If the answer was no we reach a node that splits the data on ’gender’. No matter if the answer is

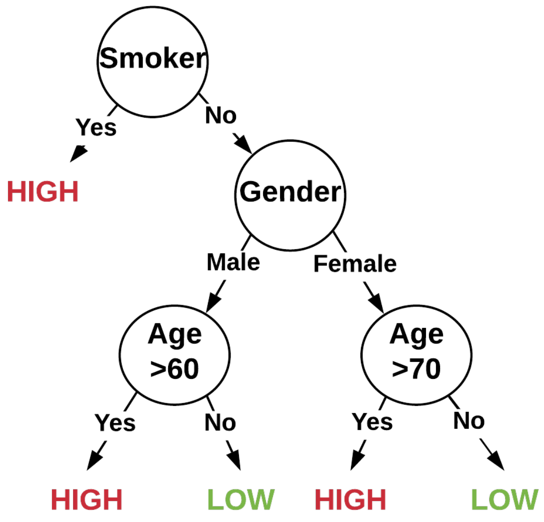


Figure 2.2: Illustrative example of a decision tree

'male' or 'female' we reach a new node which splits on 'age'. However these two nodes may split on different ages, perhaps 60 for men but 70 for women. If 'age' is greater than this threshold then we output 'high' otherwise 'low'. Choosing the feature to split on in each node and the limit to split at for the non-binary node 'age' is how the decision tree is trained.

2.1.2 Training

Training can basically be divided into two strategies, supervised- and unsupervised. The supervised strategy requires that each data sample is accompanied by a label containing a target output that the model should compare its predicted output to. By doing so it can figure out how bad its prediction was and update accordingly. The unsupervised learning strategy does not rely on labeled input data but can learn using other relationships present in the data [13]. The machine learning models used in this thesis are all trained using the supervised strategy, but before training can commence we need to have good data to train our models on.

Data

Acquiring good data is the main problem of machine learning because it is difficult and without it no model can be trained. If possible, machine learning problems are formed around existing data but if the data does not exist then it first needs to be gathered and once it has been collected it needs to be processed and moulded into numeric features such that it can be used by a model. Both of these tasks are highly problem specific and time-consuming, especially data gathering which can take years.

In general, the more data available for training the better since the model has more examples to learn from. In addition, the entire data set, including future data that the finished model will operate on, should originate from the same process or distribution. If it does not then we might train a good model on data with certain characteristics which ends up performing poorly on new data due to a change in these characteristics. For example, let us say that we have sales data between June and October for a retail store and wish to predict the sales of the following months. Sales during the months we know look quite similar so the model expects coming months to look the same but then Black Friday and Christmas comes around and the predictions are useless because of the sudden, never before seen spikes in sales. In this case, since Black Friday and Christmas are periodic- and thereby predictable events, the problem could have been avoided had we only had data from these periods from previous years, though this is not always the case.

Data splits

For models trained using the supervised strategy the data is often split into a training-, a test- and a validation set for evaluation purposes since we want to know how well a model is likely to perform when it is deployed. The size of the splits depends on the amount of available data but the training set is by far the largest and the test- and validation sets are usually the same size. What defines a large data set depends on the complexity of the problem. A complex problem may need many features which in turn requires a more complex model demanding more data to train. The training set is used to train the model and the test set to evaluate the model on. For the evaluation to be fair the integrity of the test set must not be compromised, thus the model cannot be trained on the data it is tested on. When training is complete we let the finished model make predictions on the test set samples and compare the predictions to the correct answers so that we can calculate an error measurement. There are several such error measurements, some of which are presented in the *Evaluation metrics* section below. The validation data is also used for evaluation but during training of neural networks for manual tuning of hyperparameters and discovering potential problems with overfitting and selection bias, concepts dealt with in *Neural networks* section below. Since the network models periodically see the validation set the evaluation becomes more and more biased from incorporating skill on the validation data into the models which is why the validation set cannot be part of the test set. For the same reason stated in the section above the different sets should also originate from the same distribution.

Data processing

Normally data is organized into indexed rows and descriptive columns of values such that each row is one sample of data. In unprocessed- or 'raw' data several of these columns may contain downright useless information. This could be due to the column being irrelevant for the problem at hand, the data in the column having little correlation with the target column or being too noisy etc. Machine learning is not a one size fits all approach but rather a blunt tool for finding relationships in data, meaning that the models we create cannot generalize particularly well to any arbitrary data. It is up to the data scientist to make it easier for the models to find the desired relationships by providing them with 'good' data, which requires the data to first be processed.

There is no one way to process data but common steps include data cleaning, feature engineering and normalization. Cleaning the data could entail identifying and removing bad columns or values, filling in gaps and converting non-numeric values into a numeric representation. Feature engineering is the most comprehensive step and can take considerable amount of time. Some data columns may be used as features immediately with minimal processing while others may require much more processing. Examples of processing could be combining several columns into one or splitting up the values in one column into several new ones, applying functions to the values of a column etc. After feature extraction and selection has been completed the data is usually normalized to have zero mean and unit variance so that all features are in similar ranges but the relative distance between data points of the same feature are preserved. This is done to provide the model with an unbiased feature set and let it learn for itself which features are important and which are not.

Overfitting

Ones the data has been processed then training can commence hopefully resulting in a good model that generalizes well to new unseen data. Unfortunately, no matter the processing, data will always contain some noise or uncertainty that is not representative of the true underlying model that generated the data in the first place. One common problem in machine learning arises when we use a too complex model with too many parameters to fit our training data. A capable model like this, would eventually not only fit the underlying dynamics of the true model, but also the noise. This is what is called overfitting and such a model would not generalize well to new, unseen data. This is often observed in practice, where the training performance increases and the test performance decreases as we increase the complexity of the model. The process to prevent overfitting is called regularization and some techniques to handle this will presented in Section 2.2.

2.2 Neural networks

2.2.1 The artificial neuron and the perceptron

Figure 2.3a illustrates the artificial neuron. The output y of the neuron is produced as follows:

$$y = \varphi(h) \tag{2.2}$$

where φ is an activation function and h the weighted sum of inputs to the neuron:

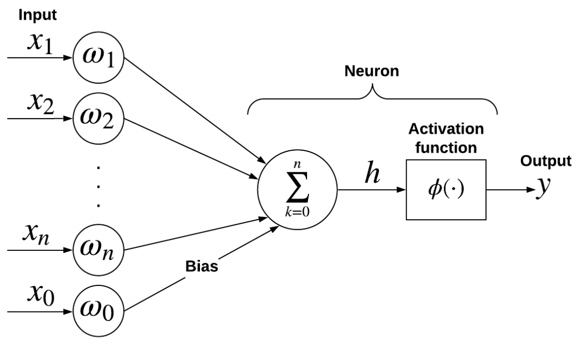
$$h = \sum_{k=1}^{n} x_k w_k + w_0 \tag{2.3}$$

where w_0 is a bias weight which allows the activation function to be shifted. If we introduce $x_0 = 1$ we can rewrite the output in vector notation as

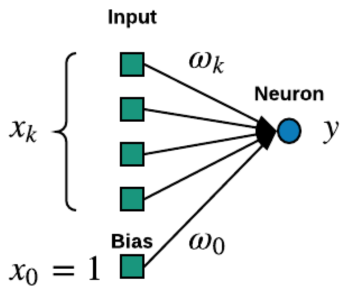
$$h = \sum_{k=0}^{n} x_k w_k = \mathbf{w} \cdot \mathbf{x} = \mathbf{w}^T \mathbf{x} \tag{2.4}$$

The neuron is defined as having two layers of nodes, the input layer and the output layer, and is the basic building block for all neural networks in which two or more neurons are connected in series [22]. For a network of two neurons in series the output of the first neuron is the input to the second. Thus, the entire network consists of three layers; the input layer which is the input to the first neuron, a hidden layer that is the output/input layer from the first to the second neuron, and the output layer, the output of the second neuron. This means that a network consisting of 3 single-neuron layers have 2 hidden layers etc. The depth of a network is usually defined by the amount of hidden layers it has got plus the output layer.

A construct consisting of a single neuron, i.e. having no hidden layer, is denoted a perceptron, see Figure 2.3b and is the most simple "network". The perceptron can perform binary classification and linear regression and implements a supervised learning algorithm which requires labeled training data, i.e. data where each input has a corresponding correct output. The perceptron, and indeed all neural networks, learn by updating the weights for every input, or batch of inputs, through minimization of a loss function. Minimizing the loss function is usually done in an on-line fashion by way of gradient descent, the details of which are presented in the *Loss function* section below.



(a) The simple neuron.



(b) The perceptron. The purpose of the bias node is to move the decision plane away from the origin by altering the bias weight, w_0. The bias input is always $x_0 = 1$.

Figure 2.3

Node activation function

This section is considering the perceptron network model from the previous section. The purpose of the node activation function is to map the weighted sum of the node inputs to an output suitable for the task at hand. However, there are many such functions to choose from so deciding on which one to use is problem dependent and requires some knowledge about each function. Neural networks are commonly used for solving classification problems, the simplest case of which is binary classification where an input can only belong to one of two classes. A function capable of solving this problem is the step function which outputs

1 if $x > 0$ else -1, see Equation 2.5 and Figure 2.4a.

$$\varphi(x) = \begin{cases} 1 \ \mathit{if}\, x > 0 \\ -1 \ \mathit{if}\, x \leq 0 \end{cases} \tag{2.5}$$

Note that the input, x, is a scalar composed of the weighted sum of the inputs including the bias. The activation function thereby determines a hyperplane in $\mathbf{x}$-space in which the output y changes sign when $\mathbf{w}^T\mathbf{x} = 0$. This equation describes a plane passing through the origin and it is the addition of the bias weight, see w_0 in Equation 2.3, that moves the plane away from the origin.

As previously mentioned networks are trained by minimizing a loss function using the gradient descent optimizing function which utilizes the gradient of the activation function. Since the step function has zero gradient it cannot be used for gradient descent learning. Sigmoid functions look like the step function and have similar properties with the addition of being differentiable and monotonic. In addition, their gradients are not monotonic but rather have a global extreme point. The most common sigmoid activation functions are the logistic function and the hyperbolic tangent, see Figures 2.4c and 2.4d. The logistic sigmoid is defined between 0 and 1 which makes it good for problems where a percentages output is desired. The hyperbolic tangent is defined between -1 and 1 so large negative numbers will be mapped towards -1 and big positive ones towards 1 while smaller negative and positive numbers will be mapped closer to 0 respectively. In contrast to the logistic sigmoid, which maps all negative numbers to 0, the hyperbolic tangent is superior in situations where this type of relationship between negative and positive numbers should be preserved. The hyperbolic tangent can be used for binary classification but is most commonly found in the intermediary hidden layers of deep neural networks [22].

Another problem commonly solved by neural nets is regression but since the step- and sigmoid functions are all bounded these cannot perform regression. To solve this problem a linear activation function is used, see Figure 2.4b. Linear activation functions are also differentiable and monotonic with a constant gradient so gradient descent can be used for learning.

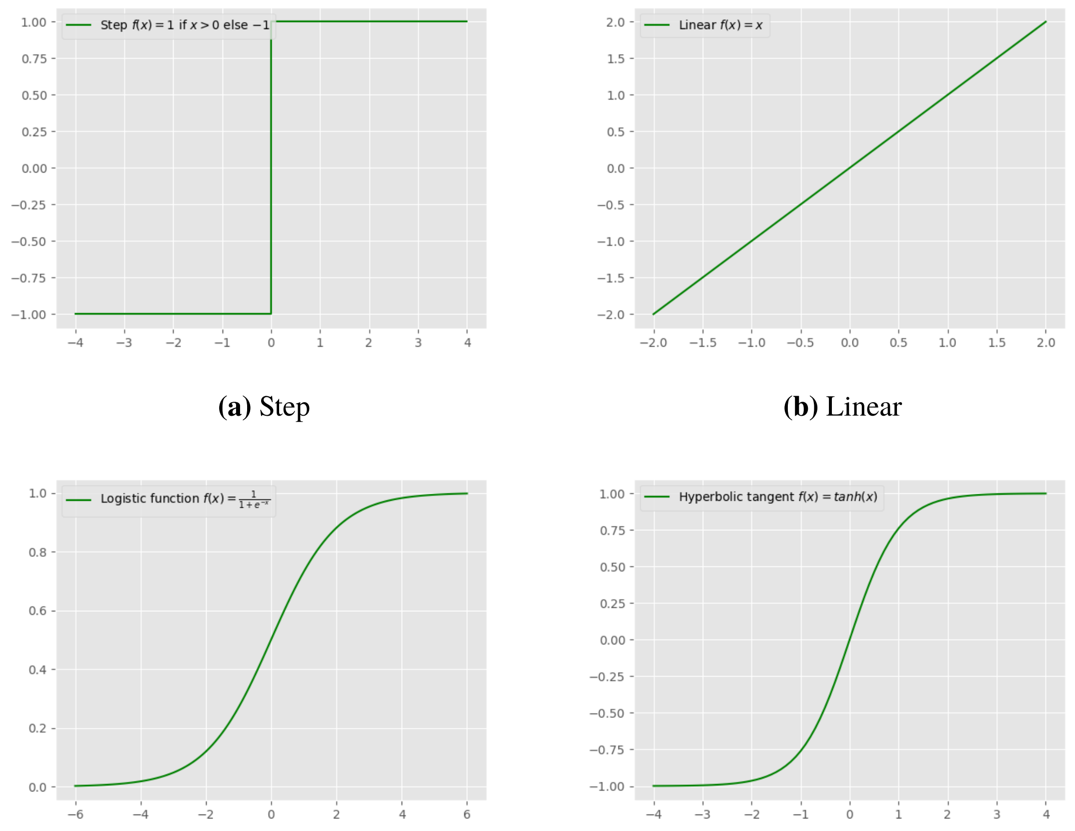
Loss function

Training a network means updating its weights so that the output produced matches the output target. To determine how good an output from the network was, a distance- or error measure like the squared error is required which we denote a loss function:

$$E(\mathbf{w}) = \sum_{n=1}^{N} (t_n - y(\mathbf{x}_n, \mathbf{w}))^2 \tag{2.6}$$

where n is the n:th input pattern, t_n and y are the target and network output for the input vector $\mathbf{x}_n$ respectively and $\mathbf{w}$ the weight vector. From Equation 2.2 we know that the output from a neuron is $y(\mathbf{x}, \mathbf{w}) = \varphi(\mathbf{w}^T\mathbf{x})$ and thus:

$$E(\mathbf{w}) = \sum_{n=1}^{N} (t_n - \varphi(\mathbf{w}^T\mathbf{x}_n))^2 \tag{2.7}$$



(a) Step **(b)** Linear **(c)** Logistic **(d)** Hyperbolic tangent

Figure 2.4: Examples of activation functions

To train the perceptron, **w** needs to be found such that this loss function is minimized. In some cases this can be solved analytically but for networks of nodes and nonlinear activation functions this becomes very difficult or even impossible which is why the gradient descent optimization schema is employed instead [22]. To begin deducing the gradient descent learning algorithm a normalizing term $\frac{1}{2N}$ is added to the loss function as:

$$E(\mathbf{w}) = \frac{1}{2N} \sum_{n=1}^{N} (t_n - \varphi(\mathbf{w}^T \mathbf{x}_n))^2 \tag{2.8}$$

$$= \frac{1}{2N} \sum_{n} \Big(t_n - \varphi(\sum_{k} w_k x_{nk})\Big)^2 \tag{2.9}$$

where k denotes a specific weight or input. As the name suggests the aim of the gradient descent algorithm is to move in the negative direction of the gradient in order to find a minimum for the loss function. The update rule summarize this for a single weight and can be expressed as:

$$\Delta w_k = -\mu \frac{\partial E(w)}{\partial w_k} \tag{2.10}$$

where μ is a step size that determines how far in the negative gradient direction to move, and needs to be chosen properly. The gradient of the loss function is:

$$\frac{\partial E(w)}{\partial w_k} = -\frac{1}{N}\sum_n \Big(t_n - \varphi(\sum_k w_k x_{nk})\Big) x_{nk}\varphi'(\sum_k w_k x_{nk}) \tag{2.11}$$

The weight update rule can then be written as:

$$\Delta w_k = \frac{\mu}{N}\sum_n \delta_n x_{nk} \tag{2.12}$$

where $\delta_n = (t_n - \varphi(\sum_k w_k x_{nk}))\varphi'(\sum_k w_k x_{nk})$. This update rule requires all input patterns x_n in the training set to calculate an update and is called offline batch updating. Usually an online approach is taken where the update is either done after each pattern or more commonly after a minibatch of patterns, P where $P \subset N$, resulting in the following update rule:

$$\Delta w_k = \frac{\mu}{P}\sum_{p=1}^{P} \delta_p x_{pk} \tag{2.13}$$

The full online gradient descent scheme for updating the neuron weights can be summarized in the following way:

```
1. Initiate all weights $w_k$ with small random numbers
2. Repeat until convergence:
    (a) Select minibatch of patterns $P$
    (b) Compute output $y(\mathbf{x}_p)$ and deltas $\delta_p$
    (c) Update weights $w_k$ as:
        $w_k \rightarrow w_k + \mu\delta_p x_{pk}$
```

2.2.2 Multilayer perceptron

The MLP is made up of an input- and output layer as well as one or more hidden layers, all fully connected and containing a bias node, see Figure 2.5. The main benefit of the MLP over the simple perceptron, which can 'only' solve regression- and binary classification problems, is that the MLP can solve any number of linear and nonlinear problems alike. In fact it has been shown that a single hidden layer MLP, using sigmoid activation functions can approximate any function, known as the universal approximation theorem [22]. However, the theorem says nothing about what a network that can compute a certain function should look like. There is also no one universal network that works for all problems, rather each problem requires its own modeling and tuning, often called the "*no free lunch theorem*" in machine learning literature. The hidden layer activation functions are always nonlinear since the linear combination of linear functions result in a linear function which can be implemented by the simple perceptron. The output layer activation function(s) are either linear for regression problems, logistic for binary classification or a version of of the logistic function called softmax for classification problems with more than two classes. The output from the MLP in Figure 2.5 for input pattern $\mathbf{x_n}$ is found by calculating a forward pass through the network as follows:

$$y_k(\mathbf{x_n}, \mathbf{w}) = \varphi_{out}\Big(\sum_j w_{jk} h_{nj}\Big) \quad \text{where} \tag{2.14}$$

$$h_{nj} = \varphi_{hid}\Big(\sum_i \hat{w}_{ij} x_{ni}\Big) \tag{2.15}$$

Expanded and written in one expression:

$$y_k(\mathbf{x_n}, \mathbf{w}) = \varphi_{out}\Big(\sum_j w_{jk}\varphi_{hid}\Big(\sum_i \hat{w}_{ij} x_{ni}\Big)\Big) = \varphi_{out}(\mathbf{w}_k^T \mathbf{h}_n) \tag{2.16}$$

To train the MLP each neuron's set of weights needs to be updated through the process of minimizing a loss function by way of gradient descent as described in the previous section. Although, as Equations 2.14 - 2.16 illustrate the output from a subsequent layer is a function of the output from the previous one and indeed all previous layers for networks with more than one hidden layer. There is a dependency between the layers which makes it slightly more complicated when calculating the gradients for updating the weights because a backward pass of information is required. To efficiently calculate the gradients in a neural network the backpropagation algorithm is often used.

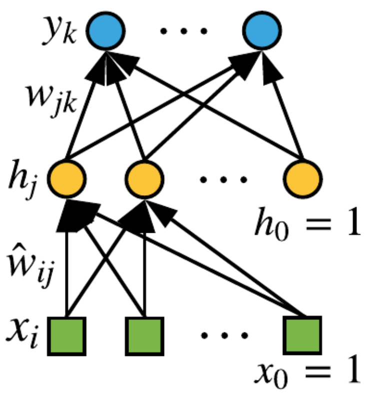


Figure 2.5: Example of an MLP with one hidden layer.

Backpropagation

Backpropagation is not very difficult to understand conceptually and the underlying mathematics are not very complicated if broken down and processed step by step, but jumping straight in may still be a bit daunting. In this section we will therefore first present a short conceptual description of how backpropagation works and then supply the mathematical explanation. In order to avoid very large expressions and have a visual aid to relate to, the two-layer network presented in Figure 2.5 will be used as the working example in the following two section.

Conceptual description

If we imagine that the network in Figure 2.5 should determine if the weather one day should be classified as 'sunny', 'overcast' or 'rainy' then we would have three blue output nodes, each representing one of the possible outcomes. The input to the network would be meteorological data for the given day, and during training this data would be labeled with a correct answer. We feed the network a training example with the correct answer being 'sunny' meaning that the blue output node corresponding to 'sunny' should output a 1 and the remaining two a 0. However, the network outputs 0.3 for 'sunny', 0.2 for 'overcast' and 0.5 for 'rainy' meaning that the network is obviously poorly trained. We want the 'sunny' node to increase and the other nodes to decrease, but not only that, the amount by which they increase or decrease should be directly proportional to how far off each output node was from its correct value. This means that it is more important to increase the output of the 'sunny' node than it is to decrease the output of the 'overcast' node since 0.3 is further from 1 than 0.2 is from 0, especially when the distance is squared.

Let us now focus on the left most node in Figure 2.5 and say that this represents the 'sunny' node. We want this node to become more activated, i.e. have an output closer to 1. Since the value of this node is calculated by passing the weighted sum of the nodes in the middle orange layer, the way we can affect the value of the 'sunny' node is either by changing the weights connecting it to the nodes in the orange layer or by changing the output value of the orange nodes. Focusing on the weights, it is important to note that modifications made to the values of weights connected to orange nodes with a high activation will have a greater impact on the activation of the 'sunny' node since the weights are multiplied by the output values of the orange nodes. If we now apply the same strategy to the other two nodes and note how much the activation of each orange node needs to change for each output node then, when we are done, each orange node will have three values saying how its activation should be altered. If we sum these three values up we get a collective 'best' value of what the activation of that orange node should be.

Now, the second way we can increase the activation of the 'sunny' node is to change the activation of the nodes in the orange layer in such a way that nodes connected to negative weights activate less and nodes connected to positive weights activate more. However, this can only be achieved indirectly by changing the weights between the green input layer and the orange layer, and since we cannot change the input values, this is also the only way in which we can affect the orange nodes. Since we know how much each orange node's activation needs to change in order to have the desired impact on the output nodes we simply need to change each weight between the green input layer and the orange layer accordingly by also considering the value of each input node. It should now be obvious that if we had had more than one hidden layer we would simply recursively apply this strategy for each preceding hidden layer until the input layer was reached, i.e. the calculation of what the activation of the nodes in each layer should be is propagated back through the layers, providing the algorithm with its name.

Mathematical explanation

Again using the normalized summed squared error as a loss function we have:

$$E(\mathbf{w}) = \frac{1}{2N} \sum_{n=1}^{N} \sum_{k} (t_{nk} - y_k(\mathbf{x_n}, \mathbf{w_k}))^2 \tag{2.17}$$

To train the network the following weight updates need to be calculated:

$$\text{input- to hidden layer gradient:} \qquad \Delta \hat{w}_{ij} = -\mu \frac{\partial E}{\partial \hat{w}_{ij}} \tag{2.18}$$

$$\text{hidden- to output layer gradient:} \qquad \Delta w_{jk} = -\mu \frac{\partial E}{\partial w_{jk}} \tag{2.19}$$

Starting by rewriting the error as a sum of per input pattern errors we get:

$$E(\mathbf{w}) = \frac{1}{2N} \sum_{n} E_n(\mathbf{w}) \tag{2.20}$$

Starting with the updates for the hidden- to output layer weights and utilizing the chain rule, we have:

$$\frac{\partial E}{\partial w_{jk}} = \frac{1}{2N} \sum_{n} \sum_{k'} \frac{\partial E_n}{\partial y_{k'}} \frac{\partial y_{k'}}{\partial w_{jk}} \tag{2.21}$$

and since $\partial y_{k'}/\partial w_{jk} = 0$ when $k' \neq k$ this reduces to:

$$\frac{\partial E}{\partial w_{jk}} = \frac{1}{2N} \sum_{n} \frac{\partial E_n}{\partial y_k} \frac{\partial y_k}{\partial w_{jk}} \tag{2.22}$$

where

$$\frac{\partial E_n}{\partial y_k} = \frac{\partial}{\partial y_k}(t_{nk} - y_k(\mathbf{x_n}, \mathbf{w_k}))^2 = -2(t_{nk} - y_k(\mathbf{x_n}, \mathbf{w_k})) \tag{2.23}$$

$$\frac{\partial y_k}{\partial w_{jk}} = \frac{\partial}{\partial w_{jk}} \varphi_{out}\Big(\sum_{j} w_{jk} h_{nj}\Big) = \varphi'_{out}\Big(\sum_{j'} w_{j'k} h_{nj'}\Big) h_{nj} \tag{2.24}$$

Now we can rewrite Equation 2.19 as:

$$\Delta w_{jk} = -\mu \frac{\partial E}{\partial w_{jk}} = \frac{\mu}{N} \sum_{n} \underbrace{(t_{nk} - y_k(\mathbf{x_n}, \mathbf{w_k}))\varphi'_{out}\Big(\sum_{j'} w_{j'k} h_{nj'}\Big)}_{\equiv \delta_{nk}} h_{nj} \tag{2.25}$$

$$= \frac{\mu}{N} \sum_{n} \delta_{nk} h_{nj} \tag{2.26}$$

which we recognize as the update rule for the perceptron from Equation 2.12. Moving on to the input- to hidden layer weight updates we have:

$$\frac{\partial E}{\partial \hat{w}_{ij}} = \frac{1}{2N} \sum_{n} \sum_{k} \frac{\partial E_n}{\partial y_k} \frac{\partial y_k}{\partial h_{nj}} \frac{\partial h_{nj}}{\partial \hat{w}_{ij}} \tag{2.27}$$

The first differential in this expression we have from Equation 2.23 and the second and third we find in a similar fashion:

$$\frac{\partial y_k}{\partial h_{nj}} = \frac{\partial}{\partial h_{nj}} \varphi_{out}\Big(\sum_j w_{jk} h_{nj}\Big) = \varphi'_{out}\Big(\sum_{j'} w_{j'k} h_{nj'}\Big) w_{jk} \tag{2.28}$$

$$\frac{\partial h_{nj}}{\partial \hat{w}_{ij}} = \frac{\partial}{\partial \hat{w}_{ij}} \varphi_{hid}\Big(\sum_i \hat{w}_{ij} x_{ni}\Big) = \varphi'_{hid}\Big(\sum_{i'} \hat{w}_{i'j} x_{ni'}\Big) x_{ni} \tag{2.29}$$

Now, using the delta defined in eqn. 2.25, eqn. 2.18, with the help of eqn. 2.27, can be written as:

$$\Delta \hat{w}_{ij} = -\mu \frac{\partial E}{\partial \hat{w}_{ij}} = \frac{\mu}{N} \sum_{n,k} \delta_{nk} w_{jk} \varphi'_{hid}\Big(\sum_{i'} \hat{w}_{i'j} x_{ni'}\Big) x_{ni} \tag{2.30}$$

and after defining a new delta as:

$$\delta_{nj} = \varphi'_{hid}\Big(\sum_{i'} \hat{w}_{i'j} x_{ni'}\Big) \sum_k \delta_{nk} w_{jk} \tag{2.31}$$

we finally get the weight update as:

$$\Delta \hat{w}_{ij} = \frac{\mu}{N} \sum_n \delta_{nj} x_{ni} \tag{2.32}$$

In summary we came to the following results for the MLP weight updates:

$$\text{input- to hidden layer gradient:} \qquad \Delta \hat{w}_{ij} = \frac{\mu}{N} \sum_n \delta_{nj} x_{ni} \tag{2.33}$$

$$\text{hidden- to output layer gradient:} \qquad \Delta w_{jk} = \frac{\mu}{N} \sum_n \delta_{nk} h_{nj} \tag{2.34}$$

For an online updating scheme, using input pattern P, we instead have:

$$\Delta \hat{w}_{ij} = \mu \delta_{pj} x_{pi} \tag{2.35}$$

$$\Delta w_{jk} = \mu \delta_{pk} h_{pj} \tag{2.36}$$

On further inspection we see that it is the deltas containing the gradients that are propagated back through the network. The backpropagation schema can of course be generalized to any number of layers [22].

Regularization

To prevent overfitting in neural networks we can decrease the number of layers and/or the number of hidden neurons in each layer. This technique directly restricts the model and makes it less prone to overfit the data. However, there is often an advantage to start with a too complex model that slightly overfits the training data and then apply more advanced regularization techniques. In addition to restricting the network architecture we will rely on two of these techniques to prevent overfitting in this thesis. The first one is called **Early stopping** and works by looking at the networks performance on the validation data during training and stopping training if no improvements are observed within a selected number of iterations. By stopping training early the model has hopefully trained enough to capture the dynamics of the true underlying model, but not enough to also fit the noisy parts of the data.

The second method is called **Dropout** and is a technique that randomly removes nodes during training. Before each new update of the network weights, a node in the network with its connecting weights are temporarily removed with a probability of p. This idea prevents the co-operation between weights to make sudden changes in the curvature of the decision boundary to bring in noisy samples, and thus limits the complex behaviour of the model making it more general [22].

2.2.3 Recurrent neural networks

Up to this point we have only considered feed-forward networks in which the data flows in a single direction from the input- to the output node. The output from the these types of networks are only dependent on the current input data and does not take into account previously classified samples. When working with sequential data there is often a benefit to model the dependencies between current output and input data from the past. RNNs adds this feature by introducing feedback connections in the network that can accumulate information from hidden- or output nodes from previous time steps [13].

Figure 2.6a shows an RNN network. Using matrix notation we see that $\mathbf{y}$ is the output vector, $\mathbf{h}$ is the vector of hidden neurons, $\mathbf{x}$ is the input vector and $\mathbf{U}$,$\mathbf{V}$ and $\mathbf{W}$ are weight matrices. The difference compared to the MLP is that we now have added feedback weights $\mathbf{W}$, that are connected from the hidden nodes back to themselves. If we omit the bias weight and introduce t as a sequence index, the network is described by

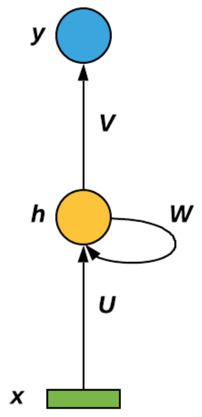
$$\mathbf{y}_t = \varphi_{out}(\mathbf{V}\mathbf{h}_t) \tag{2.37}$$

$$\mathbf{h}_t = \varphi_{hid}(\mathbf{U}\mathbf{x}_t + \mathbf{W}\mathbf{h}_{t-1}) \tag{2.38}$$

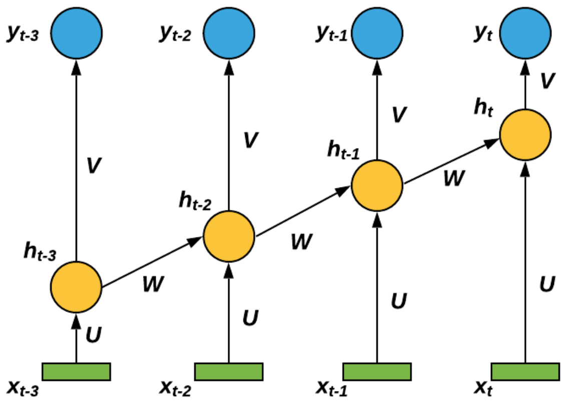
Figure 2.6b shows the same RNN network unfolded in time for 3 time steps. One important thing to notice is that we have the same weight matrix $\mathbf{W}$ for all the recurrent connections. This parameter sharing allows the network to take arbitrary sequence lengths as inputs and share statistical strength at different positions in time. The unfolded network graph has the same structure as an MLP with many layers and it is possible to train the network by doing normal backpropagation. This is called backpropagation through time (BPTT), since there is a new hidden layer for each time step, unfortunately this approach has some big drawbacks in practice. When updating the weights using BPTT we need to multiply gradients from several layers back in time and there is a high risk that this leads to multiplication of many small numbers which is called the vanishing gradient problem [13].

2.2.4 Long short-term memory

In practice, most successful machine learning models use gated RNNs to handle the problem of vanishing gradients mentioned in the previous section [13]. Here we describe one such architecture, namely the Long Short-Term Memory (LSTM) network, see Figure 2.7.



(a) RNN network



(b) RNN network unfolded in time

Figure 2.6: Unfolding the RNN network in (a) three time steps results in the MLP network in (b) which is not fully connected.

To handle long time dependencies in an efficient way, the hidden nodes from the simple RNN in the previous section are replaced with a memory cell to compute the values of $\mathbf{h}_t$. The LSTM cell keeps an internal memory c_t and has three gates that govern the flow of information in the cell. The three gates are called the input gate **i**, the forget gate **f** and the output gate **o** and they are computed as;

$$\mathbf{i} = \sigma(\mathbf{U}^i\mathbf{x}_t + \mathbf{W}^i\mathbf{h}_{t-1}) \tag{2.39}$$

$$\mathbf{f} = \sigma(\mathbf{U}^f\mathbf{x}_t + \mathbf{W}^f\mathbf{h}_{t-1}) \tag{2.40}$$

$$\mathbf{o} = \sigma(\mathbf{U}^o\mathbf{x}_t + \mathbf{W}^o\mathbf{h}_{t-1}) \tag{2.41}$$

where $(\mathbf{U}^i, \mathbf{U}^f, \mathbf{U}^o)$ and $(\mathbf{W}^i, \mathbf{W}^f, \mathbf{W}^o)$ are new matrices of trainable weights and σ is the logistic sigmoid function that limits a value between 0 and 1. First a candidate vector $\mathbf{c}_t^*$ of the internal memory is computed, similar to how $\mathbf{h}_t$ was computed for the simple RNN.

$$\mathbf{c}_t^* = \tanh(\mathbf{U}\mathbf{x}_t + \mathbf{W}\mathbf{h}_{t-1}) \tag{2.42}$$

The new internal memory $\mathbf{c}_t$ are then updated by filtering the candidate vector with the input gate $\mathbf{i}$ and adding the old internal memory filtered by the forget gate $\mathbf{f}$.

$$\mathbf{c}_t = \mathbf{c}_{t-1} \cdot \mathbf{f} + \mathbf{c}_t^* \cdot \mathbf{i} \tag{2.43}$$

Finally the output of the hidden nodes $\mathbf{h}_t$ are obtained by filtering $\tanh(\mathbf{c}_t)$ with the output gate $\mathbf{o}$.

$$\mathbf{h}_t = \tanh(\mathbf{c}_t) \cdot \mathbf{o} \tag{2.44}$$

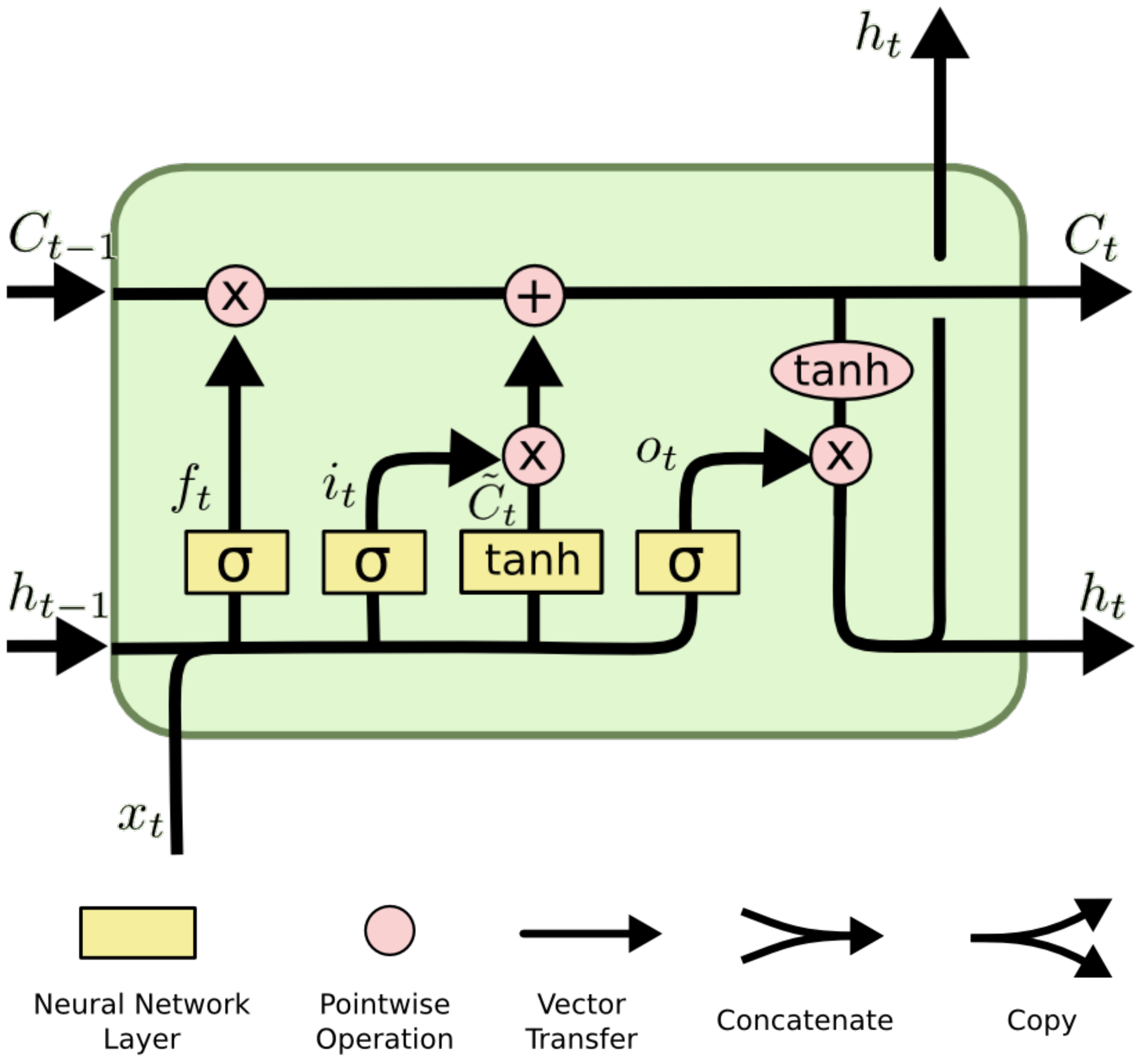
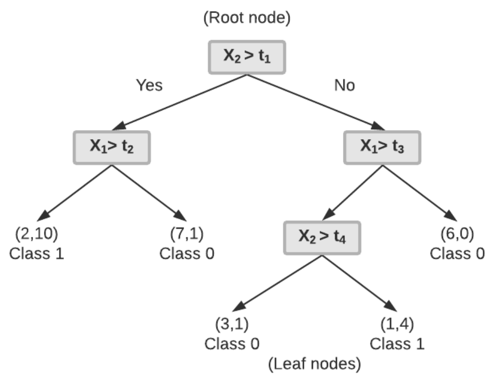


Figure 2.7: LSTM memory cell.

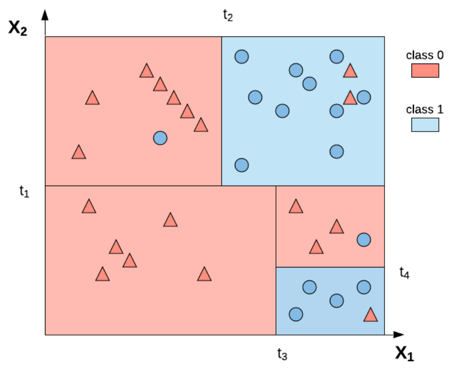
2.3 Decision trees and random forests

2.3.1 Decision trees

When traversing the tree in Figure 2.8a to make a predictions for the samples in the data we can view this as the decision tree making axis-parallel splits in the data set feature space in order to divide the training data into regions, see Figure 2.8b [20].



(a) Decision tree



(b) Decision boundaries

Figure 2.8: a) Decision tree for classification with two inputs. (n_0, n_1) in a leaf shows the counts for each class by following the path from the rote node to that leaf. b) Decision boundaries corresponding to the decision tree in a).

2.3.2 Growing trees

To construct a decision tree we have to decide which feature and threshold to use when splitting a node. When building decision trees it is common to use a greedy approach where the best feature and threshold for a split is computed locally for every node [20]. Let $\{\mathbf{x}_i, y_i\}$ be the feature vector and label for training sample i, where $x_{i,j}$ denotes the j:th feature. Starting from the root node, we recursively split every node into two child nodes. The best feature j and threshold t to split a certain node are obtained by minimizing the error function

$$E(j,t) = \text{cost}(\{\mathbf{x}_i, y_i : x_{i,j} \leq t\}) + \text{cost}(\{\mathbf{x}_i, y_i : x_{i,j} > t\}) \tag{2.45}$$

For regression problems we define the following cost function:

$$\text{cost}(\mathcal{D}) = \sum_{i \in \mathcal{D}} (y_i - \bar{y})^2 \tag{2.46}$$

where $\bar{y}$ is the mean of all y_i in $\mathcal{D}$. For classification problems we will use an error measurement called Gini index which is defined as;

$$\text{cost}(\mathcal{D}) = \sum_{c=1}^{C} \hat{\pi}_c (1 - \hat{\pi}_c) \tag{2.47}$$

where $\hat{\pi}_c$ is the probability that a random entry of the data $\mathcal{D}$ in a leaf belongs to class c. This is the expected probability of incorrectly classifying a sample. Note that the input data set $\mathcal{D}$ to the cost function is different for every split since we are partitioning the data into smaller and smaller subsets of the original data as we move down the tree. To avoid overfitting there is usually a stopping criteria that controls if a node is worth splitting. A few examples of stopping criteria are maximum tree depth, too few samples in child nodes or too small reduction in cost.

2.3.3 Random forest

Random forest is an ensemble method that utilizes the predictions from many decision trees. Ensemble methods are learning algorithms that base their predictions by taking a weighted sum from many estimates, i.e they combine the predictions from many training algorithms. This technique reduces the variance of the predictions, especially if the different predictors are uncorrelated [20]. Random forest tries to accomplish this by taking the average of many decision trees. To get uncorrelated predictors, random forest uses a new subset of the training data and a new randomly chosen subset of the features for every tree that is built.

2.4 Feature extraction

Since machine learning models are essentially function approximators they require numerical features to function which means that categorical features like text, somehow needs to be converted to numbers before they can be used. If the categories that we want to assign to feature have an ordinal relationship between each other we can simply convert k categories to numbers between 1 to k. This assignment implies that the model learning algorithm thinks certain values of a category are closer to each other than others, see Table 2.1. This assignment is not always possible to make, since very often there are no ordinal relationship between the values that can be assigned to a feature. To handle this, one-hot encoding and bin-counting can be used which are two different techniques that come with their own drawbacks and benefits.

Age
New
Old
Oldest

$\rightarrow$

Feature
1
2
3

Table 2.1: Coding of categorical feature *Age* with ordinal relationships, *New* is 'closer' to *Old* than *Oldest* so they are assigned values 1, 2, 3 respectively.

2.4.1 One-hot encoding

One-hot encoding takes all possible values in a category that should be assigned to a feature and creates a new feature for each value. If a category has k possible values, the new feature vector becomes a k-dimensional binary vector where a position has a 1 if the value is present, see Table (2.2). The benefits of this approach is that no information is lost. The downside is that the number of features can become very large and sometimes unmanageable to handle [9].

Category
Horse
Cat
Rhino

$\rightarrow$

Horse	Cat	Rhino
1	0	0
0	1	0
0	0	1

Table 2.2: One-hot encoding of the feature ’Category’.

2.4.2 Bin-counting

Bin counting replaces a categorical value with the historical statistics related to that value. If we have a target y and a categorical feature $x = c$, bin counting replaces c with the conditional probability $p(y = 1|x = c)$. This probability is estimated from the training data by counting the number of times $x = c$ and $y = 1$, divided by the total number of times c appears in the training set. The feature space remains the same for this method which sometimes makes it an obvious choice over one-hot encoding. The main drawback is that it requires historical data, if there are only one samples of several features these are mapped to the same number [9].

2.5 Evaluation metrics

Common metrics for classification are accuracy and F1-score. To derive the metrics we first define some terminology for classification problems.

- TP (true positives): number of positive samples that were correctly classified
- FP (false positives): number of negative samples that were incorrectly classified as positive
- TN (true negatives): number of negative samples that were correctly classified
- FN (false negative): number of positive samples that were incorrectly classified as positive
- Recall: proportion of all positive samples that were correctly classified - TP/(TP+FN)
- Precision: proportion of all samples classified as positives that were correctly classified - TP/(TP+FP)

Accuracy can now be defined as the proportion of all samples that were correctly classified.

$$\textbf{accuracy} = (\text{TP+TN})/(\text{TP+FP+TN+FN}) \tag{2.48}$$

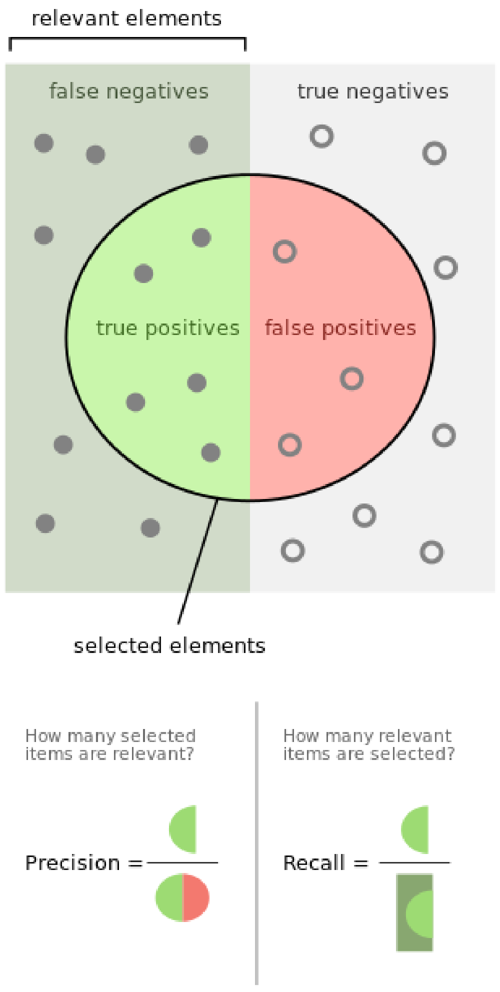


Figure 2.9: Precision and recall.

Accuracy is a common measure that is easy to understand but is not always a good measure when we have unbalanced classes since it is possible achieve high scores just by having high accuracy in one of the classes. The F1 score is often a more fair measurement that takes both classes into consideration and is defined as the harmonic mean between precision and recall.

$$\textbf{F1 score} = 2 * (\text{recall} * \text{precision})/(\text{recall} + \text{precision}) \tag{2.49}$$

The above measurements are used for evaluating predictions of class labels. If we instead want to evaluate the predicted probabilities of the class labels then the *cross entropy* measurement can be used which is also a common loss function used for training classification neural networks. In [22] a motivation for using cross entropy is made by taking a probabilistic approach to learning. First let d denote the output from the true underlying model that generates the data, $\mathbf{x}$ and y are the inputs and predicted output as before. We would like to know the probability distribution $P(d|\mathbf{x})$, i.e. for an input $\mathbf{x}$ we would like to know the probability that d belongs to a certain class. In binary classification d can only take the values $[0, 1]$. The conditional probability distribution can thus be modeled as a Bernoulli distribution

$$P(d|\mathbf{x}) = p^d(1-p)^{1-d} \tag{2.50}$$

where p is the probability that $d = 1$, i.e. the probability that $\mathbf{x}$ belongs to class 1. To estimate the unknown probability p we replace it by our predictive model $y(\mathbf{x}, \mathbf{w})$ as

$$P(d|\mathbf{x}) = y(\mathbf{x}, \mathbf{w})^d(1 - y(\mathbf{x}, \mathbf{w}))^{1-d} \tag{2.51}$$

In [22] it is shown that minimizing the average cross entropy is equivalent to the maximum likelihood estimation of y($\mathbf{x}$,$\mathbf{w}$), i.e. the y($\mathbf{x}$,$\mathbf{w}$) that maximizes the probability of our observed data under the model in Equation 2.51. Cross entropy is thus a highly motivated measure to use for evaluating estimated probabilities and is defined as;

$$\textbf{cross entropy error} = -\sum_n (d_n \log(y(\mathbf{x}_n, \mathbf{w}) + (1-d_n)\log(1 - y(\mathbf{x}_n, \mathbf{w}))) \tag{2.52}$$

where n denotes the sample index.

Chapter 3
Approach

3.1 Model selection

As explained in the *Problem formulation* Section we knew that we would be working with log data that could be moulded into time series data. Thus we decided to implement an LSTM neural network to solve the problem of providing an improved estimate of projects' probability of success since these networks have been shown to work well on time series data.

We would require a baseline estimate from a simpler ML model when evaluating the LSTM network so we chose to implement a logistic regressor for this purpose. In addition, we would require another type of network to contend with the LSTM one to evaluate its performance. To accommodate this need we decided to implement a random forest classifier and an MLP as the reference frame against which we could compare the LSTM network. We chose the MLP because in this setting, with time series data, it should be inferior to the LSTM. The random forest classifier we chose because it is a very popular-, completely different- and powerful model that wins many ML competitions and is relatively simple to implement.

The software tools we would use to build our networks were in a sense pre-determined since there are only a few languages that implement good ML libraries and of these the python language with the keras and sci-kit learn libraries is what we knew best.

3.2 Data processing

We knew that we would have a lot of data to process and would need to condense it, this would have to be done in steps, each one boiling off more data. Initially we would need to look at all the data and get a feel for what may be useful, save that and discard the rest. From the remaining data we would need to identify features or combinations thereof that could be deemed valuable by for example being correlated to project success rate. Finally we would have to process the data into a format that the networks could accept. Each step in this process would require processing the data in some way which from experience we knew to be a time-consuming task. Thus we would need to limit the amount of time we initially spend on finding good data so that we would have time to implement the networks and optimize them for the currently available data. Once functioning networks were in place we would need to re-iterate the data processing step and extract better features on which to retrain our networks on. We would repeat this process until we hit the deadline for this thesis project.

3.3 Evaluation

In order to evaluate our networks we would need to calculate some measurement of error. There are many ways in which to measure or estimate an error, with some being definitively better than others. It is also better to look at more than one type of measurement to get a more fair picture of the truth by mitigating biases and lowering measurement variance in this way. We chose to look at the accuracy and f1-score which are common metrics in this context that are used to evaluate the predictions of class labels. We also decided to look at the cross entropy error measurement to evaluate the estimated probabilities that a project would have to succeed.

Going into this project we only knew about Axis' interest in getting a proof of concept that ML could be applied to their log data and that estimating project success would be a possible and potentially valuable such proof. Showing that it would be possible would be the purpose of the networks, but to evaluate the added value an improved project success estimate would give to Axis we would first need to enquire about the following: What part of Axis' organization utilizes the project success estimate and to what extent? How would an improvement of this estimate affect these departments and/or the people working with it, could it result in a change in the way they perform their work, easing the workload etc.? To answer these questions we would need to interview relevant people at relevant departments, something we would have to investigate once the work started at Axis. Our plan going into the interviews would be to have an overarching line of questions prepared that we wanted answered whilst still leaving room for the interviewees to think and elaborate over the impact an improved project success estimate could have on their respective departments and the people working there. We prepared the following overarching questions that we wanted answered:

How does your department use the 'probability of success' field today, does it relate to any of the following topics?

- Ranking/prioritization of projects
- Resource planning
- Expected revenue prediction

Is the figure good enough to be of value today?

- If not, when would it be good enough and does it differ between use cases?

Can an improvement of the field be of use now or in the future?

- How?
- Could it change the way you work if a more reliable estimate is produced?

The questions were chosen to be relatively open ended and thought provoking so as to incentivize thoughtful answers and spark discussions since we would have limited knowledge about the interviewees' departments and how they use the project success rate field, meaning we would not be able to be too specific when preparing interview questions. Finally, we would need to record the interviews and later listen back and transcribe them at our own leisure. This would allow us to stay present and attentive during the interview and not be distracted from taking notes and enable a deeper post interview analysis. We planned to deliberately conduct the interviews in the later stages of the project so that we would hopefully have some real results from our networks we could present to the interviewees for them to relate to.

Chapter 4
Method

4.1 Data processing

4.1.1 Data extraction and partitioning

Axis uses a customer relationship management tool to track and manage sales also denoted ’projects’. In the graphical user interface of this tool, sales people and other stakeholders at Axis can update projects by editing fields. When a project is updated a corresponding table in a database, which we will refer to as ’Live’, is also updated. ’Live’ contains many tables related to projects that together describe the current status of a project. However, for the sake of traceability all updates to the tables in ’Live’ are stored in an ever expanding system log table located in its own database, see Figure 4.1. This log stretches back some 15 years and covers over 400.000 projects of varying size. Since the log is a single table it has a column indicating which table was updated and a column containing a stringified json-object of the altered fields in the table and their updated values, see Figure 4.2. In addition there are time- and id columns. It is from this log table that we built the data used in our networks.

We had to demarcate the scope of this thesis by limiting the number of tables used to three due to complexity- and time constraints. Including more tables would have meant more features which would have required more data pre-processing and feature engineering, both very time-consuming activities. In addition, the log table is over 100GB large so it took a long time to query. The three tables we chose to work with contain data about projects in general, products and stakeholders such as sales people, distributors and resellers. We started by rebuilding these three tables from the log such that they also contained all historical records of a project and not just its current state, see Figure 4.3. This broke the problem down into separate, more manageable entities with drastically reduced table sizes enabling much faster querying and easier testing of features to be built. We also performed an initial clean-up of the data during this extraction and partitioning phase by removing bad entries.

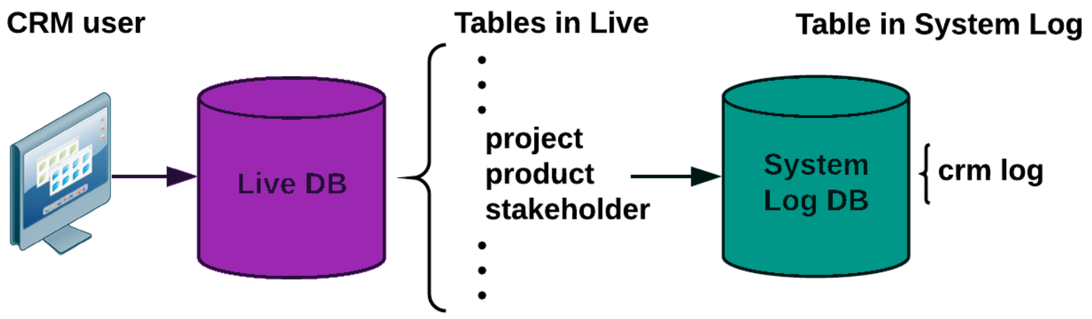
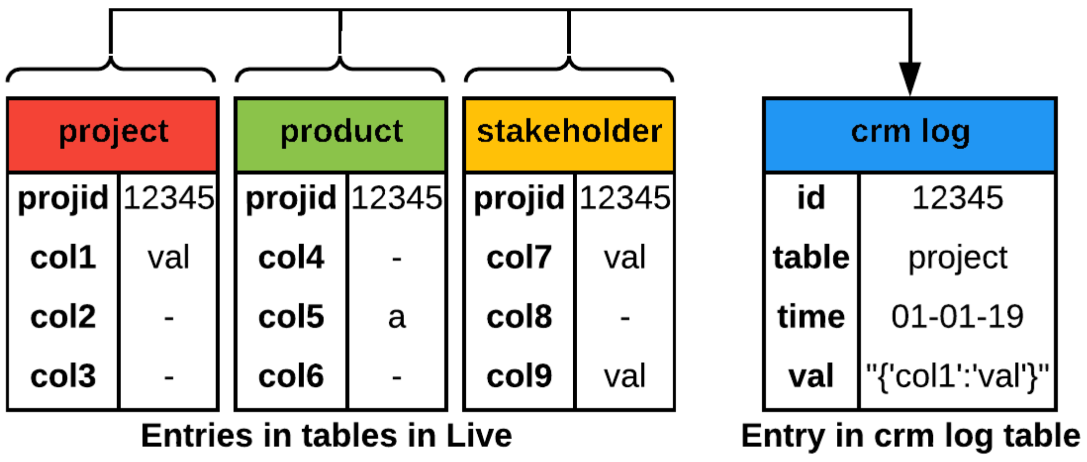


Figure 4.1: Data flow from CRM tool GUI to the tables in the Live database and onward to the System Log.



project	
projid	12345
col1	val
col2	-
col3	-

product	
projid	12345
col4	-
col5	a
col6	-

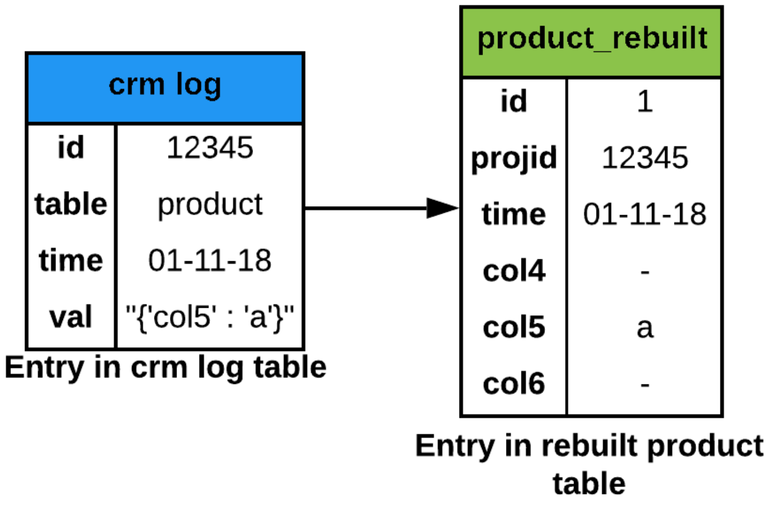
stakeholder	
projid	12345
col7	val
col8	-
col9	val

crm log	
id	12345
table	project
time	01-01-19
val	"{'col1':'val'}"

Figure 4.2: How data is logged from a table in the Live database to the CRM log table in the System Log, here an entry in the project table is being stored. Note that the project table in Live is represented by a 'table' column in the log table and that the data is saved in 'val' as a stringified json-object.

4.1.2 Data consolidation

At this point the project data was spread out over the three rebuilt tables, each containing all project updates concerning that table. The next step was to consolidate the data into one cohesive set. Before this could be done, the tables concerning products and stakeholders had to be modified to reflect changes in other products and stakeholders within the same project as these had been stored as separate insertions thus far. After these modifications, all entries for a product or stakeholder in a project were organized in chronological order so that each subsequent timestamp had all information about everything that had happened previously within the same project, for each table respectively, see Figure 4.4.

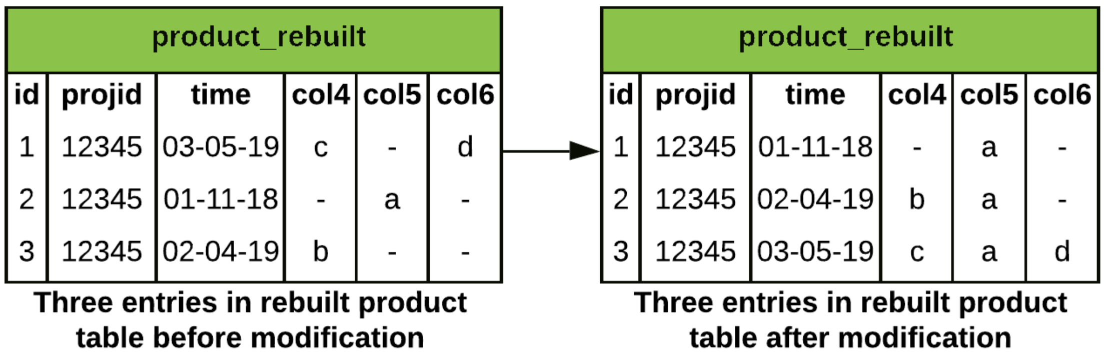


crm log	
id	12345
table	product
time	01-11-18
val	"{'col5' : 'a'}"

product_rebuilt	
id	1
projid	12345
time	01-11-18
col4	-
col5	a
col6	-

Figure 4.3: Rebuilding the product table so that it contains all the updates made to the original product table for every project.

To consolidate the three tables into one, we simply dropped irrelevant columns from each table before stacking them on top of each other. Now we had one table containing all columns from the three tables except for the columns that were dropped. Columns that did not exist in all tables now had NaN values or 'holes' for these columns. To fix this the table was sorted by project-id and time, in that order, and then iterated through for each project and values front filled, letting information flow forward in time. We had to do this on a a per-project basis to avoid leaking information from a project with a lower id into the project with the next project id. We now had a time series table with all information from the three original tables arranged in chronological order with each timestamp containing the information of all previous timestamps, see Figure 4.5.



product_rebuilt

id	projid	time	col4	col5	col6
1	12345	03-05-19	c	-	d
2	12345	01-11-18	-	a	-
3	12345	02-04-19	b	-	-

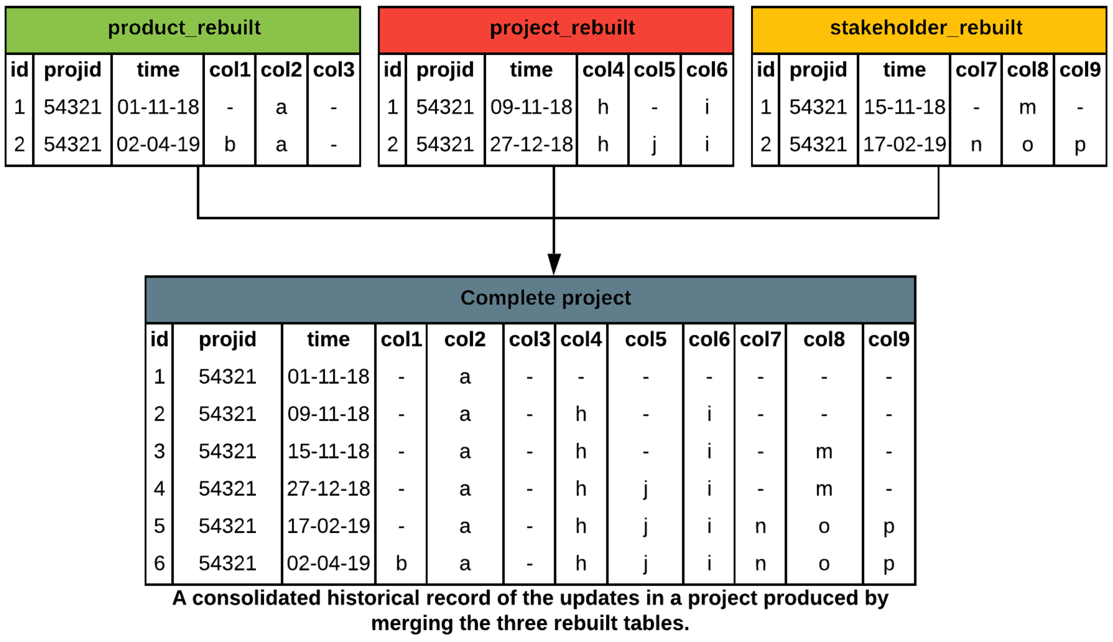
Three entries in rebuilt product table before modification

product_rebuilt

id	projid	time	col4	col5	col6
1	12345	01-11-18	-	a	-
2	12345	02-04-19	b	a	-
3	12345	03-05-19	c	a	d

Three entries in rebuilt product table after modification

Figure 4.4: Modifying the entries in the rebuilt product table so that information is being propagated forward in time. The same modification is applied to the rebuilt stakeholder table whereas the rebuilt project table already had this structure.



product_rebuilt

id	projid	time	col1	col2	col3
1	54321	01-11-18	-	a	-
2	54321	02-04-19	b	a	-

project_rebuilt

id	projid	time	col4	col5	col6
1	54321	09-11-18	h	-	i
2	54321	27-12-18	h	j	i

stakeholder_rebuilt

id	projid	time	col7	col8	col9
1	54321	15-11-18	-	m	-
2	54321	17-02-19	n	o	p

Complete project

id	projid	time	col1	col2	col3	col4	col5	col6	col7	col8	col9
1	54321	01-11-18	-	a	-	-	-	-	-	-	-
2	54321	09-11-18	-	a	-	h	-	i	-	-	-
3	54321	15-11-18	-	a	-	h	-	i	-	m	-
4	54321	27-12-18	-	a	-	h	j	i	-	m	-
5	54321	17-02-19	-	a	-	h	j	i	n	o	p
6	54321	02-04-19	b	a	-	h	j	i	n	o	p

A consolidated historical record of the updates in a project produced by merging the three rebuilt tables.

Figure 4.5: Consolidation of project data from the three rebuilt tables into one complete historical record of a project. Notice how the data is propagated forward in time.

4.1.3 Labeling

A project is successful if, according to CRM, a delivery is made. This is a somewhat strange definition since a project can remain in this state for a long time, as long as there are still products to be delivered. This also means that it is considered successful even if just one out of a thousand products was ever delivered. Though, we still chose to define a project as being successful as soon as it is considered successful according to CRM's definition in order to reduce the complexity of the problem substantially. Otherwise we would have needed to implement some other logic for when a project should be considered successful which would potentially be very hard and not in line with the CRM definition. In addition, the CRM definition of a successful project works well for most projects, since projects with a lot of orders and very few deliveries are rare outliers.

A project is considered unsuccessful if there has not been a delivery and it has been closed by a salesperson. This usually happens when Axis loses the project to a competitor.

In our data set we only used the projects that had reached the state successful or not according to the definitions above. This means that we removed all ongoing projects with no deliveries since the outcome of these projects were still unknown.

We wanted to predict the probability that a project would be successful or not at timestamps where no deliveries had taken place. Because of this we searched every project for the first timestamp indicating that the project was successful or not and removed all subsequent updates. We then supplied all remaining updates for that project with its appropriate target value, 1 if successful and 0 if unsuccessful, i.e. all timestamps in a project got the same target value.

Feature	Description
sales region	Categorical feature that has been one-hot encoded into sales regions.
res_stats	Bin-counting statistics for resellers.
dist_stats	Bin-counting statistics for distributors.
qty	Total amount of products in a project at the current update.
tot_price	Total price of all products in a project.
disc_tot_price	Total discounted price of all products in a project at the current update.
product_line	One-hot encoded feature, where each column specifies the amount of products belonging to a certain product line in a project at the current update.
demo	Binary feature that shows if the project has a demo product.
cfd	Binary feature that shows if a project has been confirmed.
del	Binary feature that shows if an update contains deleted products.
days_since_last	Days since last update in the project.
days_since_start	Days since the project was created.

Table 4.1: Description of features.

4.1.4 Feature extraction

Now that the data was in a consolidated and labeled state the feature extraction could begin. A description of the different features can be seen in Table 4.1. Some features were already extracted before the consolidation phase to enable the merging of the three tables. These features include *total_price*, *disc_total_price*, *qty*, *del* and *product_line*.

Product line is a one-hot encoded categorical feature. Products were originally stored in projects with their low-level product name that in detail describes the specifications of a product. To avoid thousands of one-hot encoded features, each corresponding to one product, the name of the product was replaced with the name of the product line it belonged to, of which there are only five. This was quite a dramatic abstraction and some product specific information was surely lost. However, the benefit was decreased complexity and noise reduction while also preserving a key feature, namely that the product lines correspond to different product affordability segments. For the same reason that product names were replaced by product line, the country names in which projects originate were also replaced by the sales region to which they belong, further lowering the feature space dimensionality required for one-hot encoding.

The last two categorical features of reseller- and distributor statistics were handled in a different manner. There were too many resellers and distributors to one-hot encode so instead we performed bin-counting for these two features. This meant using the statistics calculated from training data regarding resellers and distributors of a project, in this case their probability of success, instead of the categorical id value. All resellers or distributors that have been involved in less than ten projects were merged together in one bin (back-off bin) and share statistics. Ten is an arbitrary limit that can be changed but was chosen because it provided a reasonable split between the number of resellers and distributors involved in less than 10 projects versus those involved in more. The statistics calculated are stored in two files which the test data utilizes to convert from reseller or distributor to statistics. If a reseller or distributor is missing from the file, it receives the statistics of the back-off bin.

The time interval between updates in a project can be rather erratic with multiple updates happening in a short time frame between long periods of inactivity. Thus, the number of updates that have taken place in a project is not a good indicator of how long a project has been active. To provide the system with a mechanism for time, we needed to explicitly add the notion of time as features. To achieve this we made use of the timestamps in the log to compute the number of days a project has been in progress since the first- (*days_since_start*) and last update (*days_since_last*). This was performed after downsampling of the data set, see Section 4.1.5. The timestamp/date itself was however not used in the system since we did not want to take advantage of which years a project took place. Such a feature would not generalize to future projects. These features also let the MLP and random forest to utilize the time aspect although in a different way than the LSTM since each update is independent in these models, making the comparison more fair.

The correlation matrix of the features and project outcome (*target*) of the training data can be seen in Figure 4.6. In Table 4.2 we have trained a random forest classifier with 100 decision trees and listed the top 10 most important features. The score shows the mean decrease of the gini index provided by a feature, averaged over all trees in the ensemble. The countries have been analyzed separately since they have been one-hot encoded into several features that each make small contributions. After analyzing the correlation matrix and the random forest feature importance, we concluded that information like reseller, distributor and duration between updates seems to have higher predictive power than information about products, prices and quantities.

Feature	**Score**
days_since_last	0.210
res_stats	0.153
dist_stats	0.146
days_since_start	0.130
cfd	0.096
demo	0.077
tot_price	0.036
disc_tot_price	0.035
qty	0.033
P_line	0.018
nam	0.013

Table 4.2: Random forest feature importance.

4.1.5 Downsampling

The majority of the data in the log is created through human-computer interactions. A sales person might update a project by filling in one field at a time as he or she gathers

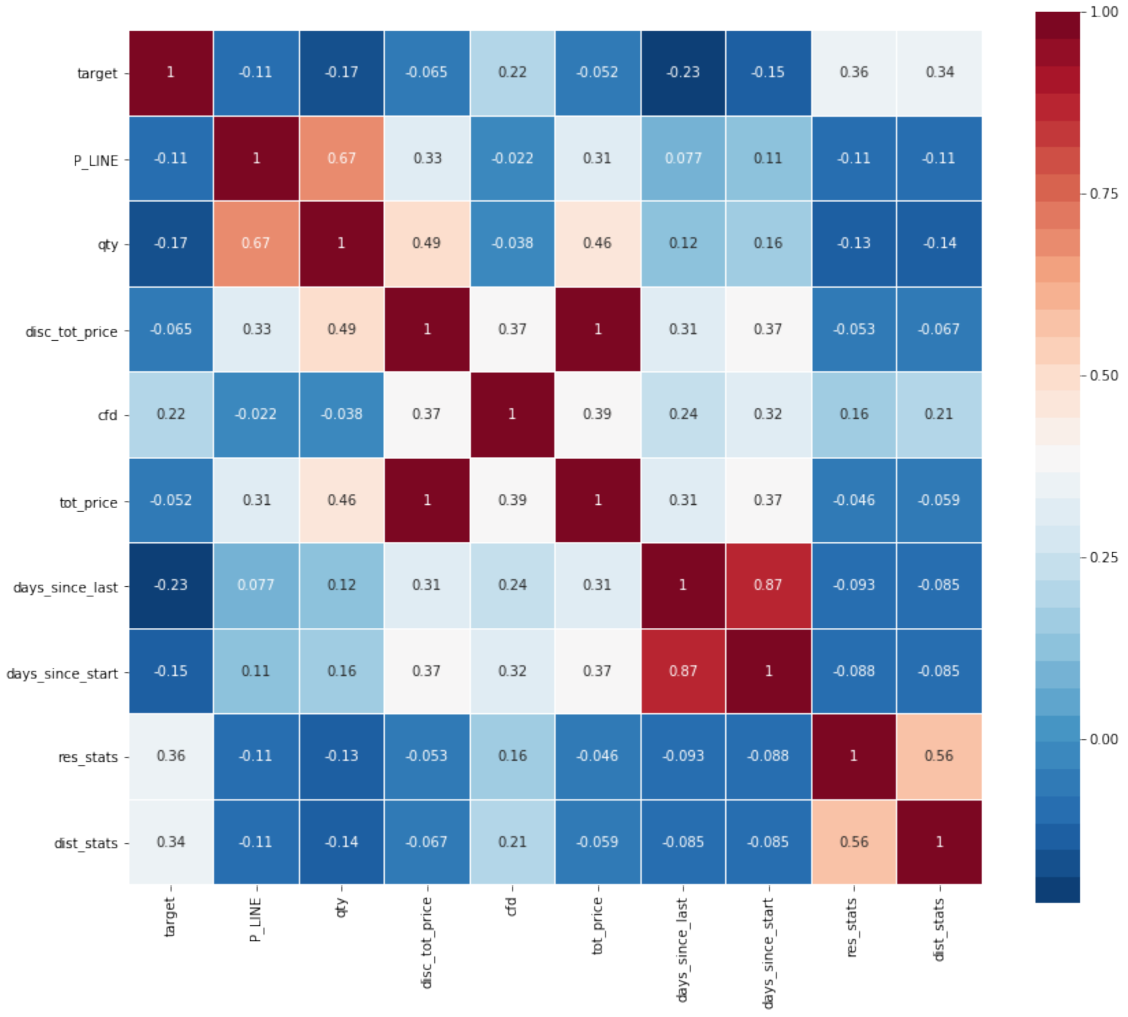


Figure 4.6: The Pearson correlation matrix.

information about a project during a workday and will inevitably make spelling- and other accidental mistakes. Since the system log captures all these interactions and creates new project updates every time an attribute changes these types of mistakes contribute a lot of noise to the data. Therefore, to represent the life cycle of a project, it makes more sense to observe a project on a somewhat longer timescale of days instead of hours, minutes or even seconds. To remove some of the effects of the noisy timestamps the data is downsampled. For each project, independently, updates were removed by sliding a window with a width of 24 hours over all project updates, sorted in chronological order. Only the last update from a pass of the window over the timestamps were kept which is a summary of all updates during the last 24 hours. This method can be seen as treating the fast changing noisy behavior as ongoing events in a project update and the update is not seen as done until there has been 24 hours of inactivity.

4.1.6 Data transformation for LSTM network

The last pre-processing step was to transform the data into a shape that could be fed to an LSTM network. For training and prediction the LSTM network takes an input feature vector of a project update along with all the previous updates associated with that project. Let $\mathbf{x}_{i,k}$ denote the feature column vector that represents the state of all attributes at time step k for project i. All information available at time step k for project i can then be described by the matrix

$$\mathbf{X}_{i,k} = \begin{bmatrix} \mathbf{x}_{i,0} & \mathbf{x}_{i,1} & \dots & \dots & \mathbf{x}_{i,k-1} & \mathbf{x}_{i,k} \end{bmatrix} \tag{4.1}$$

Hence we want our network f to model the function

$$f(\mathbf{X}_{i,k}) = y_i \tag{4.2}$$

where y_i is the outcome for project i. Since the prediction of a project outcome could take place anytime during a project, the network was trained for all possible combinations of (i, k), i.e. we created the matrix in Equation 4.1 for all updates in all finished projects. Each such sample was zero-padded to match the length of the project with the most amount of updates. This was not strictly necessary, but having the same shape for all samples allowed feeding the data in batches of samples for faster training. The transformation resulted in a 3D tensor with the added dimension of *time*. An example of the transformation for a project with five updates can be seen in Figure 4.7.

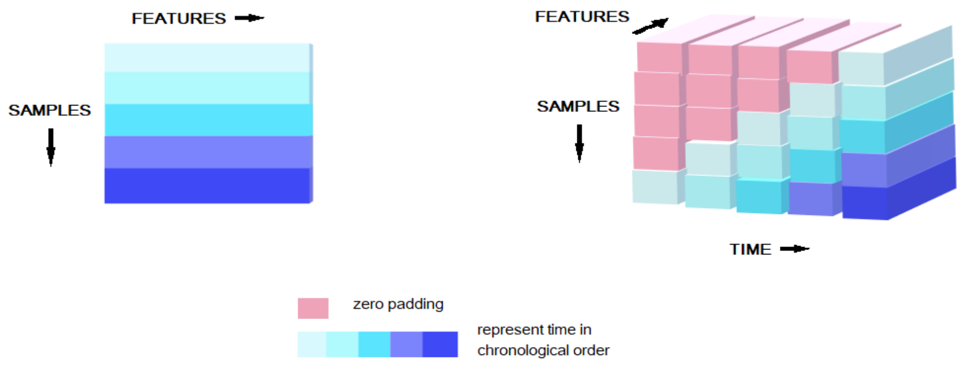


Figure 4.7: Transformation of data set(left) to a 3D-tensor(right) that can be processed by an LSTM network.

4.2 Experimental setup

In this section we motivate the model designs, data sets, metrics and baseline methods that were used for the evaluation. Axis wanted to test a dynamic system that could predict the probability of success anytime during an ongoing project. We therefore trained and tested on all updates during a project. We did this to be able to give accurate estimates both early on in the projects, where the information might be sparse and later in the projects where most of the fields had been set. The final data set, after pre-processing, contained 151551 projects with a total of 650428 updates (samples) which results in an average of 4.3 updates per project.

4.2.1 Training and test data

The models were trained on a training set containing 553317 samples from 126486 projects created between 2015/02/18 - 2018/05/24. The different models were evaluated on a test set that contained 97111 samples from 25065 projects created between 2018/05/24 - 2019/05/31. The training and test set had no projects in common. The two data sets were somewhat unbalanced, the ratio of successful/unsuccessful projects were 50/50 in the training set and 70/30 in the test set. Prior to the experiment we did try to balanced the training and test set. But since no difference was observed, we kept the original split, otherwise there would be a mix of old and new projects in the test set. Also, the F1 score still considers the accuracy on both won and lost projects so we know the network performs well on both metrics.

4.2.2 Model designs

As previously mentioned we used an MLP and a random forest to evaluate the LSTM network against and a logistic regressor as a baseline linear model. All the models were implemented in Python using open source libraries. The neural networks were created using the Tensorflow/Keras library with GPU support, and the scikit-learn library was used for the other machine learning methods [5]. The LSTM layers in Keras were implemented using a version with GPU support, see CuDNNLSTM in the Keras documentation [6].

The different models had several hyperparameters that needed to be tuned manually. To limit the scope of the experiment we took a heuristic approach where we trained several models prior to the experiment to determine some of the parameters. These parameters were then kept constant during the experiment, otherwise the computation time for testing all different combinations of parameters would have been unmanageable.

Random Forest

For the random forest classifier we tried different numbers of estimators/trees in the ensembles and we also varied the max depth of the trees, see Table 4.3. For the rest of the hyperparameters we used the default values, see the scikit-learn documentation [7].

Hyperparameters	
number of estimators	20-200
max depth/tree	15-30

Table 4.3: Hyperparameters for the random forest classifier.

Neural networks

RNNs can be structured in many different ways. We used a structure called many-to-one, where we took inputs from an arbitrary number of time steps and in the last layer only used the output from the last time step. If there were more than one hidden layer, the earlier layers would return the outputs from all hidden nodes as depicted in Figure 2.6b. The hyperparameters used can be seen in Table 4.4, where an epoch refers to one round of training using all the samples. For a more thorough explanation of the hyperparameters we refer to the Keras documentation [5].

Hyperparameters	**LSTM**	**MLP**
activation	tanh	tanh
recurrent activation	hard sigmoid	-
dropout	$p = 0.2$	$p = 0.2$
recurrent dropout	-	-
epochs	10	10
early stopping	patience = 3 epochs	patience = 3 epochs
batch size	128	128
step size	0.001	0.001
optimizer	adam	adam
loss function	cross entropy	cross entropy
neurons/layer	5-100	5-100
hidden layers	1-3	1-3

Table 4.4: Hyperparameters for the LSTM and MLP.

4.2.3 Metrics

The output of the models is the estimated probability of success for a project. This probability can be used directly or it can be rounded to perform binary classification. Because of these two use cases, we evaluated the models with three different metrics. As motivation for these metrics we refer to Section 2.5. Cross entropy was used to evaluate the predicted probabilities. Accuracy and F1 score were used to evaluate the models when used as classifiers. The F1 score in the result section is the weighted average of the F1 score for both classes.

4.3 Evaluation

4.3.1 Project's probability of success stakeholders

In our search for stakeholders of project's probability of success field we came in contact with two people from Axis' demand planning- and enterprise sales projects departments who were interested in our results. The demand planning department, abbreviated DP from here on, looks at trends in product sales over many small- and medium size projects and tries to predict demand in order to minimize lead times in the supply chain all the way from suppliers to customers. The enterprise sales projects department does essentially the same as demand planning but focuses on the much larger projects. Projects are subject to a procurement process during which products are added as the projects mature, but since projects can be changed or lost entirely at any given point the current total amount of products in all projects is a stochastic variable, dependant on the probability of success of the projects. A key difference between how the two departments work is that enterprise sales projects has a lot fewer projects and can stay in direct contact with the companies involved in the projects. Therefore they know how many products they will need at any given time whereas the demand planning department has too many projects to monitor to have a direct line of communication to all companies involved, making them dependant on trends and statistics rather than actual order figures. Thus, the demand planning department has the most to gain from the project success rate statistic which is why we mainly focused on their perspective during our interview.

We conducted a joint interview with Alexandra Wikström, sales- and operations planner at DP, and Anton Gustavsson, operations manager at the enterprise sales projects department. Going into the interview we had the overarching questions presented in the Approach Chapter. The interview was recorded and later transcribed in its entirety and then analyzed by going through the different questions discussed and answered. We followed up with Alexandra or Anton when questions arose and conveyed our findings and conclusions to them for verification which we present in the *Results Chapter* below.

Chapter 5
Results

5.1 Model results

Tables 5.1 and 5.2 show the results for the different evaluation metrics for the four models used. Figure 5.1 illustrates the evaluation metrics taken on a per update basis on the test set for the best overall model among the random forest classifier (RFC), MLP and LSTM models respectively. That is, the best model for each RFC, MLP and LSTM found in tables 5.1 and 5.2 were evaluated on the first-, second-, third-, etc. up to the tenth update separately in all projects in the test set. We chose to stop at the tenth update because there were too few projects with more than ten updates too provide a reliable metric.

In the following section we present the answers given by demand planning during our discussion around the questions presented in the *Evaluation* Section of the *Approach* Chapter.

	Trees	Max Depth	Cross Entropy	Accuracy	F1-score
RFC					
	20	15	0.448	0.812	0.819
	50	15	0.442	0.815	0.821
	100	15	0.442	0.813	0.819
	150	15	0.441	**0.818**	**0.824**
	200	15	0.443	0.815	0.821
	20	20	0.438	0.808	0.814
	50	20	0.438	0.810	0.817
	100	20	0.436	0.812	0.818
	150	20	0.433	0.814	0.820
	200	20	0.434	0.813	0.819
	20	25	0.448	0.797	0.804
	50	25	0.439	0.803	0.810
	100	25	0.435	0.807	0.813
	150	25	**0.432**	0.809	0.815
	200	25	0.434	0.808	0.815
	50	30	0.441	0.801	0.808
	100	30	0.437	0.802	0.809
	150	30	0.436	0.804	0.811
	200	30	0.438	0.802	0.810
LR					
			0.491	**0.769**	**0.778**

Table 5.1: Experimental results for the random forest classifier(RFC) and logistic regression(LR).

	Layers	N/Layer	Cross Entropy	Accuracy	F1-score
MLP					
	1	5	0.425	0.812	0.816
	1	10	0.411	0.820	0.823
	1	20	0.421	0.813	0.819
	1	30	0.418	0.813	0.818
	1	40	0.415	0.817	0.822
	1	50	**0.404**	**0.823**	**0.827**
	1	60	0.410	0.816	0.822
	1	70	0.411	0.818	0.823
	2	10	0.415	0.817	0.822
	2	20	0.423	0.812	0.817
	2	30	0.407	0.820	0.825
	2	40	0.428	0.802	0.809
	2	50	0.416	0.813	0.819
	2	60	0.416	0.816	0.821
	2	70	0.429	0.808	0.813
	2	100	0.417	0.811	0.817
	3	100	0.419	0.809	0.815
LSTM					
	1	5	0.408	0.818	0.823
	1	10	0.398	0.824	0.829
	1	20	0.405	0.818	0.824
	1	30	0.397	0.824	0.829
	1	40	0.413	0.814	0.820
	1	50	0.405	0.821	0.826
	1	60	0.406	0.820	0.825
	1	70	0.396	0.826	0.830
	2	10	0.396	0.824	0.829
	2	20	0.389	0.828	0.832
	2	30	**0.385**	0.829	0.833
	2	40	0.386	**0.831**	**0.835**
	2	50	0.397	0.823	0.828
	2	60	0.396	0.824	0.828
	2	70	0.387	0.830	0.834
	2	100	0.395	0.822	0.827
	3	100	0.390	0.826	0.831

Table 5.2: Experimental results for the Multilayer perceptron(MLP) and Long short-term memory(LSTM).

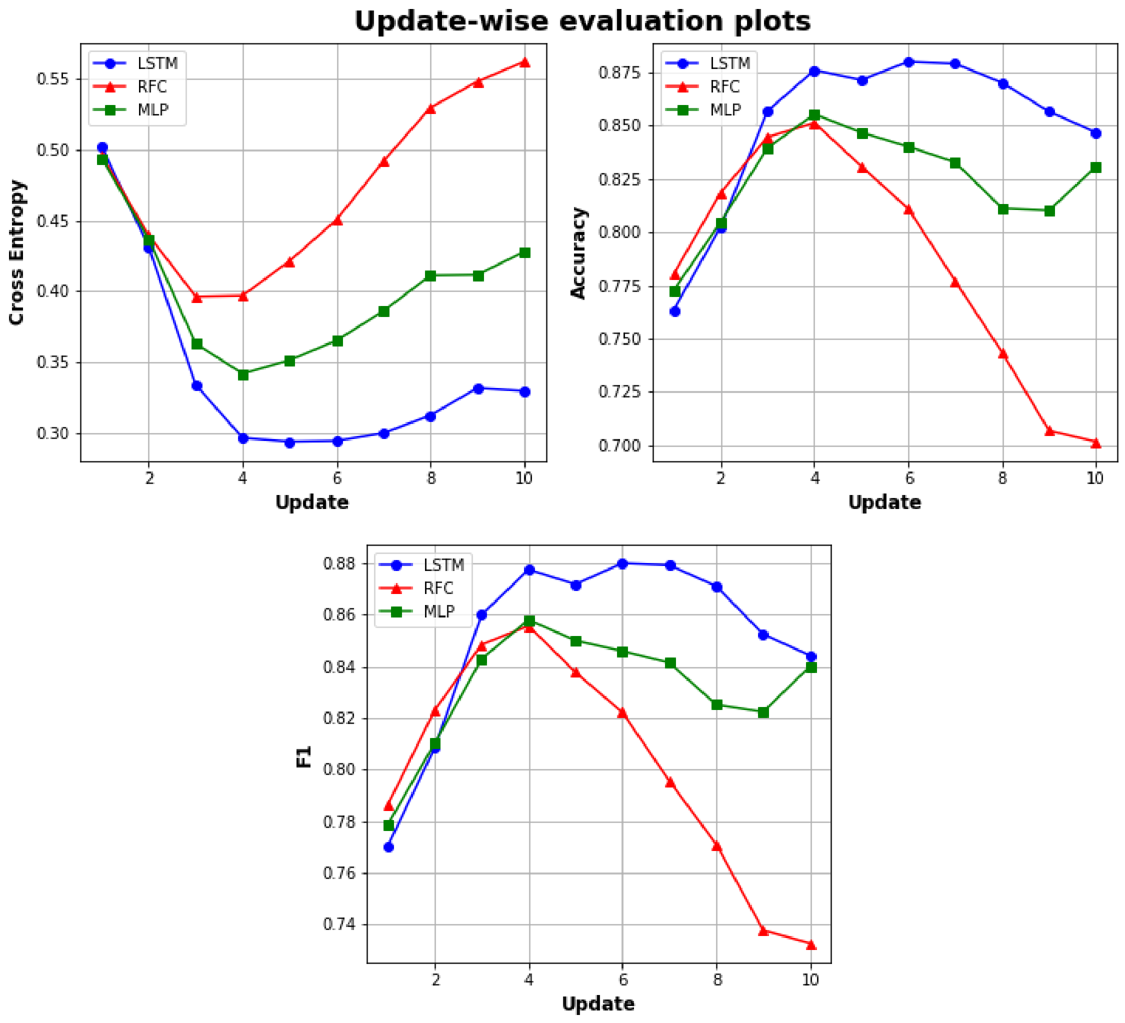


Figure 5.1: Plots of the different evaluation metrics cross entropy, accuracy and F1-score on a per-update basis.

5.2 Interview with Axis stakeholders

The demand planning department tries to predict what the demand for a certain product will be in the coming months in order to avoid bottlenecks in delivery. By looking at the product orders placed in projects they order parts such that the products are ready in time for when they need to be delivered. This means that DP wants to know, among other things, if a project will actually take delivery of the ordered products and when this delivery will take place.

The following answers come from the interview we held with Alexandra Wikström and Anton Gustavsson.

> How does demand planning use the 'probability of success' field today, does it relate to any of the following topics?
>
> – Ranking/prioritization of projects
> – Resource planning

   - Expected revenue prediction

Alexandra - *"We have not been able to use those probabilities in CRM because we have not been able to trust that field. Instead, we have built a process where we have contact persons in the different regions that can provide us with more reasonable estimates and a better flow of information."*

DP does not have any use of the field today but rather rely on other tools, techniques and people, beyond what is mentioned in the quote, to provide them with good estimates. Furthermore, DP's interpretation of the field is fundamentally different from the CRM definition. At CRM the field is interpreted as the probability that a project will be successful according to the best guess of the sales person in charge of the project. A successful project is in turn defined as at least one product being delivered. According to DP the 'probability of success' percentages should correlate to a project having reached a certain stage or milestone.

Anton - *"The probabilities are based on a 'quota'-process one could say. It also depends on other parameters like if there is any competition etc. So for example, if the probability of a project is initially set to 25%, I don't know exactly what that means, but then Axis at least knows about the project. At 50% Axis may be a top contender to get the project and if it reaches 75% then Axis is the 'favored' provider and there is probably a confirmed technical solution etc. At 90% then basically a contract has been signed and there may be product orders placed by the customer.*

A project is subject to a bidding process during which the project takes form as more details are unearthed the longer this process proceeds. The further into the bidding process a project has progressed the more information Axis has regarding the likelihood of them winning the project compared to the competition, which is the actual basis for setting the percentages. There are guidelines for how the percentages should be set depending on what stage in the bidding process a project is at. However, these guidelines differ between regions and are complied with in varying degrees and in the end it is a sales person who sets the percentage which introduces the human element of uncertainty. Therefore, the interpretation of 'probability of success', from DP's perspective, is more of a stage based system but one that is implemented differently across the organization, which is the main reason why DP cannot utilize this particular metric in their estimates.

Is the figure good enough to be of value today?

   - If not, when would it be good enough and does it differ between use cases?

The figure is not deemed good enough due to the high uncertainty in the estimate. Going forward, DP has plans to change the mechanics in which the field is set so that is not directly set by the sales person but rather set based on a number of criteria which would produce a more consistent figure. Due to the discrepancy between the regions regarding the interpretation of the field and the fact that in the end it is the sales persons perception of what the field means that the percentage is based on, no specific number of when the figure would be good enough could be provided. Rather, discussions around new potential use cases for the field based on CRM's interpretation of the field ensued.

Can an improvement of the field be of use now or in the future?

- How?
- Could it change the way you work if a more reliable estimate is produced?

DP sees several scenarios where a more reliable estimate of the 'probability of success' field could be very beneficial. Since Axis is a very large company there are a lot of projects in the pipeline at any one time. Individually tracking all these projects is not possible, so all but a small percentage of projects are bundled together and instead products are tracked over all these projects. This provides an overview of how many products will need to be delivered in the near future, what revenue they will potentially yield and what growth can be expected going forward for each product, and on a higher level, for Axis as a whole. Using CRM's perspective of the field as an actual percentage of whether or not a project will succeed, the tracking of products could be improved if each project in the bundle gets a reliable success classification. In addition, this type of information could give insight into trends in the different sales regions and also help mitigate demand peaks.

Anton - *Let's say that 3 months before a project will actually be won, CRM says there is a 50% chance of a successful deal but your prediction says 80%. Then we could take action based on your estimate because otherwise a month may pass and then CRM updates their number to 75% and normally then we would say, "let's go", but then we would have lost a month. So such indications would be beneficial, absolutely.*

Chapter 6
Evaluation

6.1 Logistic regression baseline

As we can see from Tables 5.1 and 5.2 the LSTM, MLP and RFC achieved higher accuracy and F1-score as well as lower entropy than the logistic regression which performed the worst. This was expected since it is the simplest and most straight forward method and was considered the baseline for our experiments. It still performed quite well achieving an entropy of 0.491, accuracy of 0.769 and F1-score of 0.778 which indicates that there are quite simple linear relationships in the data that account for much of the information needed to make a relatively good prediction. This is not all that surprising when looking at the correlation matrix in Figure 4.6 which shows several small- to medium linear correlations between features and the target. The feature importance table, Table 4.2, shows the most important features according to the RFC and considers nonlinear relationships as well. We can see that the correlation matrix and the feature importance table rank features differently, for example the *'days_since_last'* feature has the 5:th highest correlation but is considered the most important by the feature importance table. This indicates that there are nonlinear relationships in the data that the LR model couldn't find but the other three models could.

6.2 LSTM compared to the RFC and MLP

As we had hypothesized the LSTM outperformed both the RFC and MLP achieving better overall scores. At first glance it may seem like the differences between the scores are small and nontrivial but this might be an incorrect assumption. First we have to realize that not even the perfect classifier is capable of an expected accuracy of 100%, due to the stochastic nature of the problem. Behind the scenes there are human interactions and several other parameters not captured by the features that makes the outcome of a project uncertain. As an example, assume that we know that a project at a certain state has a true probability of 80% to succeed. We then know that whenever we see a project in this state in the future we will predict 'success'. If we now have a perfect test set that captures these statistics we will still be wrong 20% of the time even though we have a perfect model. Hence the best we can hope for is a model that makes the right decision all the time, i.e. a model that always predict the true expected output. With this in mind, we believe there is an upper bound for accuracy/F1 score and a lower bound for cross entropy that is possible to achieve. If it were possible to relate the results to this unknown upper/lower bounds for the three evaluation metrics, we believe that the differences between the models would be even clearer.

Applications and business goals would have to dictate which of the LSTM-, MLP- and RFC models are preferable in practice. LSTM shows greater results overall, and using the model in e.g. demand planning, the seemingly small differences could have a huge impact on business. In other use cases one could argue that the differences between the models are negligible and that the LSTM requires more memory and is more complex to implement and maintain.

In Figure 5.1 we see that for early updates the three models perform similarly but for projects with more updates the LSTM starts outperforming the MLP and random forest. This confirms that utilizing historical information and treating the projects as time series increased the performance when there are a lot of events in a project. These findings are not shown clearly in the test results, see Tables 5.1 and 5.2. The reason for this is that the average prediction approximately takes place on update 2.8 in a project, where prior information was limited and the capabilities of the LSTM were not fully utilized. After the preprocessing of the original data, the lengths of the time series were completely influenced by the number of updates the sales persons has made to a project. A different approach could be to dictate the lengths of the time series ourselves by choosing a different sampling method. This would give us longer time series but there is a risk of either getting a lot of redundant information or missing true updates made by the sales persons if the sampling method is not chosen carefully. This would have to be further investigated in a future project.

6.3 Interview

It turned out that the definition of the 'probability of success' field as described to us by CRM was vastly different from the actual definition that demand planning uses. According to CRM it is basically a percentage set by a sales person based on his or hers personal estimate of how likely the project is to succeed. The DP definition dictates that the percentage be set depending on where in the bidding process a project currently resides. When certain milestones are reached in this process the percentage should be set to a specific figure. Using percentages is an arbitrary and a somewhat confusing system to reflect this definition and a stage based system may have been clearer. However, even though it is not a direct percentage it still correlates quite well to one since the further into the bidding process Axis gets the higher the likelihood of winning a project should become. Considering this, our specific implementation could still be beneficial to DP as is, in two different ways; as a classifier and as a regression model. If used as a regression model the percentage produced by the network could be an aid when estimating product volumes which in turn could be used to mitigate logistical bottlenecks and to estimate revenue and growth. On a higher level, using the binary won/lost classification, information about the different sales regions could be extracted. Since business is conducted very differently around the world, in different business cultures, even within Axis' own organization and especially between Axis and its affiliates, anticipating and/or finding patterns in the way Axis' business is affected in different regions is an important but difficult task.

CRM considers a project to be won as soon as a product is delivered, DP has no such clear definition and are not really interested in one either. To them, what matters is the amount of products that will be delivered and when. What our network does is predict whether or not a project will succeed which is the same as if at least one product will be delivered. Expanding upon our implementation, it should be possible to predict not only the first delivery, but subsequent deliveries as well and also when they are likely to take place. However, this is a much harder problem, although not entirely unexplored using LSTMs [23][21] or potentially convolutional neural networks [10].

6.4 Error sources

A problem with projects, sales and most other real world systems, specifically in business are that they are not stationary which means that the statistics of a system changes over time. For example the distribution of the features or targets might not look the same in the future. To avoid this effect the model should be re-trained regularly with a new training set, where some of the old samples have been replaced by newer ones. The results of the test set would therefore only be valid for a certain amount of time, depending on how fast the data shifts, so the performance of future models would have to be monitored closely to quickly discover shifts.

Chapter 7
Conclusion

In this thesis we have found that it is indeed possible to estimate the probability that a project will be successful to a meaningful degree. We find our results better than we initially thought possible due to the seemingly complex nature of the problem and the fact that the estimates provided by the sales people were as bad as random guesses. The information about projects in the CRM database is rich and there are a relatively strong correlations between project attributes and the outcome of the projects. Performance of the models are similar in the initial phase of the projects, but LSTM networks outperform the other methods in this task as the sequences of project updates increases when projects move into to the later stages of the sales process.

The work provides a proof of concept that the project data can successfully predict the probability of success for projects. Furthermore the data pipeline we created can serve as a base for future predictive tasks at Axis using CRM's log data.

7.1 Future work

Future work directly related to this project could be to try out different sampling strategies and to more thoroughly investigate other tables in the database related to projects. Products could be transformed to a continuous attribute by training an embedding layer, where similar products are trained to be represented close to each other in a lower dimensional space. The benefit of using embeddings is that such a representation could potentially be used for other tasks related to products. Finally we also believe it is possible to change the problem formulation and extend the architectures to also estimate product demand and project duration.

Bibliography

[1] https://en.wikipedia.org/wiki/Thomas_Bayes[2019-07-18].

[2] https://www.bbc.com/timelines/zypd97h[2019-07-18].

[3] https://www.bbc.com/timelines/zypd97h#zmnvgdm[2019-07-18].

[4] https://www.doc.ic.ac.uk/~jce317/history-machine-learning.html[2019-07-18].

[5] http://www.keras.io[06-03-2019].

[6] https://keras.io/layers/recurrent/[2019-06-11].

[7] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html[2019-06-11].

[8] What is the differences between artificial neural network (computer science) and biological neural network? https://www.quora.com/What-is-the-differences-between-artificial-neural-network-computer-science. Accessed: 2019-07-26.

[9] A.Casari A. Zheng. *Feature Engineering for Machine Learning*. O'Reilly Media, Inc., 2018.

[10] Jiang L. Al-Jebrni A., Cai H. *Predicting the Next Process Event Using Convolutional Neural Networks*. In *proceedings of the 2018 IEEE International Conference on Progress in Informatics and Computing(PIC).*

[11] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. *End to End Learning for Self-Driving Cars*. *arXiv:1604.07316v1[cs.CV]*, 25 Apr 2016.

[12] D. Yu G. Dahl A. Mohamed N. Jaitly A. Senior V. Vanhoucke P. Nguyen T. Sainath G. Hinton, L. Deng and B. Kingsbury. *Deep neural networks for acoustic modeling in speech recognition.* IEEE Signal Processing Magazine, 29(6), 2012.

[13] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[14] QV Le I Sutskever, O Vinyals. *Sequence to Sequence Learning with Neural Networks.* In *proceedings of the conference, Neural Information Processing Systems* (NIPS 2014).

[15] D. Heckerman J. Goodman and R. Rounthwaite. *Stopping Spam What can be done to stanch the flood of junk e-mail messages?* Scientific American, 292(4), 2005.

[16] Lovekesh; Shroff Gautam; Agarwal Puneet Malhotra, Pankaj; Vig. *Long Short Term Memory Networks for Anomaly Detection in Time Series.* In *proceedings of the European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning — ESANN*, 2015.

[17] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, Dec 1943.

[18] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. The MIT Press, 2012.

[19] Paul Mozur. One month, 500,000 face scans: How china is using a.i. to profile a minority. `https://www.nytimes.com/2019/04/14/technology/china-surveillance-artificial-intelligence-racial-profiling.html`, 2019.

[20] Kevin P. Murphy. *Machine learning:a probabilistic perspective*. The MIT press, Cambridge, Massachusetts, 2012.

[21] Polate M. Sperduti A. Navarin N., Vincenzi B. *LSTM Networks for Data-Aware Remaining Time Prediction of Business Process Instances*. In *proceedings from 2017 IEEE Symposium Series on Computational Intelligence(SSCI)*, 2017.

[22] Mattias Ohlsson. *Lecture Notes on Introduction to Artificial Neural Networks and Deep Learning (FYTN14/EXTQ40).* Computational Biology and Biological Physics Department of Astronomy and Theoretical Physics Lund University, Fall 2018.

[23] La Rosa M.and Dumas M Tax N., Verenich I. *Predictive Business Process Monitoring with LSTM Neural Networks. In: Dubois E., Pohl K. (eds) Advanced Information Systems Engineering. CAiSE 2017. Lecture Notes in Computer Science, vol 10253.* Springer, Cham, 2017.

Appendices

Appendix A
About This Document

A.1 Acronyms

CRM:	Customer Relationship Management.
DP:	Demand planning department.
ML:	Machine learning.
LR:	Logistic regression.
LSTM:	Long short-term memory.
MLP:	Multilayer perceptron.
RFC:	Random forest classifier.
RNN:	Recursive neural networks.

A.2 Glossary

Stringified:	Converted into a textual representation i.e. a String object.

DEPARTMENT OF COMPUTER SCIENCE | LUNDS TEKNISKA HÖGSKOLA | PRESENTED 2020-03-12

THESIS Predicting successful sales opportunities from business log data using LSTM neural networks.
STUDENTS Robin Sauer, Marcus Månsson
SUPERVISORS Martin Höst (LTH), Robin Gustafsson (Axis)
EXAMINER Björn Regnell (LTH)

Mining for gold - Predicting sales outcome using machine learning.

POPULAR SCIENCE SUMMARY **Robin Sauer, Marcus Månsson**

In this modern age of computing machine learning is a popular tool many companies utilize throughout their business. In our thesis we developed a powerful neural network trained on historical sales log data to estimate the success rate of future sales.

Axis communications is a world leading company in the surveillance camera industry. Through its distributors Axis sells its cameras to resellers who in turn sell them to an end customer. The end customer can for example be an airport wanting to set up surveillance. The airport will ask for tenders from multiple companies, some of them Axis resellers, of a suite of cameras that cover the airport's needs along with the price. This exchange between customer and reseller is communicated back to Axis via a sales person and is recorded in what Axis denotes a *Project*. A project is basically a form that gets updated with more data as a project progresses. Projects contain data about products, prices, stakeholders etc. as well as the current project status which can be either ongoing, won or lost. All project information, including historical changes, are stored in a huge system log at Axis.

To a data scientist such a log is like a gold mine with the scientist being the miner. It requires a ton of work, using different tools, to dig out the vasts amounts of useless 'dirt' data which has to be processed and sifted through in order to find the rare and few golden nuggets of valuable information hiding throughout.

For our thesis we shouldered the role of miners of Axis' system log mine, using a machine learning model called an artificial neural network, more specifically a Long Short-Term neural network or just LSTM, as the main tool to dig up information gold. The artificial neural network is vaguely inspired by the biological neural network that constitute the brain. Much like the brain, neural networks learn to perform tasks from experience through practice. In our case we let our network gain experience by training it on historical won- and lost Axis projects to see if it would learn to correctly classify these, which it did! During the training phase we fed the network snapshots of projects and had it predict the outcome based on what the project looked like at that point in time. We then gave the network the correct answer so the it could adjust its future predictions if its current prediction was wrong. As a final test to see if our network had succeeded, we exposed it to previously unseen projects and calculated its accuracy based on how many times it correctly predicted a project as won or lost. We trained several networks this way and the best one we were able to produce achieved an accuracy of 83.1%.

The potential benefits to Axis of achieving good project predictions are at least two-fold; First, it serves as a proof of concept that it is possible to extract valuable information from the system log, which can serve as inspiration for future similar

THESIS Predicting successful sales opportunities from business log data using LSTM neural networks.
STUDENTS Robin Sauer, Marcus Månsson
SUPERVISORS Martin Höst (LTH), Robin Gustafsson (Axis)
EXAMINER Björn Regnell (LTH)

machine learning projects. Secondly, it can have a more direct impact for departments dealing with, for example, demand planning. Potentially allowing them to anticipate highs and lows in product demand and adjust supply accordingly, possibly saving the company a lot of money.