An Artificial Neural Network Approach to Within-Game Predictions in Football using Spatiotemporal Data

Stijn Römer

Master of Science Thesis

TU Delft Delft University of Technology

Delft Center for Systems and Control

An Artificial Neural Network Approach to Within-Game Predictions in Football using Spatiotemporal Data

MASTER OF SCIENCE THESIS

For the degree of Master of Science in Systems and Control at Delft University of Technology

Stijn Römer

April 24, 2022

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Abstract

Recent developments in Machine Learning (ML) have paved the way for unprecedented possibilities in the field of data analytics in numerous team sports, such as American football, baseball, and basketball. In more recent years, ML techniques have been applied to football as professional teams got inspired to collect enormous quantities of data to evaluate the performance of their players and their applied tactics. However, research in performance analytics has primarily focused on the development of offline evaluation techniques, while online evaluation of a game has remained largely unexplored. To attract the attention of executive decision-makers in football, research requires the creation of comprehensive metrics that can give answers to the continuous calls of coaches and trainers to evaluate within-game performance. This work therefore aimed at finding an online evaluation technique to capture the complex aspects and highly non-linear dynamics of the sport in an interpretable way.

To achieve this goal, an online predictive model was developed using an Artificial Neural Network (ANN) approach, which is able to give an instantaneous value to every moment in the game through the computation of the Expected Danger (ED). The ED is a comprehensive metric created in this work, which defines a form of relative, goal-related danger and encapsulates the complex spatio-temporal characteristics of the 22 players on the pitch. The ANN's have been trained and evaluated on two newly created datasets of simulated data which contain a combined total of 72 games and include the tracking data of the players and ball as well information on in-game events such as shots, passes, and goals. The extensiveness of the datasets enabled the creation of a novel Markov Decision Process (MDP) that describes the complicated dynamics of the system. The developed MDP model is a crucial component of this work and served as the basis for the design of the ED and allowed the creation of the ANN-based prediction model. The prediction pipeline was tested over five different prediction horizons, ranging from 5 to 45 minutes. Accurate predictive performance was achieved over all horizons, thereby showing the ANN architecture was able to account for all the nuances within the wide range of games of the datasets. The predictor functions were also tested on a small, additional dataset of unknown opponents to evaluate online performance and test robustness against unknown opponents. Experiments on this dataset demonstrated that reasonably accurate predictions can be made as long as the prediction horizon is not larger than 15 minutes.

Results achieved in this work show that a ANN-based approach to online predictive modelling can achieve accurate results. However, future research should focus on validating the performance of the ANN's on a more extensive dataset of unknown opponents and, ultimately, on a dataset of real-life football games. In addition, the online prediction pipeline of this work can be extended to an online decision-making model with the purpose of changing tactics dynamically during the game.

Contents

List of Figures

List of Tables

Preface

This work is the thesis project and final document of my master 'Systems and Control'. The idea of doing my thesis on this subject is inspired by a large list of suggestions from my supervisor, who apparently has a very wide range of interests. His suggestions included topics such swarming robots, cooperative ants, Lyapunov functions, automatic diabetes pumps, and the COVID-19 pandemic. However, one topic immediately attracted my attention, which was the topic of football. I have had a large interest in football for as long as I can remember, as I have played it a lot when I was younger, watched many professional games, and also played a lot of FIFA on the PlayStation. When I came to notice that football could actually be combined with the thesis project, my choice was easily made.

Fortunately, I was not alone in the creation of this work, and I would therefore like to thank the people who have assisted and supported me throughout the past year. Foremost, I would like to thank my supervisor Dr. Manuel Mazo for his support, his guidance, and his enthusiasm about the topic. The weekly meetings were always insightful and have helped me greatly on numerous occasions. I would also like to thank Daniel Jarne Ornia for his help and assistance. Our meetings have been really helpful and always excited me to continue with the project. Finally, I would like to thank my dear friend Lars van der Heijden, who has helped and guided me many times throughout the past year.

Delft, University of Technology — Stijn Römer
April 24, 2022

Glossary

List of Abbreviations

AI	Artificial Intelligence
ANN	Artificial Neural Network
ED	Expected Danger
EPV	Expected Possession Value
GRF	Google Research Football
HCT	Hough Circle Transform
KPI	Key Performance Indicator
K-S	Kolmogorov-Smirnov
MAE	Mean Absolute Error
MDP	Markov Decision Process
ML	Machine Learning
MLP	Multilayer Perceptron
MSE	Mean Squared Error
NN	Neural Network
POMDP	Partially Observable Markov Decision Process
ReLu	Rectified Linear Unit
RL	Reinforcement Learning
RMS	Root Mean Square
RMSE	Root Mean Squared Error
RNN	Recurrent Neural Network
STD	Standard Deviation

List of Symbols

α	Learning rate
$\bar{\theta}$	Average reward over all states in
β_1	Adam's decay rate 1
β_2	Adam's decay rate 2
ϵ	Adam's correction term
θ	Reward function
b	Biases of the ANN
ED^h	Expected danger function within time interval $[t+1, t+h]$
F	Set of regions on the pitch
f	Region
g	Activation function
G_t^h	Average reward within time interval $[t+1, t+h]$
h	Prediction horizon
n_h	Number of hidden neurons
n_x	Input size of ANN
R	Result matrix
S	Search image
T_{ball}	Template image of ball
T_{player}	Template image of player
W	Weights of the ANN
X	Input matrix of the ANN
Y	Output matrix of the ANN
$\mathscr{L}$	Loss function
Pr	Probability
θ_1	Reward function of user team
θ_2	Reward function of opponent
θ_t	Reward at time t
$\tilde{P}$	Set of updated state transition probabilities
$\tilde{S}$	Updated state set
$\tilde{s}$	Updated state
$\tilde{x}_t$	Updated state of the game at time t
A	Set of actions
a	Action (i.e., formation)
d	Danger value
F_d	Set of regions with the same danger value
m	Order of higher-order Markov chain
m_s	Number of examples in dataset

n	Size of state set
n_l	Number of hidden layers
n_r	Number of regions in F
n_s	Size of updated state set
n_y	Output size of ANN
n_θ	Number of sets of regions in F
P	Set of state transition probabilities
R	Set of rewards that refer to goals
S	Set of states
s	State
T	Template image
x	Input vector to ANN
x_c	Vector of contextual variables
x_p	Vector of player positions
x_t	State of the game at time t
x_{poss}	Value denoting the possession team

Chapter 1

Introduction

This chapter introduces the topic of research of this work. The relevancy of the area of research will first be discussed in section 1-1, which also highlights the potential of the field to be researched. Section 1-2 then formulates and explains the research questions and sub-questions. The chapter concludes with an overview of the structure of the report in section 1-3.

1-1 Problem and Relevance

Sports is a domain that has been growing consistently for the past decades and is of significant importance to many economies, while simultaneously influencing our social and cultural fabric. This importance is emphasized by a quantification of the size of the industry through a recent study from Kearney [1], which found that the global size of the sports industry is worth between €350 billion and €450 billion. Additionally, the growing rate of the sports industry is much higher than national gross domestic product (GDP) rates around the world, which proves this industry will likely stay economically interesting for the years to come. At the heart of the industry are the sports teams themselves, including the players, the coaches, and the staff. It is therefore not surprising that people have started thinking of ways to boost player and team performance within sports.

In the past, tactical analysis was based on observations dealt with by domain specialists, such as managers, coaches, scouts and other staff members, with limited analytical skills. However, the variables they use discard most contextual information. This gap can be filled through the use of data science, which exploits the knowledge hidden inside large datasets. Already, successful applications of data science within sports have been made. One of the first to adopt a successful statistical data analysis approach were the 2002 'Oakland Athletics' baseball team [1]. In 2002, they had the third-lowest team payroll in the league, with approximately $44 million in salary, compared to the New York Yankees $125 payroll that season. Through statistical data analysis, they found that historically important player statistics were often outdated, subjective, and flawed. By

[1] `https://www.nl.kearney.com/communications-media-technology/article?/a/the-sports-market` Date accessed: 12-01-2022

identifying the most important statistics, Oakland was able to find and buy the most undervalued players on the market. Despite their significantly smaller budget, this approach led them to consecutive playoffs places in both 2002 and 2003. A more recent example is showcased by the success of football club Leicester City, a team who were in the third tier of English football in 2009 and spend the majority of the 2014-2015 season battling for bottom place. However, a remarkable run in the 2015-2016 season lead them to win the league title, as their data analytics and sports science departments were the most advanced in the league [2]. During their champions season, Leicester City used various advanced data and analytical tools in combination with wearable GPS technology aiming to train and play optimally, and find the best-performing strategies. Success stories like Oakland Athletics and Leicester City have promoted technological and data-driven approaches by sports clubs. Even though there are many uncertainties that contribute to the outcomes of matches, it seems that assistive algorithms in sports can be very beneficial for teams. These algorithms may assist in various decision-making processes, ranging from the selection of players, in-game tactics, player transfers, and the planning of training sessions. However, the first step in making such decisions is through the creation of predictive algorithms able to model the likely outcomes of games and within-game plays.

In recent years, a large amount of interest has been shown in predictive analysis within sports, including baseball [3], American football [4][5], basketball [6], and football [7] [8] [9] [10] [11]. However, contrary to these other sports, data in football has only been collected systematically in the past decades [12]. It is therefore not surprising that predictive modelling within football is still in its infancy. Several reasons have contributed to this late arrival. Traditionally, football has always been regarded as the sport that cannot be modelled due to its complex nature and less controllable settings [13]. The nature of the sport is highly dynamic, the pitch is large, it is played by numerous players, the patterns in plays are highly complex, and the game sequences are long without any interruptions. As a result, teams and big companies have only just started collecting football data systematically. Current techniques are able to capture a large variety of data, which includes: the position tracking data of players, health statistics, and individual player performance statistics during games [14]. Data gets collected through state-of-art technologies developed by big, commercial companies. An example is the Apex Pro Pod wearable tracking device, able to transfer 260+ metrics real-time [2] , thereby allowing the creation of enormous datasets.

From a scientific perspective, large datasets like these can best be understood though the use of Machine Learning (ML) and Artificial Intelligence (AI) algorithms due to their great processing capabilities and high generalization powers. Predictive analysis through AI and ML is therefore being used extensively in the football industry in various areas, such as: match outcome prediction [7] [8] [9], strategic and tactical decision-making [15] [10], injury management [16], and the scouting of players [17]. However, only relatively recently, breakthroughs have been made in learning techniques that are able to cope with the newly acquired highly dimensional spatio-temporal datasets in football [12]. Finally, after lagging behind, predictive algorithms have gained the potency to assist in the tactical decision-making processes of teams using ML. Football team therefore do not have to rely solely on the knowledge of their experts, but can put their trust in more reliable and less biased technological assistance to, ultimately, gain a competitive advantage. Considering the size of the football industry, the potential gains in predictive analytics in football are enormous, since research in this field has been lagging behind when compared to other sports. The integration of football data into prediction models therefore provides a great area of research, by attempting to

[2] `https://statsports.com/apex/` Date accessed: 13-01-2022

model 'the unmodellable sport'.

1-2 Research Questions

The purpose of this work is to explore if and how a learning-based algorithm can be created, which is able to make accurate within-game predictions in football when having knowledge of the past and present states of the game. This section will discuss the research questions and sub-questions of this research. The main research question that is to be answered is formulated as follows:

How can accurate online predictions in football be made with spatio-temporal data using learning-based algorithms?

The definition of online predictions intended here is that the algorithm should focus on making within-game predictions with as little latency as possible. As a result, predictions can be made during the game, which, in turn, can assist in the making of online tactical decisions. Furthermore, spatio-temporal data refers to football data that is collected across both space and time, that is, the state of the football game with all its state variables (e.g., player positions and information about within-game events) across time. Finally, learning-based algorithms refer to algorithms designed using ML techniques.

Two sub-questions were created. Each sub-question will assist in the process of finding a solution to the main research question as formulated above. The sub-questions are as follows:

1. *How can a theoretical, descriptive model describing the course of a football match be created?*

   A theoretical, descriptive model will help to conceptualize and comprehend the complexity of the evolution of a football match and will provide a better understanding of the actual, underlying system. Comprehension of the system is especially useful when dealing with learning-based algorithms, as they don't provide comprehensible information about the system due to their black-box nature. Furthermore, when prediction results are not as desired, the theoretical model may help locate the potential problems.

2. *How can a representative, realistic dataset of football data be obtained, suitable for training and evaluating a learning-based algorithm?*

   Learning-based models are only as good as the datasets that are used to create them. It is therefore of crucial importance to collect a representative dataset that is able to capture the complex dynamics of a football match. Furthermore, the dataset should be large and rich enough for training and evaluation purposes.

1-3 Thesis Structure

This section provides the structure of this thesis report and intends to give an overview of the topics that will be discussed. Chapter 2 describes the heritage of predictions in football, with a particular focus on ML-based algorithms. In addition, the chapter presents the algorithm requirements, discusses the scope of this research, and elaborates on the limitations and assumptions of this work. The chapter concludes with a structured overview of the entire project and a visual representation of the algorithm pipeline of the entire project.

The theoretical background of this work is explained in chapter 3, which dives into concepts such as Markov chains and Markov Decision Processes (MDP's), Artificial Neural Networks (ANN's), and various digital image processing techniques. Chapter 4 then provides the theoretical modelling framework as used throughout this work. A top-down modelling approach was applied, where a very general model is formulated first and abstracted into a more functional MDP model thereafter. The chapter concludes with a proposal of the prediction model, which is based on the MDP model.
Chapter 5 gives a detailed analysis of the creation of the datasets, where it elaborates on the selection process of the data sources and describes how the datasets were obtained. The designing process of the prediction algorithm is then discussed in chapter 6. The chapter evaluates the training, tuning, and testing procedures while also discussing various design choices. The results achieved by the designed algorithms are analyzed in chapter 7, where the main focus lies on the evaluation of the performance of the predictions model. Finally, chapter 8 summarizes the main conclusions of this work and aims to answer the research questions and sub-questions, alongside recommendations for future work.

Chapter 2

Heritage

This chapter describes the scope of this work. Section 2-1 will describe the heritage of predictive modelling in football. A particular focus will be put on research where predictions rely on Machine Learning (ML) algorithms. Section 2-2 then lists and discusses the prediction algorithm requirements. Lastly, section 2-3 discusses the scope of this research. The intention of this chapter is to provide an overview of the purpose of the research. Furthermore, it aims to already provide a preview of the work as a whole.

2-1 Heritage

The use of Artificial Intelligence (AI) techniques and methods have gained large popularity in society over the past decades due to the need to transform large amounts of data into meaningful knowledge and solutions. Nowadays, a growing amount of data is getting documented and the development of automated algorithms for practical solutions are becoming increasingly important. However, the efficiency of such AI methods within society is still developing, as is the case in sports [14]. In the past decade, the interest in data analysis in football has increased immensely as elite sports oriented companies and football clubs have invested large amounts of resources to analyze performance [12]. A shift in reasoning has been taking place, in which analysts do not only want to know who will win, but are keen on investigating why teams will win. In other words, people want to know what the influencing factors are and why certain scenarios take place.

This has led to extensive research in the field of performance evaluation in football in the form of meaningful metrics [18]. Then, based on the analysis of those metrics, teams can apply tactical changes and gain a competitive advantage. Teams apply these tactics mostly offline (i.e., prior to a match) [13], thereby not anticipating unexpected tactical decisions by the opponent or occurrences of unforeseen events. Tactical decisions are thus primarily made reactively, and only little research has been conducted in proactive online decision-making (i.e., within-game decisions) [12].
The first step in online decision-making is making accurate online predictions (i.e. within-game predictions). Teams can then change or update their tactics during a football match based on the

predicted outcomes.
In recent years, prediction algorithms in football have been largely based on the concept of ML, as will be touched upon in subsection 2-1-1. ML-based prediction algorithms have shown great predictive performance and have been able to outperform traditional methods in football [16]. Their power lies in their potential to find relevant pieces of knowledge in huge datasets, which would have been impossible to find otherwise. However, lots of data is needed for the creation of ML algorithms. An analysis of the availability of football data will therefore be given in subsection 2-1-2.

2-1-1 Predictions in football

Predicting the outcomes of future matches in football has been a topic of research since at least the 1960s. Some pioneering researchers within the field of football predictions were Reep and Benjamin (1968) [19] and Hill (1974) [20], who proposed methods to predict result outcomes of future matches. In earlier years, research was based on various statistical methods, which include but are not limited to: Poisson models [21], Bayesian models [22] and rating systems [23]. ML techniques were applied only later and have been gaining popularity over the past decade as a result of the widespread availability of data. One of the first to adopt a ML approach was O'Donoghue et al. (2004) [24], who used ML together with statistical methods for result predictions of the FIFA World Cup 2002.

Machine learning models

In recent years, through the developments in the field of ML technology and the abundance of data, researchers have been able to make predictive models in football on a more accurate level, thereby outperforming the statistical methods [14]. Various authors have proposed different ML models for result prediction in football: Baboota et al. (2019) [7] used a gradient boosting model, Goes et al. (2019) [18] used a logistic regression model, Balogun (2019) [9] used an Artificial Neural Network (ANN) model, but many other ML approaches have been applied to date. The predictive accuracy of the methods applied by most studies was found to be surprisingly high, with accuracy rates of more than 90% in multiple studies, although most studies only made use of a rather limited sample size or focused on only a single dominant team. A systemic review of ML-based performance predictions in team sports was conducted by Claudino et al. (2019) [16]. They found that the ML techniques with the highest predictive potential within football are given by ANN's. Traditional ML algorithms such as Support Vector Machines or Logistic Regression have not been able to keep up with the newly acquired abundance of data and, consequently, have reached their peak performance. ANN's, on the other hand, are able to take full advantage of an increasing amount of data, thereby outperforming the other ML alternatives when datasets get larger. The potential of Neural Networks (NN's) is demonstrated by an algorithm able to detect buildings and create building footprints using only satellite images [25]. Accurate results were obtained by training the network using a dataset of 100k satellite images containing 1.75M labelled building instances, thereby emphasizing that NN's show great performance when confronted with huge datasets. NN's can therefore be regarded as a highly potent predictive technique for football modelling, where datasets are generally enormous. Research in *ANN-based prediction algorithms* was therefore further explored.

Tracking data

Large datasets are generally used for training an ANN-based algorithm. In football, the data types can be classified into two main classes, namely: *tracking data*, defined as the positional coordinate data of players and ball; and *event data*, defined as detailed information about events (e.g. shots on target, goals and passes). A general analysis of the state of ANN-based predictive modeling within football was conducted by Memmert et al. (2017) [26], who based their study on a single, objective dataset of tracking data. They found that dynamic passing and movement patterns before critical events (e.g., goal scoring chances, goals) can be identified by making use of tracking data. In turn, this can help understand the process of preventing or scoring goals, thereby helping a team towards success. Their conclusion is based on various studies which have shown how predictive algorithms can use tracking data to compute collective variables (e.g., centroids, stretch indexes, length and widths, surface areas of teams) to identify the inter-team and inter-line (e.g. line of defenders) coordination of teams before goals [27] [28] [29] [30] [31]. Their findings showcase the potential of ANN-based predictive models in football that exploit tracking data, especially when predictions are made on a team level. A particular focus will therefore be put on research in *ANN-based algorithms that use tracking data for team-level predictions*.

Key performance indicators for online modelling

Current literature in ANN-based algorithms for team-level predictions in sports has mainly focused on the predictions of Key Performance Indicators (KPI's) [18]. "A KPI is a selection, or combination, of action variables that aims to define some aspects of a performance in a given sport and, these KPI's, should relate to successful performance or outcome"[32]. Different authors have succeeded in defining dynamic KPI's capable of quantifying team-level performance within team sports in continuous-time [15] [4] [33] [34] [35] [13]. *In basketball*, Cervone et al. (2014) [15] propose a framework for calculating a dynamic KPI known as the expected possession value (EPV); a real-time estimate of the expected points obtained at the end of a single possession phase. They propose a two-level Markov chain model in which the continuous-time movements of players and ball as well as the discrete-time events (e.g., shots, turnovers, passes) are combined hierarchically for the real-time computation of the Expected Possession Value (EPV). To date, the EPV still remains a popular metric and is still used in the American NBA. *In American Football*, Yurko, Horowitz, and Ventura (2020) [5] propose a novel method for continuous-time within-play valuation in the National Football League using player tracking data. They constructed a Long Short Term Memory (i.e., a class of ANN) to estimate how many meters the ball-carrier is expected to gain from their current position, depending on the locations and trajectories of the ball-carrier and it teammates and the opposing players. They propose a conditional density estimation such that the expectation of any measure of play value can be calculated in continuous-time, similar to the EPV. Similar work within American Football was done by Burke (2019) [33] who successfully attempted to model passing game outcomes during a play using a deep learning approach.
In football, only a handful of authors have tried to create models to compute similar interpretable online metrics, such as the EPV, which can be used in continuous-time. One of the first were Link, Lang, and Seidenschwarz (2016) [34], who were able to quantify attacking shot danger by estimating the probability of scoring a goal at every moment in time throughout a possession phase of a team, using an extensive dataset that included tracking data. A more recent study by Wagenaar et al. (2019) [35] propose a deep convolutional neural network (a class of ANN) to predict goal scoring

opportunities in football based on tracking data and event data of certain events (e.g. possession team, passes). The most promising online KPI in football is proposed in a study by Fernandez and Bornn (2019) [13], who created a deep-learning model to find the EPV, thereby quantifying the expected outcomes at each point in time during a possession phase, similar to the work of Cervone et al. (2014) [15] in basketball and Yurko, Horowitz, and Ventura (2020) [5] in American Football. In their case, the EPV is defined as a real number in the set $[-1,1]$ in which -1 and 1 express the likelihoods of the attacking team and the defending team respectively scoring a goal during their possession phase. An example of the EPV progressing throughout a possession phase can be seen in Figure 2-1.

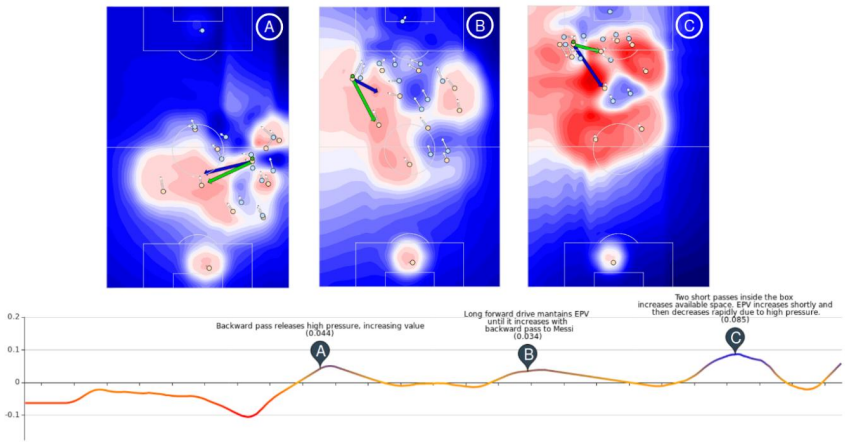


Figure 2-1: The EPV of a possession phase on the y-axis and the time on the x-axis during a Real Madrid - F.C. Barcelona match, where three remarkable turning points have been highlighted. Obtained from Fernandez et al. (2019) [13].

To conclude, research has shown that online, team-level ANN-based predictions in football can be made in the form of dynamic KPI's [35] [13]. Many interesting online KPI's thus remain unexplored, thereby opening doors for novel research. Furthermore, no studies were found examining the link between online predictions and within-game tactical decision-making. This further shows that a great potential still lies in the field of online predictive modelling.

2-1-2 Data availability

One of the major challenges within ML-based algorithms is the need to obtain enormous amounts of data. It is therefore of crucial importance to obtain a dataset as complete as possible. As loosely mentioned in subsection 2-1-1, a complete dataset in football consists of: *tracking data*, defined by the (x,y)-coordinate data of all players and the (x,y,z)-coordinate data of the ball; and *event data*, which consists of detailed information about events (e.g. shots on target, goals and passes), including their time-stamps, and the players involved. Different methods were therefore explored with the aim of obtaining a dataset as complete as possible.

Commercial datasets

In practice, almost all of this data gets extracted from video data and sensor data [36]. In the case of *video data*, camera-based tracking can be performed using computer vision technology, able to accurately calculate the positions of players and ball. Video recordings are used everywhere within football, as is showcased by the British broadcaster Sky TV, who generally use 24 cameras on the ground for their recordings [1]. The current market leader within the world of automatic tracking through video recordings in football is the company ChyonHego, who created the 'TRACAB Optical Tracking' system [2]. Various prominent football leagues (e.g. the English Premier League, the Spanish La Liga, UEFA Champions League) are currently using their piece of technology to extract data.
In the case of *sensor data*, various companies (e.g., Opta Sports, Statsperform, Sportlogiq, Panini-Digital, Scisports) provide detailed and advanced datasets as products, created by making use of state-of-art GPS trackers created by numerous companies (e.g., Statssport, Scisports, Catapult, KINEXON), which are able to accurately transfer the (x,y,z)-coordinates of players during a game of football. There are currently no rules from the FIFA and UEFA prohibiting players to wear trackers during games [3]. Studies with similar research scopes were conducted by influential and prominent research teams, able to get assistance from commercial companies or big teams. The team from the study by Wagenaar et al. (2019) [35] created their dataset of 29 full-length matches by making use of the Amisco multiple camera-system, which was able to track all moving players on the football field. The team from the study by Fernandez and Bornn (2019) [13] worked closely together with the Spanish world-class team F.C. Barcelona, who provided them with a complete dataset. Unfortunately, a collaboration with a team or company could be realized for this project. This work will therefore not go into more detail about these commercial datasets and will only focus on finding data sources available to this project. Additionally, this work reviewed the state-of-art tools and metrics designed by commercial companies, which is presented and discussed in Appendix A-1. Although their studies and products are not accessible for the wide public, it is important to know what tools, techniques and methods have already been created in the industry, even if the theory behind them is not fully accessible.

Data extraction algorithms and available datasets

datasets from commercial companies and top-tier teams are not available for public use. As a result, various authors have attempted circumventing them by producing their own techniques for the extraction or creation of football datasets.
In regard to *event data*, various authors have attempted to create automatic event detection algorithms in football to make datasets cheaper and more available. Studies by Imai et al. (2018) [37], Morra et al. (2020) [38], and Khaustov and Mozgovoy (2020) [39] all propose techniques to automatically classify events from event data. Their results show event recognition can be: effective for some event classes, such as shots on goal, successful passes, unsuccessful passes, and goals; while challenging for other classes, such as tackles and player duels. A more recent study by Morra et al. (2020) [38] proposed a method able to recognize 16 different events using a declarative Interval

[1] http://www.dailymail.co.uk/sport/football/article-1301170/The-impossible-job-Sky-TV-24-cameras-referees-mu
html
[2] https://chyronhego.com/products/sports-tracking/tracab-optical-tracking/
[3] https://www.bbc.com/sport/football/34267968

Temporal Logic based algorithm. Their method was able to identify 9000 events from their dataset. Some event classes (e.g. shots and passes) showed accurate detection results, while others classes (e.g. tackles) showed poor detection performance, as can be observed in Figure 2-2.

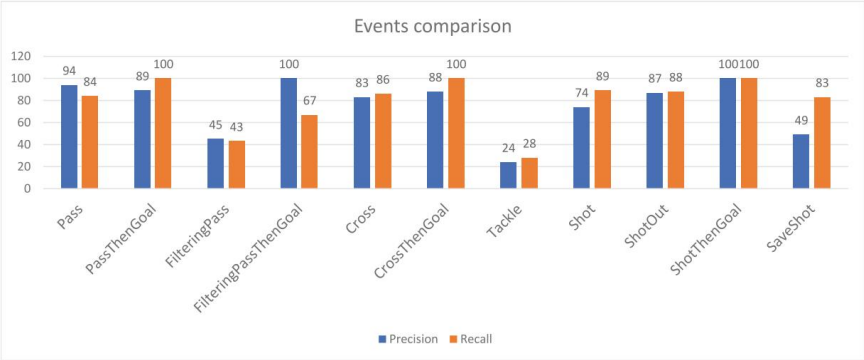


Figure 2-2: Results in precision and recall for event detection from tracking data within football from a study by Morra et al. (2020) [38].

Furthermore, multiple publicly available datasets providing only *event data* were found. One of the most promising datasets covers around 1,941 matches (Pappalardo et al. (2019) [40]). The events in the dataset all consist of multiple tags, such as: player information, timestamps, coordinate positions, event types and event outcomes. Another large dataset includes a total of 9,074 games from the biggest 5 European football leagues and can be downloaded for free on Kaggle [4].
In regard to *tracking data*, different work has been proposed for automatic tracking. A subdivision was made on work proposing: player tracking algorithms, and ball tracking algorithms. Both will be discussed briefly. One of the first to detect and label players were Liu et al. (2009) [41], who proposed an unsupervised labeling technique for automatic player detection using Markov chain Monte Carlo data association. Results were promising but also biased as the sample size consisted of only three short clips. A more recent study by Cui (2018) [42] used a player detection approach based on a so-called fuzzy decision-making Support Vector Machine (SVM) algorithm. They pointed out several shortcomings of current detection techniques, however, their results showed not all players could be detected and tracked. The most promising algorithm follows from a study by Hurault et al. (2020) [43], who present a semi-automatic player tracking and detection algorithm using video images, and were able to obtain impressive results using a self-supervised learning approach, obtaining an average detection accuracy of 97.3%. Their algorithm shows robust performance to different camera angles, although detection accuracy performance decreases whenever player collide or move close to each other. In the field of ball tracking algorithms, a study by Najeeb (2020) [44] was able to estimate the position of the ball using an extended Kalman filter through video images. Results show their approach was able to correctly identify the ball 92.8% of the time. Their code is available for public use and can be found on Github [5].
In addition, a few publicly available few datasets containing only *tracking data* were found. The most promising dataset is presented in a study by Pettersen et al. (2014) [45]. However, the dataset captures only three different games played at the Alfheim Stadium, played by the Norwegian top-

[4] `https://www.kaggle.com/secareanualin/football-events`. Date accessed: 04-07-2021
[5] `https://github.com/samuro95/Self-Supervised-Small-Soccer-Player-Detection-Tracking/`. Date accessed: 04-07-2021

tier team Tromsø IL. Another detailed dataset is presented by Martin Kleppe (2013) [6], who used the work of Muschler et al. (2013) [46] to create a dataset containing the exact locations of all players and ball during a full match played at the Nuremberg Stadium in Germany.

Simulation data

Simulation data can solve the problem of data scarcity, as simulations can be executed millions of times using modern computers. State-of-art football simulators might very well be capable of capturing the complex dynamics of real players, making them a great potential data source for the creation of an ANN-based prediction algorithms. This is demonstrated through games such as FIFA 21 and PES 2021, which are becoming increasingly realistic to a point they can almost not be distinguished from real-life football. In order to mimic the movements of players as close as possible, state-of-art games have been making use of sensor-equipped suits on football players. The FIFA game series, as published by EA Sports, has even been able to equip all 22 players on the field with the state-of-art Xsens suits [7]. The Xsens suits allow the game developers to record every touch, tackle, sprint, and duel of each player on the field at a high intensity. As a result, they have been able to optimize the movements, responsiveness, and physicality of a wide variety of different types of players alongside the behavior of a team as a whole. Realistic simulators, such as the FIFA or PES football game series, give possibilities to analyze simulated games instead of real-life games, as even the complex dynamics of real-life football can be modelled. However, even the most advanced simulators experience the so called simulation-to-reality gap [47], which describes the inherent inability of simulators to capture all the real-life influencing factors of a football match. For example, it is hard to consider factors such as: calf pain, player restedness, supporters influence, or individual coach instructions. This will make the simulated data flawed when compared to a real-life dataset, which may cause problems when predictive models fitted on simulated data will be used for real-life football.

Unfortunately, no realistic football simulators were found which allow fast and easy access to the underlying data. However, two football simulators were found, allowing rather cumbersome methods to extract the data. These two simulators were therefore thought to be interesting for further research. The most promising simulator is the game "eFootball" (formerly known as "PES2021"). The game is not open-source. It is thus not possible to access their software files and transfer the underlying data directly. However, one of their online game-modes allows a top-view camera angle of the match, where players and ball are represented by player-indexed and team-colored circles. In addition, numerous match statistics are represented in a separate section of the interface. This relatively simple visual representation gives possibilities to extract the underlying data by writing a separate digital image processing algorithm. Although this indirect approach may appear cumbersome, it has the potential of creating the most extensive football dataset publicly available.
Another simulator is the Google Research Football (GRF) simulator. The simulator is a project by the Google Research Football team and the football club Manchester City F.C. [8] and explores a new type of realistic gameplay in which they allow people to code their own agents as players through Reinforcement Learning (RL) techniques. The simulator is open-source and thus allows extra lines of code to be written with the goal to transfer the raw underlying data to a separate file. The raw data includes the exact locations of all players and ball alongside extensive contextual information

[6] `https://github.com/ubilabs/soccer-debs-challenge/tree/master/paper`. Date accessed: 04-07-2021
[7] `https://www.xsens.com/blog/fifa-22-motion-capture`
[8] `https://www.kaggle.com/c/google-football/overview/training-rl-agents`

of the match (e.g. event data, players orientation, fitness levels). By letting the simulator run a number of matches, an extensive dataset can quickly be generated. However, when compared to the eFootball simulator, the main drawback of the GRF simulator is that the players move according to rule-based mechanics, resulting in simplistic and less realistic gameplay.

2-1-3 Conclusion

Many authors have tried to model and predict the game of football through ML-based approaches [7] [18] [9] [35] [13]. Within the field of ML, ANN's were found to have the highest predictive potential as they have the ability to identify the relevant pieces of knowledge in enormous datasets, while performance in traditional algorithms lags behind under similar conditions [16]. Zooming in on ANN-based methods, algorithms using tracking data were found to achieve the most promising results for within-game predictions, especially when predictions are made on a team-level [26]. This has led to the development of a series of dynamic KPI's, which can be designed and computed using ANN techniques and tracking data, capable of quantifying plays within team sports in continuous-time. Promising KPI's were created in basketball [15] and in American football [5] [33], which are still being used today at the highest level. In football, only a handful of authors has tried to find ways to compute similar interpretable metrics for online predictions [34] [13] [35]. Many interesting online KPI's thus remain unexplored, opening doors for novel research. Furthermore, no research was found connecting online predictions to online decision-making, i.e., optimization of team tactics during the game. Past research has focused primarily on reactive, offline decision-making, but a great potential still lies in proactive, online decision-making [12]. The first step in creating an online decision-making algorithm is the creation of an online prediction algorithm. This further shows the hidden potential of online predictions in football.

A suitable dataset is required for the training of each ANN-based algorithm. In the case of ANN-based online predictive modelling in football, a suitable dataset includes tracking data and preferably also event data. However, a publicly available dataset meeting those requirements was not found, as the state-of-art technology for automatic tracking and event detection are all in the hands of commercial companies and big football teams. Various authors have attempted circumventing them by creating their own techniques for the extraction of event data [37] [38] [39] and tracking data [43] [44]. Their work encourages people to create their own datasets, which can then be used for any research purpose. Several datasets were found containing either event data or tracking data. The most promising dataset regarding only event data covers around 1,941 matches [40] and is free to download. Two promising datasets regarding only tracking data were found, covering a summed total of four different football games [45] [46]. Both datasets can be downloaded for free. Another possibility for the acquirement of a large enough dataset can be achieved through simulations. Simulations can solve the problem of data scarcity, as they can be executed millions of times using modern computers. However, a drawback of simulators is their inherent inability to capture all the real-life influencing factors of a football match [47]. No existing datasets of simulated data were found, but two promising football simulators were identified which can lead to the creation of a new dataset, namely: the GRF simulator and the eFootball simulator. The GRF simulator is an open-source simulator and thus allows possibilities in the transfer of raw data by writing extra code in the development files. In theory, the GRF simulator is therefore able to generate a fully accurate and complete dataset within a short amount of time. In contrast to the GRF simulator, the eFootball simulator is not open-source. Nonetheless, the simulator has the option to view the

gameplay from an animated, top-view angle of the field, thereby opening doors for indirect access to the underlying data through digital image processing algorithms.

2-2 Algorithm Requirements

Numerous requirements for the prediction algorithm were set based on the heritage, as discussed in chapter 2. The requirements assist in finding a solution to the main research question by narrowing down the problem. The prediction algorithm requirements are as follows:

- **AR01:** An algorithm shall be designed, able to make online predictions in football
- **AR02:** The algorithm shall predict a KPI defining performance on a team-level
- **AR03:** The algorithm shall rely on the usage of an ANN
- **AR04:** The algorithm shall be trained on a large-scale dataset consisting of at least 15 games of tracking data

Algorithm requirement **AR01** is straightforward and follows from the problem and relevancy as described in section 1-1. The requirements **AR02**, **AR03**, and **AR04** are based on the developments within the literature as discussed in subsection 2-1-1, which showed a great potential still lies in ANN-based online predictions of KPI's defined on a team-level through the usage of tracking data.

2-3 Scope of the Research

The scope of the research, as defined in subsection 2-3-1, will further specify the aim of this project and is based on the research questions as provided in section 1-2, the heritage as discussed in chapter 2, and the algorithm requirements as stated in section 2-2. In addition, it will already give a preview of the work as a whole. Afterwards, subsection 2-3-2 will discuss the assumptions and limitations of this research with the intent to give the reader an idea of what can be expected.

2-3-1 Scope of the research

The fast progress in ML in recent years has opened up many possibilities for further research within the field of predictive modelling in football. Current research has shown that even football, previously described by many as an unmodellable sport, can actually be modelled and predicted using state-of-art techniques. However, there still lies great potential in the field of online predictions. This work therefore aims to design an algorithm able to make accurate predictions online in the game of football. These predictions will be made in the form of a goal-related metric specifically tailored to the problem. The problem and ultimate goal of this work can be described as follows:

Problem statement: *Obtain an ANN-based predictor function able to map the state of a football game to a metric, which quantifies goal-related attacking danger.*

The proposed predictor function was inspired by and shows similarities to the predictor function created in the work of Fernandez and Bornn (2019) [13]. They created a deep-learning model to

find the EPV; a KPI quantifying the expected goal probability at each point in time during a possession phase. Nonetheless, the angle of this work will be to design a different metric using another learning-based approach. A comparison between the two will therefore be meaningless and will not be made in this work.

In assistance to finding a solution to the problem statement, a theoretical model will be created first. The theoretical model will help solve the problem in a more structured way by describing the system and the behavior of the system in a mathematical language. This work aims to accurately describe the course of a football match using Markovian theory and, to be more precise, formulate the system into a Markov Decision Process (MDP) framework. The modelling problem can be defined as follows:

Sub-problem statement: *Obtain a MDP model of the system, where the system describes the state of a football match over time.*

The Markovian modelling framework for online predictive purposes is inspired by the work of Cervone et al. (2014) [15], who proposed a MDP framework for calculating a real-time estimate of the expected points obtained at the end of a single possession phase in basketball.

2-3-2 Limitations and assumptions

The following points state and discuss the limitations and assumptions of this work.

1. **Only simulation data is considered:** The goal is to obtain a dataset as complete as possible, which includes tracking data and preferably also event data. No suitable real-life football data sources or datasets were found meeting these initial requirements. A choice was therefore made to use simulation data using two different simulators, namely the GRF simulator and the eFootball simulator. Both provide possibilities to indirectly extract the large amounts of data. The usage of simulators for the acquirement of data has several advantages. First, simulators provide the possibility to generate more data if needed. Furthermore, they allow controlled experiments to be executed. Finally, the heritage showed that state-of-art simulators, such as the eFootball simulator, are able to create highly realistic football environments similar to real-life scenario's. Simulation data was therefore chosen for further research. A more thorough analysis of this choice will be given in section 5-1.

2. **Prediction results will be based primarily on data from the eFootball simulator:** A main goal of this project is to create a dataset of simulated football data. Both the GRF simulator and the eFootball simulator were found to be of interest to the project and were further investigated for usage. Data can be acquired a lot faster using the open-source GRF simulator, which was therefore chosen as the initial source of data. However, the gameplay of the eFootball simulator was found to be more advanced and realistic, and thus more relevant to the problem. This naturally led to the following plan of action. First, a ANN-based prediction pipeline will be created based on the data acquired by the GRF simulator. Then, after satisfactory prediction results are obtained, a new dataset will create using the eFootball simulator. A digital image processing algorithm will be written to extract the data from the simulator. The same ANN-based prediction pipeline will be used on the eFootball dataset. Since the data from the eFootball simulator is more realistic, this work will primarily focus

on making predictions within the eFootball environment.

3. **The dataset is not perfect:** The data from the eFootball simulator will be collected using a digital image processing algorithm. This digital image processing algorithm will be coded during this project and will contain flawed accuracy in processes such as the labelling and detection of players, ball and events. The dataset will thus not consist of completely accurate data.

4. **Within-game predictions can be made with limited knowledge:** The prediction algorithm will be based upon a simplified model of a football match, where the state of the game is described by only a selected amount of relevant state variables. The prediction algorithm will then base its predictions on a selected series of past states of limited variables, where an assumption was made that is possible.

5. **Predictions based on simulated football data are, to some extent, representative for real football predictions:** Simulators cannot capture all real-life influencing factors of a football match. However, state-of-art football simulators have become increasingly realistic to a point they can almost not be distinguished from real-life football. As discussed in the heritage, the game-makers have even used sensor-equipped suits on real football players to optimize player movements and team behavior. It is therefore assumed that predictions based on the eFootball simulator will be, to some extent, representative for real-life football predictions.

6. **The purpose of this work will be dedicated to the development of an online prediction algorithm:** The scope of this research is focused solely on the development of a within-game prediction algorithm. A link to within-game tactical decision-making will therefore not be made, although several decision-making solution proposals based on the prediction model of this work will be presented and discussed in Appendix C.

7. **The abstracted system description of the MDP model is an accurate representation of the actual system:** Various heuristics will be used to reduce the complexity of the actual system and frame it into a MDP model such that the problem becomes solvable. Examples of such heuristics are the reduction of the state space through state variable selection, and state discretization. It is assumed that the MDP model will still be an accurate representation of the actual system after simplifications have been made.

8. **The designed/chosen metric quantifies goal-related attacking danger:** No separate analysis will be made to discover the correlation between the chosen metric and goal-related attacking danger, although an assumption was made that this correlation does exist. This assumption was made using logical reasoning and basic football knowledge.

2-3-3 Overview of entire project

The entire project consists of a series of steps. First, a model of the system will be proposed using a MDP modelling framework in chapter 4. Chapter 5 will then discuss the process of finding the right data sources and elaborate on the creation of the datasets used in this work. Subsequently, based on the MDP model, a predictor function will be designed using an ANN in chapter 6. Finally, the prediction results will be discussed in chapter 7. An overview of the entire project is visualized in Figure 2-3.

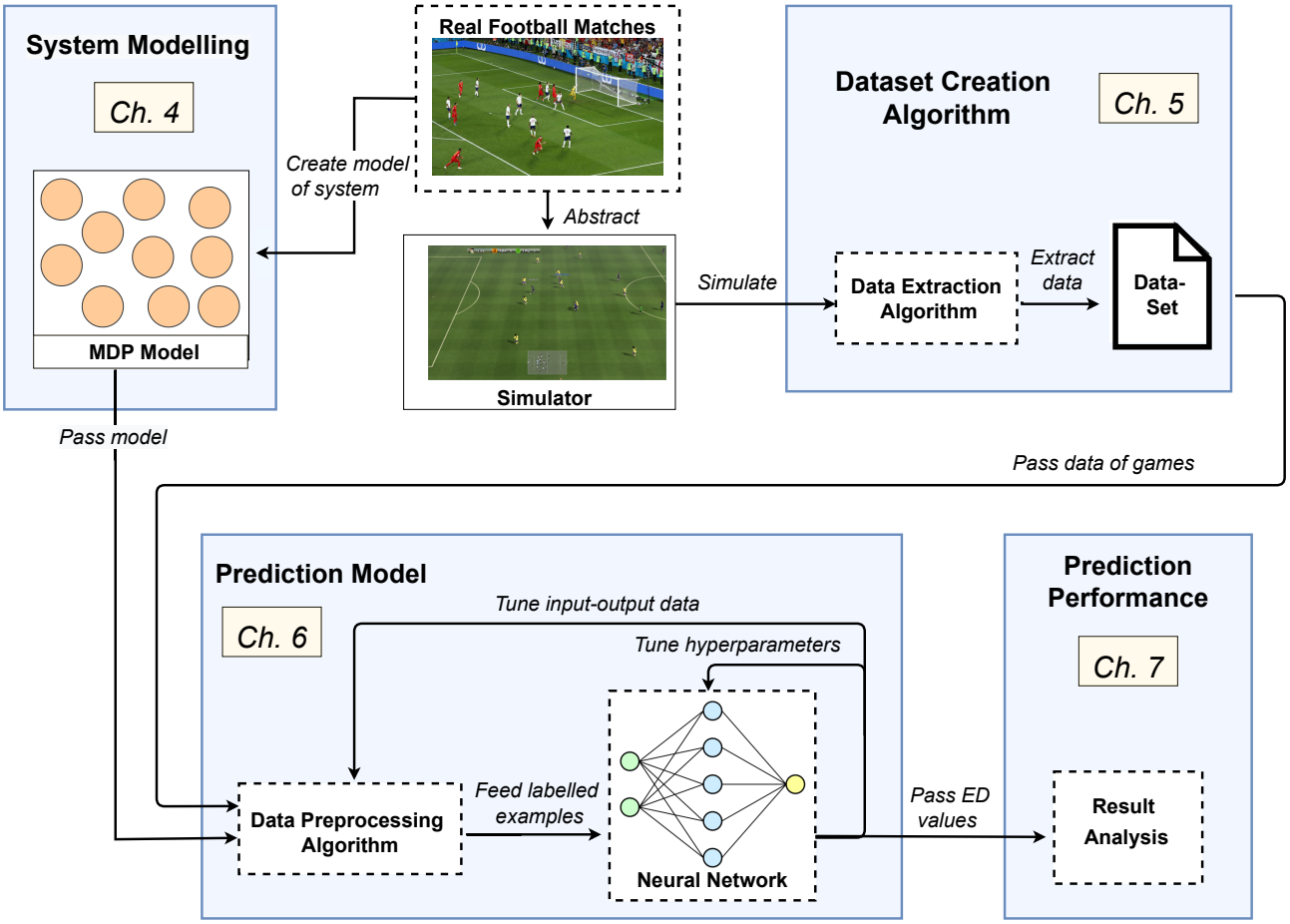


Figure 2-3: Workflow diagram of project, classified in separate segments. Each segment is labelled with the corresponding chapter.

Chapter 3

Preliminaries

This chapter explains the theoretical concepts used throughout this work. First, Markovian models will be explained in section 3-1. The Markovian models used in this work are a Markov chain, a higher-order Markov chain, and a Markov Decision Process (MDP). Second, the concept of Machine Learning (ML) will be explained in section 3-2, with a particular focus on Artificial Neural Networks (ANN's). Finally, the used digital image processing techniques will be explained in detail in section 3-3.

3-1 Markovian Models

Markovian models are stochastic models used to model random or partially random changing systems. Markovian models assume all future states do not depend on past states, but only on the current state; a concept also known as the Markov property [48]. Mathematically speaking, given a state $s = \{s_t : t \in \mathbb{N}\}$, the Markov property holds that,

$$\Pr(s_{t+1} \mid s_t, s_{t-1}, \ldots, s_0) = \Pr(s_{t+1} \mid s_t). \tag{3-1}$$

Following the Markov property assumption, it becomes possible to make models of systems which would otherwise be intractable. In the case of predictive modelling in real-life scenarios like football, it is therefore desirable to design a model which possesses the Markov property. Markovian models can be used as a mathematical framework to describe a football match in a simplified manner, aiming to only include the most relevant game features and variables.

In this work, multiple Markovian models were used to describe the system. The theoretical background behind each of the relevant models will be explained in the following. First, a Markov chain will be described briefly in subsection 3-1-1. Subsection 3-1-2 will then extend the Markov chain into a higher-order Markov chain model. Finally, subsection 3-1-3 will describe the theory of MDP's.

3-1-1 Markov chain

A Markov chain is a stochastic sequence of states, where the probability distribution over the next states is only dependent on the current state. This means that state transitions are random, where each state transition is determined based on a conditional probability distribution. Markov chains are considered to be memoryless, although the dependence of state transitions on a conditional probability distribution could also be regarded a form of memory of the system [49]. A Markov chain can be defined as a discrete-time or a continuous-time process. Only discrete-time Markov chains were considered for this project.

Discrete-time Markov chains can be defined by a 2-tuple (S,P), where the Markov chain itself is a sequence of states: $x_0, x_1, x_2, \ldots, x_t$, where each state $x_t : t \in \mathbb{N}$ is located in state set S. The possible values $s_i \in S$ with $i = \{0, 1, 2, \ldots, n\}$ form a countable set S of size n. Generally, S is a finite size with n being a finite integer. However, Markov chains with countably infinite state spaces have also been explored throughout the literature [50], and show different behaviour than Markov chains with finite state spaces [51]. Throughout this work, only finite state spaces were explored, and countably infinite state spaces will therefore not be discussed any further.

If the probability $P(x_{t+1} = s_j | x_t = s_i)$ does not depend on time t, a Markov chain is considered to be homogeneous, while the opposite case is called inhomogeneous. Provided a discrete-time Markov chain is homogeneous with S being a countable set at most, the transition probabilities are given by $P = (p_{ij})_{S \times S}$, where

$$p_{ij} = \Pr(x_{t+1} = s_j | x_t = s_i) \tag{3-2}$$

is the probability of transitioning from $s_i \in S$ to $s_j \in S$. The process of possible state transitions, where each possible transition has its own probability, is graphically visualized in Figure 3-1.

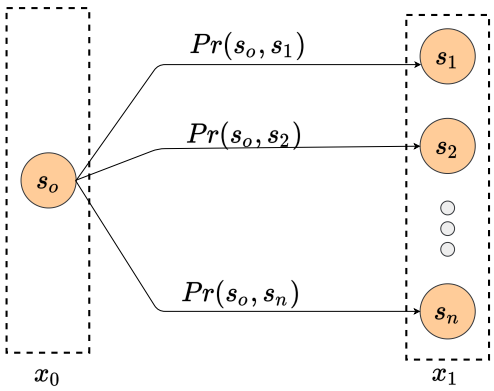


Figure 3-1: A visualization of a discrete-time, homogeneous Markov chain probability distribution from initial state $s_o \in S$ towards any other state $s_i \in S$ between two consecutive time instances.

An important step within the creation of a Markov chain is the estimation of the one-step transition probabilities between states. Consider a discrete-time, homogeneous Markov chain with a finite state space S, such that $|S| < \infty$. The Markov chain is observed at the following points in time, $(0, 1, 2, \ldots, T)$, on the trajectory $(s_0, s_1, s_2, \ldots, s_T)$. The transition probabilities p_{ij} can be now estimated by means of the maximum likelihood; a technique which can be used for estimation a probability distribution, based on a given dataset. Through maximization of a likelihood function, it becomes clear which probabilities are most probable for a given dataset. The individual

transition probabilities between states p_{ij} can be estimated through the likelihood function,

$$\begin{aligned} &\Pr(x_0 = s_0) \prod_{t=0}^{T-1} \Pr\left(x_{t+1} = s_j \mid x_t = s_i\right) \\ &= \Pr(x_0 = s_0) \prod_{i=1}^{n} \prod_{j=1}^{n} p_{ij}^{k(ij)}, \end{aligned} \tag{3-3}$$

where $k(i,j)$ is defined as the number of transitions from s_i to s_j in the observed trajectory. After discarding the initial term (the initial state is arbitrary), the maximum likelihood estimator of p_{ij} can be computed as follows,

$$\hat{p}_{ij} = \frac{k(ij)}{k(i,\cdot)} \tag{3-4}$$

with $k(i,\cdot)$ defined as the total number of transitions coming out of state s_i in the given Markov chain. Asymptotic convergence applies, where a longer Markov chain will yield a more reliable probability distribution, and optimal estimation is realized whenever $k(i,\cdot) \to \infty$. Furthermore, before estimation and given all state transition probabilities are unknown, the complete probability distribution will consist of unknown $(n-1)n$ parameters, where n is the size of the state set. The last parameter can be deduced from the fact that the sum of all transition probabilities towards a single state should sum up to one.

3-1-2 Higher-order Markov chain

A major limitation of a Markov chain when modelling a highly dynamic environment is its inability to describe complex correlation properties between long-distance states [52]. This limitation can be solved using the concept of higher-order Markov chains, additive Markov chains or a discrete-time Markov chain of order m [53], which all refer to the same theoretical model. Consider the state sequence $(x_n, x_{n-1}, x_{n-2}, \ldots, x_0)$, where each state x_i is confined within finite state space S. Then, a Markov chain of order m, with m being a finite number, is a process which satisfies:

$$\begin{aligned} &\Pr(x_n \mid x_{n-1}, x_{n-2}, \ldots, x_1) \\ &= \Pr(x_n \mid x_{n-1}, x_{n-2}, \ldots, x_{n-m}) \text{ for } n > m. \end{aligned} \tag{3-5}$$

Instead of future states being dependent on only the current state, Equation 3-5 describes a process where future states are dependent on the past m states, thereby creating a model which explicitly includes memory. A Markov chain of order m can be redefined such that it possesses the Markov property by constructing a new chain $\tilde{x}_n$ from x_n. A chain $(\tilde{x}_t)$ from states $(x_t, x_{t-1}, \ldots, x_{t-m+1})$ can be constructed for $t > m+1$ as follows:

$$\tilde{x}_t = \begin{bmatrix} x_t^T & x_{t-1}^T & \ldots & x_{t-m+1}^T \end{bmatrix}^T, \tag{3-6}$$

where each state $\tilde{x}_i$ is a concatenation of m different states following a certain trajectory. Furthermore, the chain now satisfies the Markov property, such that:

$$\Pr(\tilde{x}_{t+1} | \tilde{x}_t, \tilde{x}_{t-1}, \ldots, \tilde{x}_0) = \Pr(\tilde{x}_{t+1} | \tilde{x}_t) \tag{3-7}$$

3-1-3 Markov decision process

An MDP is a stochastic control process which can be used as mathematical framework for decision-making models where the outcomes are partly stochastic and cannot be fully controlled by the

decision-maker [54]. An MDP is an extension of a Markov chain and can be defined by a 4-tuple (S, A, P, R), where: S is the state space; A is the action space; P is the set of probabilities, with $P_a(s_t, s_{t+1})$ the probability that action a_t in state s_t will lead to state s_{t+1}; and R is the set of rewards, with $R_a(s_t, s_{t+1})$ the immediate reward obtained after a transitioning from state s_t to state s_{t+1} after action a_t. Furthermore, MDP's follow the Markov property, a principle stating that the next state s_{t+1} is dependent on the current state s_t and the decision-maker's action a_t, but conditionally independent of all previous states and actions.

The goal in MDP's is to find a good policy π for the decision-maker. Policy $\pi(s)$ is a function which defines which action to be selected when in state s. A good policy π will maximize some cumulative function of rewards, which usually is the expected sum over a potentially infinite horizon. An assumption is made that the state s is observable when the action needs to be taken. Whenever this assumption holds untrue, the problem can be called a Partially Observable Markov Decision Process (POMDP)[55]. This work, however, only considers fully observable states for its MDP model.

3-2 Machine Learning Models

This section discusses the theory behind the ML concepts used throughout this work. First, a general introduction to ML will be given in subsection 3-2-1. Subsection 3-2-2 will then elaborate on the general theory behind ANN. A description of how ANN's are designed is given in subsection 3-2-3.

3-2-1 Machine learning

ML is the study in which computer algorithms automatically get improved through experience [56]. ML algorithms make use of sample data, known as training data, to make a model with the purpose of making predictions or decisions without explicitly being programmed to do so [57]. This way, the machine will learn abilities to perform accurately on new, unseen tasks which were derived from the training data. ML is an umbrella term which encapsulates many different computational algorithms. Some of the most known algorithms are: linear regression, logistic regression, decision trees, support vector machines, and Neural Networks (NN's). As already discussed in chapter 2, this work will focus on NN's.

Classes of Neural Networks

NN's consist of three major classes: ANN's, convolutional neural networks, and Recurrent Neural Networks (RNN's). An ANN is the simplest class of NN's, but can already find any input-output relation when given enough data. Convolutional neural networks use the same basic principles as ANN's, but are specifically designed to analyze images. Finally, recurrent neural networks also use the same principles as ANN's, but specialize in analyzing sequential data, such as speech recognition, handwriting recognition, or other sequential data types. This work only considers ANN's as they already serve as a very powerful prediction tool and are generally easier to design than more advanced concepts such as models within RNN's. Nonetheless, RNN's models have the potential to be more suitable for predictive modelling as the states of a football game across time can be seen as a temporal sequence. This may therefore be an interesting topic of research for future work.

3-2-2 Artificial neural networks

ANN's are computational networks which are able to solve large, complex problems by computing a universal function approximator [58]. ANN's are made up of large amounts of artificial neurons (or nodes) interconnected with each other like a web. The goal is to learn from presented information and find a right combination of nodes, connections and weights in a network such that any input-output relationship can be mimicked. A distinction can be made between *classification* and *regression* problems. Classification is the task of predicting a *discrete y* label as an output based on a x example as an input, whereas regression is the task of predicting a *continuous* quantity y as an output based on an x example as an input.

The process of finding the right network can be subdivided into a *training phase*, a *validation phase*, and a *testing phase*. The initial dataset is usually split up into three parts such that each phase has its own sample of data and can be carried out in an unbiased manner. During the training phase, labelled input-output examples ($x \rightarrow y$) are given repeatedly to the network. Output results then get improved by adjustments of the network's parameters using a learning method called backpropagation, i.e., the backward propagation of the error. During the validation phase, the parameters of the fitted model get tested on a separate dataset known as the validation set, with the purpose of tuning a set of parameters called the hyperparameters. During the testing phase, a final evaluation of the fitted model is performed on another dataset known as the test set, which serves as an unbiased testing procedure to quantify the performance of the parameters and hyperparameters of the fitted model.

Neurons

A *neuron* is a node within the ANN structure which computes a weighted average of its input values and passes this sum on through a non-linear function known as the *activation function*. The output of this activation is then passed on to other neurons in the network. In a mathematical sense, a single neuron has the following parameters: an input vector x of input size n_x, weights w, and scalar bias b, defined as follows:

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n_x} \end{bmatrix}, \qquad w = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_{n_x} \end{bmatrix}, \qquad b = \begin{bmatrix} b \\ b \\ \vdots \\ b \end{bmatrix}. \tag{3-8}$$

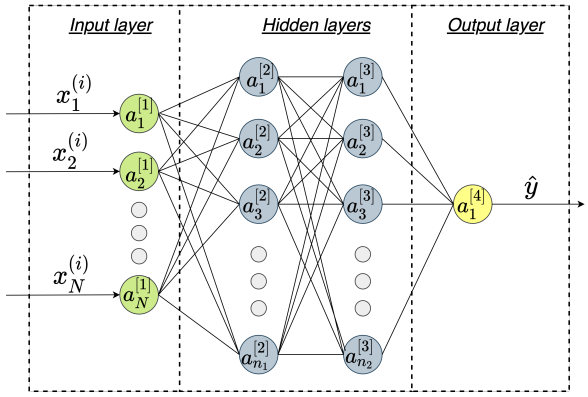
The parameters w and b are needed to compute the intermediate output vector z, which is a weighted sum of n_x different input values plus a bias term b. The *activation function* $g(z)$ then determines the output of each neuron. Usually, nonlinear functions are used, as this allows complex non-linear problems to be solved through relatively few neurons. These equations can be formulated as follows,

$$\begin{aligned} z &= w^T x + b \\ a &= g(z) \end{aligned} \tag{3-9}$$

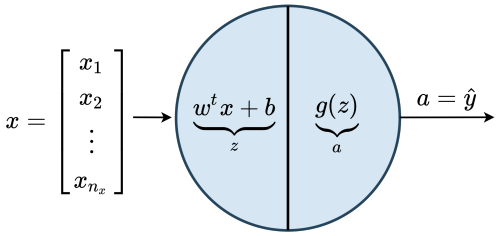
A graphical representation of a neuron is given in Figure 3-2b.

Forward propagation

The neurons are interconnected through connections with certain weights. Each weight gives a relative importance to the connection. Typically, ANN's are structured trough different layers of artificial neurons, in which each layer will perform different transformations on their inputs. Different connections are possible between two layers. Layers can be fully connected, in which each node in one layer is connected to each node in the second layer. These connections between nodes result in numerous possibilities in patterns for the whole network, in which the network can be visualized through a directed, weighted graph, as portrayed in Figure 3-2a. Signals will travel all the



(a) Graphical representation of a three-layer Multilayer Perceptron (MLP), which is a class of an ANN. During the training phase, the weights w and biases b are adjusted to minimize the cost function.



(b) Graphical representation of a single neuron, where two separate computations take place. First, z is computed, which, in turn, is used for computation of a.

Figure 3-2: Architecture of an ANN, where Figure 3-2a depicts a general overview of a network, and Figure 3-2b zooms in on what happens inside a single neuron.

way from the first layer to the last layer, in a process called *forward propagation*. The first layer of the network is called the input layer and will obtain the external data. The last layer, that ultimately yields an output, is called the output layer. In between the input and output layer are the hidden layers of the network. The output layer will eventually complete a certain task. For example, an algorithm could have the task to predict the likelihood of the occurrence of a goal in the next 10 minutes.

Labeling each neuron in the network is done through: subscript i, representing the i^{th} position of the neuron within a layer; and superscript $[l]$, representing the l^{th} layer in the network. Furthermore, each training example is labeled through superscript (k), representing the k^{th} input-output training example. The equations of Equation 3-9 can now be extended to compute a single forward propagation step. For a single training example, the output $z^{[l](i)}$ can be computed as follows,

$$\begin{bmatrix} z_1^{[l](k)} \\ z_2^{[l](k)} \\ \vdots \\ z_{n^{[l]}}^{[l](k)} \end{bmatrix} = \underbrace{\begin{bmatrix} w_1^{[l]T} \\ w_2^{[l]T} \\ \vdots \\ w_{n^{[l]}}^{[l]T} \end{bmatrix}}_{W^{[l]}} \begin{bmatrix} a_1^{[l-1](k)} \\ a_2^{[l-1](k)} \\ \vdots \\ a_{n^{[l-1]}}^{[l-1](k)} \end{bmatrix} + \underbrace{\begin{bmatrix} b_1^{[l]} \\ b_2^{[l]} \\ \vdots \\ b_{n^{[l]}}^{[l]} \end{bmatrix}}_{b^{[l]}} \tag{3-10}$$

Observe that only a single training example k is considered in Equation 3-10. When considering all training examples m, the forward propagation equations can be extended by including the whole training dataset. A general matrix notation is applied and yields the final equations defining the entire forward propagation process of all parameters over all training examples.

$$\begin{aligned} Z^{[l]} &= W^{[l]T} A^{[l-1]} + b^{[l]} \\ A^{[l]} &= g^{[l]}\left(Z^{[l]}\right) \end{aligned} \tag{3-11}$$

where matrices $W^{[l]}$ and $b^{[l]}$ remain the same as defined in Equation 3-10. Matrices $Z^{[l]}$ and $A^{[l]}$ are defined as follows:

$$Z^{[l]} = \begin{bmatrix} z_1^{[l](1)} & z_1^{[l](2)} & \cdots & z_1^{[l](m)} \\ z_2^{[l](1)} & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ z_{n^{[l]}}^{[l](1)} & \cdots & \cdots & z_{n^{[l]}}^{[l](m)} \end{bmatrix}, \quad A^{[l]} = \begin{bmatrix} a_1^{[l](1)} & a_1^{[l](2)} & \cdots & a_1^{[l](m)} \\ a_2^{[l](1)} & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ a_{n^{[l]}}^{[l](1)} & \cdots & \cdots & a_{n^{[l]}}^{[l](m)} \end{bmatrix} \tag{3-12}$$

with $Z^{[l]} \in \mathbb{R}^{n^{[l]} \times m}$, $W^{[l]} \in \mathbb{R}^{n^{[l]} \times n^{[l-1]}}$, $A \in \mathbb{R}^{n^{[l-1]} \times m}$, and $b \in \mathbb{R}^{n^{[l]} \times m}$. $A^{[0]} = X$ is used as initialization, where X is a matrix of horizontally stacked input vectors from the examples coming into the input layer. X is defined as follows,

$$X^{[l]} = \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & \cdots & x_1^{(m)} \\ x_2^{(1)} & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ x_{n_x}^{(1)} & \cdots & \cdots & x_{n_x}^{(m)} \end{bmatrix}, \quad X \in \mathbb{R}^{n_x \times m} \tag{3-13}$$

It is important to prevent all neurons in the first layer (as well as subsequent layers) to compute the same function, a concept also known as the symmetry problem. Fortunately, this problem can easily be solved by randomly initializing W and b.

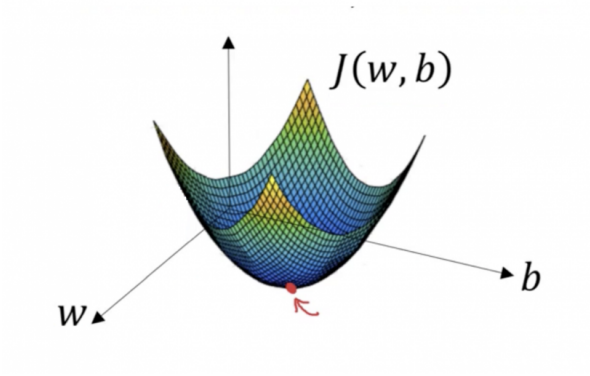
To summarize, forward propagation is the process of feeding all training examples layer-by-layer through the network. The input values of the training examples are given to the input layer of the network without any operations. Each subsequent layer then receives the values of the previous layer and applies multiplication, addition and activation, until the output layer is reached.

Backpropagation

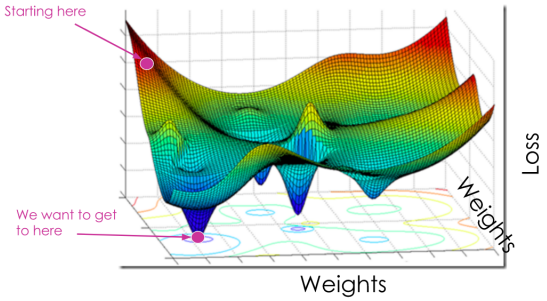
Backpropagation is used to update the parameters W and b during the training phase. This process occurs after forward propagation and analyses of the error between the predicted output value $\hat{y}$ after forward propagation and the actual output values y. The prediction error between the predicted output $\hat{y}^{(i)}$ and the actual output $y^{(i)}$ of a single training example i is called *the loss* and can be calculated by a *loss function* $\mathscr{L}(\hat{y}, y)$. Then, by taking the adjusted average of the loss functions over all training examples m, the *cost function* $J(W, b)$ can be computed. The goal is to adjust W and b in a way that it minimizes the overall cost function $J(W, b)$. Minimization of the cost function,

$$\min J(W, b) = \min \frac{1}{m} \sum_{i=1}^{m} \mathscr{L}\left(\hat{y}^{(i)}, y_{(i)}\right), \tag{3-14}$$

through the process of backpropagation will yield a local (or global) minimum and thereby also yield corresponding values for the weights W and biases b. A graphical representation of a convex cost function is given in Figure 3-3a. The illustration shows a rather simple representation of the cost function, with a global minimum value (highlighted with a red arrow) given by optimal weights w and biases b. In practice, the cost function will almost certainly have local minima and thus not be convex, as is the case in Figure 3-3b [1]. The figure shows a more realistic representation of the cost function, where the highly non-linear behavior is visualized. It is important to initialize the weight carefully to avoid getting stuck in local minima or plateaus when minimizing the loss each iteration.



(a) Simple representation of cost function $J(W,b)$. [59]



(b) A more realistic representation of cost function $J(W,b)$.

Figure 3-3: Examples of two cost functions $J(w,b)$ of an ANN are given. For plotting purposes, W has been simplified to a single dimension in both plots.

Finding a minimum is done through an iterative process called *gradient descent*. Each iteration of gradient descent, the parameters W and b are updated in the direction of the steepest descend, defined by the negative gradient. The step size for the updates of W and b each iteration is determined by the *learning rate* α. An iteration process defined by the following equations:

$$\begin{aligned} W^{[l]} &:= W^{[l]} - \alpha \frac{\partial J(W,b)}{\partial W^{[l]}} \\ b^{[l]} &:= b^{[l]} - \alpha \frac{\partial J(W,b)}{\partial b^{[l]}} \end{aligned} \tag{3-15}$$

Learning rate α is a tuning parameter, known as a *hyperparameter*, and is generally regarded as the most important hyperparameter [60]. Great care should be put into selecting an appropriate value for $\alpha \in [0,1]$. A learning rate too low will have very tine updates to W and b, resulting in terribly slow progress. A learning rate too high may lead to divergent behavior or undesirable overshoot near minimum points. The effects of the learning rate are graphically illustrated in Figure 3-4, where four different learning rates were tested on a convex cost function $J(W,b)$. Furthermore, when training deep networks, it can be helpful to include a learning rate decay. As you get closer to the minimum, smaller steps are desired, because a large step might result in overshooting the intended minimum point. Exponential decay seems to be most commonly applied whenever using a variable learning rate [61]. More advanced optimization algorithms are gradient descent with

[1] `https://medium.com/@RosieCampbell/demystifying-deep-neural-nets-efb726eae941,Dateaccessed: 12-02-2022`

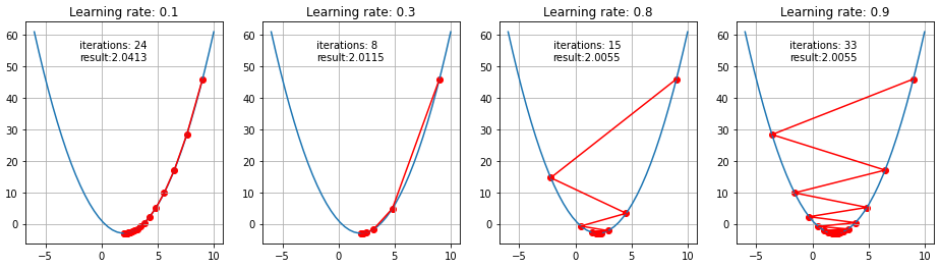


Figure 3-4: Multiple iterations of gradient descent are applied on the convex cost function $J(W,b)$, updating the parameters of W and b with each iteration. Four different values for the learning rate α were tested, namely: [0.1, 0.3, 0.8, 0.9] [62].

momentum [2], Root Mean Square (RMS) prop [3], and Adam optimization [63]. All these optimization algorithms are improvements or extensions on the basic principle of gradient descent and will be discussed in more detail in chapter 6.

The process of backpropagation starts from the output layer and moves backwards towards the input layer. When all layers have been passed in a backwards motion, forward propagation is done again. This process is repeated until stopping conditions are reached. Ideally, this happens when the value of the cost function is at a global minimum and satisfactory results are obtained.

Multi-layer perceptrons

This work uses MLP's to create the predictor functions [64]. A MLP is an extension of the original Perceptron model, proposed by Rosenblatt [65], and a popular class within ANN's. A MLP consists of one or more hidden layers in between the input and output layer, with all its neurons arranged as layers. The neurons in each previous layer are fully connected to the next layer, and each connection is directed towards the subsequent layer (see Figure 3-2a). MLP's have numerous advantages which all contributed to the choice of selecting the MLP as a framework for prediction. First, within the context of ANN's, the architecture of MLP's is relatively basic (with only feedforward signals), making them easier to tune and optimize as fewer hyperparameters exist. Second and third, they can be used for complex, non-linear problems and work great for large input data sequences. Fourth, fast predictions can be made when training is completed, thereby making them suitable for online prediction algorithms.

Transfer learning

Training an ANN often requires a large labeled dataset and substantial computational resources. It can therefore take quite some time to build, shape, and design an ANN. However, random initialization of the weights and biases assumes there is no a priori information available before the network is trained. In the case there exists prior knowledge about the problem to be solved, transfer learning could come in handy. Whenever an ANN has already been designed, tuned and trained on a similar dataset, the weights and biases can be transferred as initialization of the new model.

[2] `https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/SGD`. Date accessed: 19-04-2022

[3] `https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/RMSprop`. Date accessed: 19-04-2022

This way, knowledge of a previously solved problem can be used as memory for a new problem and, as a result, can save substantial time and efforts.

3-2-3 Design of an artificial neural network

Design of an ANN consists of a series of systematic steps, namely: collecting the data; preprocessing the data; building, shaping, and training the network; and testing the performance of the network. All steps will be discussed briefly in the following.

Data collection

An important step in the design of an ANN is the collection of a sufficient amount of data. Initially, raw data is gathered into a large dataset. Subsequently, the data is processed into a sample set of labeled input-output examples, which can then be used for training the network. Unfortunately, there is no general solution to know beforehand how much data is required for the design of a *good enough* model, although some general guidelines do exist. For example, a common rule of thumb is that the sample size needs to be at least 10 to 100 times the number of features (which is the size of the input layer) [66]. Another more widely applied rule-of-thumb is that the sample size should be at least 10 times larger than the total number of weight parameters in the network [66]. Ultimately, the model should generalize the data such that it can make predictions on new data. To achieve this task, enough data needs to be gathered to cover for the diversity and complexity of the underlying system. Whenever too little data is obtained, it can always be beneficial to add more data to the dataset, as long as the complementary data can add to the diversity of the existing data, such as the inclusion of more corner cases.

Data preprocessing

Data preprocessing is the process of preprocessing data before it can be used efficiently for a specific ML model. There are different reasons why data cannot be used directly but needs to be preprocessed. A couple of examples are: extra requirements need to be added to the data; noise and errors need to be adjusted and corrected; and complex nonlinear dependencies need to be simplified or removed. Although transforming raw data into a suitable dataset can be specific to each project, various standard practices can be used during the data preparation phase.
These practices are: data cleaning, feature selection, feature engineering, and dimensionality reduction. Data cleaning refers to the identification of errors in the data and correcting them. Feature selection is the identification of input variables most relevant for solving the problem. Data transforms is the process of changing the scale or the distribution of the variables. Feature engineering consists of the derivation of new variables from the data based on relevant features. Dimensionality reduction is the process of reducing the dimension of the data through creations of compact projections. All of these methods are a complete discipline on their own and have already been studied thoroughly. Moreover, each of them have their specialized algorithms that can be applied to the dataset.

Building, shaping and training the network

The process of tuning the weights and biases on the training data is known as the *training phase*. The training phase is followed by the *validation phase*, where the hyperparameters get tuned on an unbiased validation data. The final model, with all its parameters and hyperparameters, is then tested for performance on another unbiased dataset, known as test dataset, during the *testing phase*.

Parameters are the weights $W = \{W^{[0]}, W^{[1]}, \ldots, W^{n_l}\}$ and biases $b = \{b^{[0]}, b^{[1]}, \ldots, b^{n_l}\}$ of all layers. These are modified during forward propagation and backpropagation with the goal to minimize the cost function. However, there are other plenty of other parameters which can be adjusted when designing a neural network, such as: learning rate α, the number of iterations, the number of hidden layers n_l, the number of hidden neurons $n^{[l]}$ in each layer l, the choice of activation function, choice of loss function, etc. These parameters are called *hyperparameters*. Finding the right parameters and hyperparameters in the design of a network is an empirical process. Results can be improved by continuously experimenting with different parameters and hyperparameters and adjusting them each iteration based on results.

Performance metrics for network evaluation

ML based models can be evaluated using a wide variety of performance metrics, which, in the case of ANN's, are integrated into the model as the cost function, but are also frequently used during the validation and testing phase to check model performance. At least 48 different performance metrics exist, all focusing on different aspects of the error distributions of the predictions [67]. Two major classes of performance metrics within ANN exist and can be subdivided into regression and classification metrics. However, this work focuses solely on the prediction of continuous values and therefore only considers *regression-based performance metrics*.

In regression problems, performance metrics all quantify how close the predicted value is relative to the actual value. A variety of performance metrics exist, each with its own strengths and weaknesses. The three most widely used performance metrics are: (1) R-square, (2) Mean Squared Error (MSE), and the (3) Mean Absolute Error (MAE) [67]. All are mathematically displayed in the following:

$$R^2 = 1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y}_i)^2} \quad (3\text{-}16) \qquad \text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \quad (3\text{-}17) \qquad \text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| \quad (3\text{-}18)$$

The R-square metric (R^2) of Equation 3-16 is a value in the set $[0, 1]$, where a value close to 1 indicates a low error between the predicted values and the actual values. R-square works well on models based on systems with lots of dependent variables. However, when systems compose of many independent variables, the model could test very well on the training data, but have poor performance on the test data as a result of overfitting. This problem can be partially solved using the extension of the R-square metric, which is the adjusted R-square. The adjusted R-square gives less weight to added independent variables, thereby preventing potential overfitting problems. The MSE of Equation 3-17 yields the average squared error of the predictions and can be used efficiently to compare multiple models with one another. The Root Mean Squared Error (RMSE) is the

square root of the MSE. It is used commonly as the MSE might become large quickly and, as a result, get more difficult to compare easily. A similar metric as the MSE is the MAE, which is defined by the average absolute error. However, the MAE is less sensitive to far-off predictions, giving the same weight to each error, while the MSE gives an increasingly large penalty to larger predictions errors.

To conclude, the R-square or adjusted R-square is preferred whenever considering only a single model, because the performance can be quantified as a percentage of the variability of the output. When comparing the performance of a multitude of models, the metrics MSE (or the root mean squared error) or MAE are generally preferred as they give a more intuitive representation of the error. In the case that large prediction errors need to be penalized, the MSE provides a great solution, while the MAE can be used whenever different predictions errors should have the same weight. The RMSE is often preferred whenever values get too big, while the MSE is a reasonable metric whenever values do not blow up.

3-3 Digital Image Processing

Digital image processing is the use of a digital computer in order to process digital images through algorithms. [68] During the project, a dataset was created by making use of two existing digital image processing techniques, namely: Hough Circle Transform (HCT) and template matching. First, the HCT will be explained in subsection 3-3-1. Then, the concept of template matching will be elaborated on in subsection 3-3-2.

3-3-1 Hough circle transform

The HCT method is a method within digital image processing which allows circles to be extracted from an image, even if circles are not complete. The HCT method was patented in 1962 by the researcher Hough and is still used frequently today [69]. The method tries to extract possible parameter values of a circle by making use of a constraint equation. A circle has the parameters of its centre coordinates (a, b) and its radius r, which can be related to the location of edge points (x,y) through the following constraint equation:

$$(x-a)^2 + (y-b)^2 = r^2. \tag{3-19}$$

Although implied in Equation 3-19, it should be recognized that the three parameters (a,b,r) are required for the description of a circle in a two-dimensional space, while any (x_i, y_i) are merely edge points on this circle.
Then, any edge point (x_i, y_i) can be a point on any circle whose parameters lie on the surface of a right circular cone in the three-dimensional (a,b,r) parameter space [70], as illustrated in Figure 3-5. When cones of multiple edge points intersect at a single point in the (a,b,r) space, a circle can be identified with those parameters.

Multiple circles of unknown radii can be detected in a single image using this technique. First, edges of possible circles are detected using a filtering algorithm. Then, by iterating over these points (x_i, y_i) with possible parameters (a_i, b_i, r_i), "votes" can be collected in the accumulator matrix. The accumulator matrix is a separate array of counters, which accumulates the "votes" for each edge point (a_i, b_i, r_i). At the end, the array entries which contain high numbers are very likely

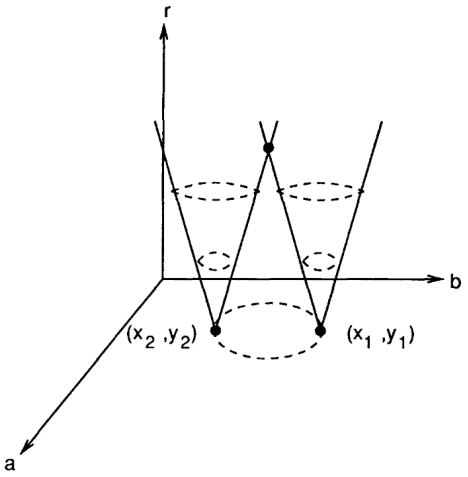


Figure 3-5: Different edge points (x,y) create cones in a three-dimensional parameter space (a,b,r). Whenever multiple cones intersect, a circle is defined with the corresponding parameters [70].

to present the shape of a circle in the image with the corresponding parameters. An algorithm for the detection of circles in an image of size (a, b) can now be constructed based on the aforementioned theoretical background. An overview of this algorithm is displayed in Figure 3-6.

The obvious problem with the algorithm as presented in Figure 3-6 is its inefficiency, as it copes with iterating over values in a three-dimensional parameter space. Furthermore, it will likely deal with data sparsity issues, as only a few entries in the accumulator matrix would be filled. Fortunately, this problem can be overcome using a method known as the Hough gradient method.

OpenCV Hough circle transform algorithm

This work will make use of an algorithm known as OpenCV Hough Circle Transform, which is an HCT algorithm making use of the Hough gradient method [4]. It is publicly available in the computer vision library OpenCV and can be used as a library within the coding environment Python. The theory behind the detection algorithm was explored, as numerous optimization parameters must be tuned before desired results can be achieved. An explanation of both the Hough gradient method and the OpenCV HCT algorithm will therefore be given in the following.

The Hough gradient method is a method takes the gradient into account in its search for circles. In the earlier described variation of the HCT method, circles in the two-dimensional (a,b) space could be identified by 'drawing' inverted cones in the three-dimensional parameter space (a,b,r) while incrementing the entries in the accumulator matrix. The Hough gradient method improves efficiency as now, instead of 'drawing' complete circles, only the entries in the accumulator matrix in the direction of the gradient of each edge pixel are considered.
The OpenCV HCT algorithm makes use of the Hough gradient method and can be broken down into two subsequent stages. First, the candidates for the centres of potential circle are identified. Second, the best radius is computed for each centre candidate. Both stages will be explained in the following part.

[4] `https://docs.opencv.org/3.4/d4/d70/tutorial_hough_circle.html`, Date accessed: 14-08-2021

Hough Circle Transform (HCT) code overview

1: Initialize accumulator array A[a,b,r] to a zero array
2: Detect edges (x,y) in image using any edge detection algorithm
3: Iterate over r, where $0 \leq r \leq \sqrt{a^2 + b^2}$
    4: Iterate over the edge pixels (x,y) in image
        5: Iterate over the degrees in a circle, where $0 \leq \theta \leq 360$
            6: Compute a: $a = x - r * \cos(\theta)$
            7: Compute b: $b = y - r * \sin(\theta)$
            8: Increment value: A[a,b,r] = A[a,b,r] + 1
9: Set suitable threshold value
10: Find [a,b,r] values in A[a,b,r] above the threshold value

Figure 3-6: Hough circle transform code overview

The edges within the image are found using the Canny edge detector [71]. Information about the gradients for each image pixel are calculated using the Sobel operator [72]. Then, for each edge pixel, only the entries in the accumulator matrix that are located in both the directions of the gradient will be incremented. Finally, the circle centres can be identified by the entries in the accumulator matrix that are both above a certain threshold and are a local maximum. Now that the circle candidates have been identified, the new stage of finding the optimal radius for each candidate begins. It is important to notice that the parameter space is reduced to a one-dimensional space, and thus a new one-dimensional accumulator matrix will be used. For each candidate, an accumulator matrix will be created by computing the distance to all the edge pixels and incrementing accordingly. The optimal radius will then be the radius corresponding to the highest incremented counter, supported by most edge pixels. This process can be repeated for all candidate centres. Selected centres that lie too close to centres that were found previously and have less support of edge points (lower counter values) are discarded. The OpenCV HCT algorithm has multiple parameters that can be tuned and can be related to the theoretical description of the algorithm as explained above. The parameters are explained in Table 3-1 based on the description given by OpenCV [5].

[5] `https://theailearner.com/tag/hough-gradient-method/`, Date accessed: 10-04-2022

Parameter	Description
image	An 8-bit, single-channel, grayscale input image.
method	Choice of method between 'Hough gradient' and a variation known as 'Hough gradient alt'.
minDist	Defines the minimum distance in pixels between the centres of detected circles. All candidates below this distance are discarded.
param1	Defines the threshold given to the Canny edge detector. A higher threshold leads to more edge points, higher values in the accumulator matrix, and thus more candidate circle centres.
param2	Defines the threshold for the accumulator matrix. A higher threshold leads to more detected circles but increases chances of false positives.
minRadius	Minimum radius in pixels considered for any circle.
maxRadius	Maximum radius in pixels considered for any circle.

Table 3-1: List of parameters of the OpenCV HCT algorithm for circle detection in an image.

3-3-2 Template matching

Template matching is a technique in digital image processing for finding parts in an image that match a template image [73]. The concept can be explained by having the smaller template image "slid" across the original image, keeping track of the locations of possible matches. In a mathematical sense, a search image $S(x, y)$ and a template image $T(x_t, y_t)$ can be presented as an example, where (x, y) and (x_t, y_t) are the coordinates of the pixels in the search image and template image respectively. The template image then moves one pixel at a time across the search image, while at the same time a metric, representing the matching accuracy, is computed and stored in the result matrix R. The entry within R with the highest value then represents the (x, y)-location within S with the highest probability of a match with T. A visual representation of this process is given in Figure 3-7.

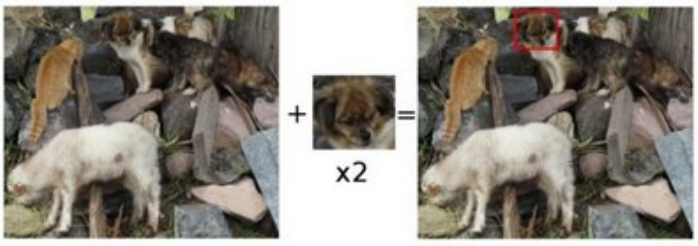


Figure 3-7: Example of a search within an original image (left) using a template image (middle). The template image is then identified by the algorithm and highlighted as a red rectangle (right).

Template matching is effective whenever the exact template can be found in the unseen search image. However, template matching algorithms are not robust against changing perspectives and

external influences, such as: a dynamic camera changing its point of view, angle and intensity of light, or three-dimensional movements of the elements within the template image. Furthermore, template matching can be computationally expensive whenever large search images are used in combination with large template images. However, whenever the circumstances allow it, template matching is a relatively simple and easy to implement technique for identification of a part within an image and can achieve high accuracy results. Moreover, when compared to more advanced image recognition algorithms like convolutional neural networks, it does not need much prior information. Instead, it only needs a template image and a search image.

OpenCV template matching algorithm

The open-source computer vision library OpenCV comes with a function for the purpose of template matching [6]. The function consists of three input parameters, which are listed and described in Table 3-2.

Parameter	Matrix symbol	Description
Search image	S	A gray-scale input search image.
Template image	T	A gray-scale input template image, which can not be bigger in size than the search image.
Value matrix	R	A choice of 6 different metrics to compute all values in the value matrix. All metrics quantify some form of correlation between the template image and an equal-sized part of the search image.

Table 3-2: List of input parameters of the OpenCV template matching algorithm.

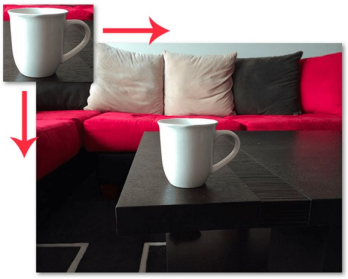
OpenCV offers six different methods for the computation of the value matrix R in the template matching function. A mathematical formulation of all metrics can be found in the document files of OpenCV [7]. Out of all six methods, the normalized correlation coefficient is typically used to determine how "similar" the intensities of the pixels of the two images are. Each entry (x, y) in two-dimensional matrix R is then computed using this metric by sliding template T over search image S. The matrix R is constructed in such a way that each (x,y)-coordinate of S also has an entry in R, provided that the size of the patch in S has the same width and height as T. When R is fully computed, and all patches of S have been compared to T, the highest values in R indicate the (x,y)-locations of the best matches.

The normalized correlation coefficient will be defined in the following. Let the S be of size $[W \times H]$ and T of size $[w \times h]$. The size of R then follows and is equal to $[(W - w + 1) \times (H - h + 1)]$, since T is slid across all parts of S and the entire template T should fit in S. For $x' = 0, \ldots, w - 1$ and $y' = 0, \ldots, h - 1$, the normalized correlation coefficient can be computed as follows:

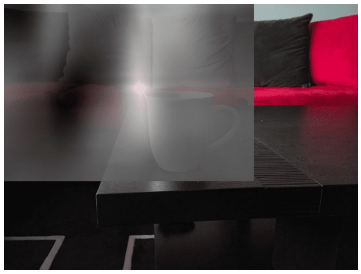
$$R(x, y) = \frac{\sum_{x',y'} \left(T\left(x', y'\right) \cdot S\left(x + x', y + y'\right) \right)}{\sqrt{\sum_{x',y'} T\left(x', y'\right)^2 \cdot \sum_{x',y'} S\left(x + x', y + y'\right)^2}}. \tag{3-20}$$

[6] `https://docs.opencv.org/4.x/d4/dc6/tutorial_py_template_matching.html`, Date accessed: 18-12-2021
[7] `https://docs.opencv.org/4.x/de/da9/tutorial_template_matching.html`, Date accessed: 18-12-2022

Again, the values for the normalized correlation coefficients are saved in the value matrix R. When R has been fully constructed, the high values in R indicate good matches, while low values indicate bad matches. A visual interpretation of R is illustrated in Figure 3-8b, which relates high values in R to bright pixels and low values in R to dim pixels. In addition, it can be noticed from Figure 3-8b that the sizes of R and S do not correspond. As described earlier, this is because the entire template T should fit in S before a value of $R(x, y)$ can be computed.



(a) T (top-left image) and S.



(b) Visual interpretation of R (top-left in front), and S (background).

Figure 3-8: Visual representation of the OpenCV template matching algorithm, where T will be slid over S as shown in **(a)**. Metric values, representing the correlation between T and patches of S, are stored in R, as illustrated in **(b)**, where high/bright values correspond to a good match.

Chapter 4

System Modelling

This chapter will provide the theoretical modelling framework of this work, which consists of a description of the system and a proposal of the prediction model. Section 4-1 will first describe a general model of a football game. Section 4-2 will then formulate a simplified model of a football game by applying various abstractions to the general model based on Markovian theory. Section section 4-3 will then formulate the system as a Markov Decision Process (MDP). Finally, section 4-4 will propose the prediction model as used throughout this work.

4-1 General Model

The abundance of data in a game of football opens up many possibilities for further research within the field of prediction. However, a careful approach needs to be taken when dealing with large quantities of data. A mathematical model will help to formulate the problem in a structured way by defining the problem in mathematical language to describe the behavior of the system. A top-down modeling approach will be applied, in which a general model will be defined first. Subsequently, abstractions of the general model can be formulated, where less relevant parts of the data will be left out for simplification purposes. Finally, the concept of Markovian Models, as described in section 3-1, will then be used to mathematically describe the course of a football match.

4-1-1 The state vector of a football match

A real football match arguably has infinitely many state variables involved in its description. Examples of these state variables are the temperature of the grass, muscle soreness of individual players and perhaps even the single molecules and atoms involved in the game, which all contribute to the evolution of a match and portray the extensiveness of the real state set. Of course, it is generally not possible to model an almost infinite amount of state variables. The problem to solve will simply become too large and complex. Therefore, in practice, state vectors will be an abstraction of the actual state vector, meaning that only a relatively small amount of state variables are selected for modeling. In the case of football, these state variables include: the positional coordinate data of

players and ball x_p; and contextual information about the match x_c. These vectors can be captured in the state vector of the game, defined as follows:

$$x(t) = \begin{bmatrix} x_p(t) \\ x_c(t) \end{bmatrix}. \tag{4-1}$$

In Equation 4-1, $x(t)$ is the n-directional state vector consisting of: a n_p-directional vector $x_p(t)$ describing the exact location coordinates of all players and ball with respect to the football field at time t; and a n_c-directional vector $x_c(t)$ describing all possible contextual information about the match at time t. Contextual information includes: the team in ball possession, player positions, passes given, shots, etc.; but also individual player statistics, such as shot accuracy, distance covered, successful tackles, etc.

4-1-2 System behaviour

The behavior of the non-linear system can now be described by a general function,

$$x(t+1) = f(x(t), x(t-1), x(t-2), ..., x(1), x(0), d(t)), \tag{4-2}$$

with d(t) being some unknown, stochastic disturbance changing over time. The disturbance follows from the unknown variables influencing the state evolution. The game starts at in initial state $x(0)$ and ends in final state $x(T)$, which represent the kick-off and final whistle of the game respectively.
Although the function f might exist, the prediction of the entirety of the next states might very well be impossible due to the size and stochasticity of the problem. Besides, exploiting all past states from a game will make the problem extremely computational intensive, while at the same time making it increasingly difficult to comprehend. A less complex problem will therefore be created by abstracting the general model and, subsequently, convert it into a MDP model. Various heuristics and simplifications will be used to help in this process.

4-1-3 Model abstractions

The general model of Equation 4-2 will be abstracted using three different heuristics. Each heuristic will help reduce the complexity and solvability of the problem. Furthermore, the heuristics are necessary requirements for the creation of the MDP model, as will be proposed in section 4-3. The model heuristics are as follows:

- **Heuristic 1:** Instead of predicting the entirety of the state vector, a single meaningful metric shall be created. The selection process and design of the metric will be discussed in subsection 4-4-1.
- **Heuristic 2:** Only a selection of state variables shall be considered for the state vector, as not all state variables are relevant for the prediction of the metric. The selection process of this state vector will be discussed in subsection 4-3-1.
- **Heuristic 3:** The continuous state variables shall be discretized to fit the MDP model. The state discretization will be presented in subsection 4-3-2.

Two relevant implications of the above heuristics should be noted. First, heuristic 2 implies that the length of the state vector shall be finite. Second, heuristic 3 implies that all state variables in the state vector shall be discrete. Both implications are taken into account in the design of the Markovian models.

4-2 Markovian Models

The system can be described as a MDP following Markovian theory, as explained in section 3-1. In the following parts, a Markov chain and a Markov chain model of a football game will be formulated. Afterwards, the Markov chain model will be extended to a MDP model.

4-2-1 A Markov chain description

The system can be described by means of a Markov chain, described by a 2-tuple (S, P), as theorized in subsection 3-1-1. The state set S is the set of possible states after discretization of the state vector and P is the set of state transition probabilities between those states. Each state $s \in S$ describes the current state of the football match, featuring a configuration of the exact locations of all players and ball and some contextual match information.
Now, let each state variable $s_i \in s$ be bounded on both sides such that S can be defined as a finite set, where:

$$s \in S \Rightarrow s \in \mathbb{Z}^n, \quad \|s\|_1 < \infty.$$

Now consider the set of transition probabilities:

$$P : S \rightarrow \mathbb{P}^n, \quad \text{with} \quad P \in \mathbb{R}^{n \times n},$$

There exist n^2 state transition probabilities, as there are n possible transitions from each of the n states in S. P therefore consists of $(n-1)n$ unknown parameters, as the last parameter can be deduced from the fact that the sum of all probabilities add up to one.

The system moves in discrete-time, where at each time instance the system switches state or stays in the same state following certain probabilities. Of course, not all state transitions are possible and will therefore have a zero probability of happening. For example, a player can not simply move from one part of the pitch to the other without crossing everything in the middle. When a whole match is played, each tick of the clock will force a state transition, where each transition has its own probability.
A Markov chain of an entire match can now be constructed,

$$\big(x_0, x_1, x_2, \ldots, x_T\big),$$

where each state x_i with $i = \{0, 1, 2, \ldots, T\}$ is confined in set S and the probabilities of all possible state transitions are defined in P.

4-2-2 A higher-order Markov chain description

The system can now be described by means of a higher-order Markov chain, as explained in subsection 3-1-2. The higher-order Markov chain is defined as a new 2-tuple $(\tilde{S}, \tilde{P})$, where future states are conditionally dependent on the past m states. A new chain $(\tilde{x}_t)$ of m states can be constructed for $t > m+1$ as follows:

$$\tilde{x}_t = \begin{bmatrix} x_t^T & x_{t-1}^T & \dots & x_{t-m+1}^T \end{bmatrix}^T. \tag{4-3}$$

Let the chain satisfy the Markov property as defined in Equation 3-7. For easier notation, the length of the state vector $\tilde{x}(t)$ will be set equal to n_s, where:

$$n_s = n \times m. \tag{4-4}$$

A new finite set $\tilde{S}$ can be constructed such that each possible state $\tilde{x}_t$ is confined within $\tilde{S}$. Each state $\tilde{s} \in \tilde{S}$ is given by a concatenated sequence of m states from S, where each $s \in S$ is bounded on both sides. The following proposition therefore holds:

$$\tilde{s} \in \tilde{S} \Rightarrow \tilde{s} \in \mathbb{Z}^{n_s}, \quad \|\tilde{s}\|_1 < \infty. \tag{4-5}$$

In contrast to the states $s \in S$, the states $\tilde{s} \in \tilde{S}$ now have memory, although the total size of the set $\tilde{S}$ has become exponentially larger. A visualization of the updated state $\tilde{x}_n$ sequence is presented in Figure 4-1.

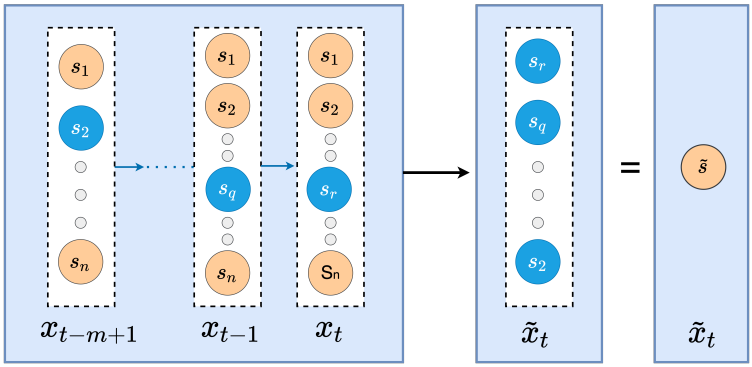


Figure 4-1: An example of the creation of a single state $\tilde{x}_t$ inside the higher-order Markov chain.

Now consider the updated set of transition probabilities:

$$\tilde{P} : \tilde{S} \to \mathbb{P}^{n_s}, \quad \text{with} \quad \tilde{P} \in \mathbb{R}^{n_s \times n_s}.$$

In comparison to a regular Markov chain, which consists of $(n-1)n$ parameters, the total number of unknown parameters of a Markov chain of order m increases exponentially to $(n-1)n^m$ [74]. A football game can now be modelled as a higher-order Markov chain,

$$\left(\tilde{x}_0, \tilde{x}_1, \tilde{x}_2, \dots, \tilde{x}_T\right),$$

where each state $\tilde{x}_i$ for $i = \{0, 1, 2, \dots, T\}$ is confined in the finite set $\tilde{S}$ and all state transition probabilities are defined in $\tilde{P}$.

4-2-3 A Markov decision process description

The system can be extended to a 4-tuple $(\tilde{S}, A, \tilde{P}, \theta)$, as explained in subsection 3-1-3, where $\tilde{S}$ is the state set as defined in Equation 4-5. The action space A consists of the actions $a \in A$ which the decision-maker (e.g. the coach, the manager) can take during the match, where each action corresponds to a formation of a team. A formation can be described as a set-up of players, usually organized by the team's coach or manager. Examples of formations are 4-3-3, 4-4-2, and 3-5-2. The numbers in the formations represent the number of defenders, midfielders, and attackers in a team respectively and exclude the goalkeeper. A decision was made to only include a single action in the action space, i.e. $n_a = 1$, as predictions about the evolution of the match can be made in a more controlled manner by focusing on a team following only a single action. As a result, within-game outcomes will not be influenced by changing actions over time.
Now consider the transition probabilities:

$$\tilde{P} : \tilde{S} \to \mathbb{P}^{n_s}, \quad \text{with} \quad P \in \mathbb{R}^{n_s \times n_s}, \tag{4-6}$$

where each entry $P_{ss\prime}$ is the probability that (action a in) state $s \in \tilde{S}$ will lead to state $s\prime \in \tilde{S}$, confined within the interval $[0, 1]$.
Finally, the metric reward function $\theta : \tilde{S} \to \mathbb{R}$ is defined as a function mapping any state $\tilde{s} \in \tilde{S}$ to some reward, where the reward is related to a goal-related metric. Already, a conceptual idea of this reward function can be obtained by defining the 'goal reward function', i.e., the reward function referring to the scoring of a goal. The goal reward function is defined by the reward function $R : \tilde{S} \to \{-1, 0, 1\}$, which maps the state $\tilde{s} \in \tilde{S}$ to a value in the set $\{-1, 0, 1\}$. A value of 1 corresponds to a goal being scored by the user team, a value of -1 corresponds to a goal being scored by the opponent team, and a value 0 defines all other cases.
An example of the MDP model is visualized in Figure 4-2, where only three states are considered. In practice, state set $\tilde{S}$ will contain much more states. Since only a single action is considered, probabilities and rewards are only dependent on the states.

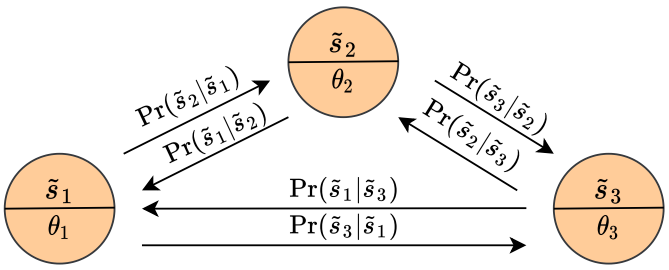


Figure 4-2: Representation of a simple MDP consisting of only three states, $|\tilde{S}| = 3$, including state transition probabilities and state-based rewards.

4-3 A Football Match as a Markov Decision Process

This section will go into more detail about how a football game can be described as the MDP as presented in subsection 4-2-3. The model heuristics as stated in subsection 4-1-3 will be implemented to acquire this model. The heuristics will simplify the general model of the system and

make it possible to formulate the system into this MDP framework. First, the state variables will be selected in subsection 4-3-1. Second, a discretization method for the state vector will be described in subsection 4-3-2. Third, a reward function will be chosen in subsection 4-3-3. Finally, a football match will be properly described as a MDP in subsection 4-3-3.

4-3-1 State vector proposal

The raw state vector of high quality football data sets includes detailed information such as the positional data of the players and ball, and various contextual variables (e.g., game mode, total passes, possession percentage). Finding the right state simplifications can be a challenging task as a trade-off needs to be made between two weighing factors. On the one hand, simplifying the state will lead to a simplification of the problem, which can lead to a better understanding of the problem and accelerate the process of finding a solution. On the other hand, simplification of the state vector already leads to the removal of valuable information, which could in turn drastically reduce the predictive potential of the algorithm.

State variable selection

The following state requirements were constructed to assist in the selection process:

- **SR01**: The state shall include the locations of all players and ball.
- **SR02**: It shall be possible to compute the value of the dangerousness metric at a single time instance from the state alone.
- **SR03**: The dimension of the state shall be kept as small as possible.

SR01 follows from the performance potential of using tracking data in the predictive algorithm, as discussed in the heritage in chapter 2. This means that the (x,y)-coordinates of the players and ball will be implemented in the state vector. Next, **SR02** is constructed, as it needs to be possible to compute the value of the metric for all time instances in order to make predictions. Besides, implementation becomes straightforward whenever the metric is a mapping from the state. **SR03** was set as a requirement to keep the problem as simple as possible, without losing valuable information. Furthermore, following the theory of Artificial Neural Network (ANN) as explained in subsection 3-2-2, one should only include information necessary for the output to be predicted. In the case of the project, it therefore makes sense to only include information that contributes to the computation of the metric. Adding extra information to the state vector will therefore not be relevant to the problem and could possibly lead to a sharp increase in the computational efforts of the ANN to find an accurate input-output relation. Besides, contextual information about within-game events can often be inferred from the tracking data over time (e.g. shots, goals, passes, etc.) and is therefore of less importance.

Based on the state requirements, it was chosen to include x_p and x_{poss} in the state vector. The vector x_p is defined as the (x_i, y_i) coordinates of all player and ball, where $i = \{1, 2, \ldots, 23\}$ refers to the 22 players of both teams and the ball. Scalar $x_{poss} \in \{-1, 1\}$ defines which team is in possession

of the ball, where -1 and 1 correspond to the opposing team and user team being in possession respectively. The chosen state variables together form a n-dimensional vector, where $n = 47$, and have been summarized in Table 4-1.

Variable name	Description	Vector size	State type	Evolution Mechanism
x_p	A vector containing the exact [x,y]-coordinates of the players of both teams.	46	Continuous	Time
x_{poss}	A value denoting the team who owns the ball (user team, opposing team).	1	Discrete	Event

Table 4-1: Summary of state vector variables.

Formulation of the state vector

The state variables can now be captured inside a state vector $\tilde{x}(t)$ that defines the state of the game at time t. This state vector is constructed as follows:

$$x(t) = \begin{bmatrix} x_p(t) \\ x_{poss}(t) \end{bmatrix}, \qquad \tilde{x}(t) = \begin{bmatrix} x(t) \\ x(t-p) \\ \vdots \\ x(t-pm) \end{bmatrix}, \tag{4-7}$$

where m defines the amount of past states and $p \in \mathbb{Z}^+$ is a parameter used for sampling. The amount of sampled states m and the sampling time p are both parameters to the model. Multiple values for both m and p were tested in an iterative way. Ultimately, the final fitted models on both data sets were given the following values for m and p, with p defined in minutes:

$$\begin{aligned} m &= 4 \\ p &= 5. \end{aligned} \tag{4-8}$$

With $m = 4$ and $p = 5$, the predictor function will make its predictions at time t based on its knowledge of $\tilde{x}_t$, which can now be updated as:

$$\tilde{x}_t = \begin{bmatrix} x_t^T & x_{t-5}^T & x_{t-10}^T & x_{t-15}^T. \end{bmatrix}^T. \tag{4-9}$$

Any state $\tilde{x}_t$ will be confined in set $\tilde{S}$, as described in Equation 4-5.

4-3-2 Discretization proposal

Discretization of the state vector is the process of transferring the continuous state variables into discrete state variables in order to reduce the problem such that it becomes more suitable for numerical evaluation and model implementation. A drawback is that information about the game is being thrown away intentionally. In this work, discretization of the state vector is necessary for the creation of a finite state set $\tilde{S}$ such that it can fit the MDP model as proposed in subsection 4-2-3.

As can be observed from Table 4-1, x_p contains the only continuous variables in the state vector. x_p will be discretized by dividing the football field into regions. Let $[d_1 \times d_2]$ be the size of the football field in metres. Now, let F be the set of n_r distinct regions on the field, defined as:

$$\begin{aligned} F = \{f_i\}_{i=1}^{n_r}, \qquad & f_i \subset [0, d_1] \times [0, d_2], \\ & f_i \cap f_j = \emptyset, \\ & \cup f_i = [0, d_1] \times [0, d_2], \qquad \forall i, j, \quad i \neq j. \end{aligned} \tag{4-10}$$

Choosing the right amount of regions is different for each project and should be approached thoughtfully, although field partitioning seems to be a common practice in football modelling. Various field partitionings have been proposed in the literature: Cotta et al. (2013) [75] used 9 regions for their football model; Narizuka et al. (2014) [76] proposed a 18-region model; Arriaza-Ardiles et al. (2018) [77] and Gamae et al. (2014) [78] both used a 24-region division.
For this project, it was chosen to divide the field into 40 regions of equal size, where 8 regions for the width (x-coordinates) and 5 regions are chosen for the height (y-coordinates), as this seemed to be a reasonable division for reward annotation, as will be further explained in subsection 4-3-3. Consequently, each continuous (x,y)-coordinate in x_p can now be converted to a discrete value such that the discrete (x)-coordinates and (y)-coordinates are all confined within the sets $\{1, 2, \ldots, 8\}$ and $\{1, 2, \ldots, 5\}$ respectively. A visualization of this division is displayed in Figure 4-3.

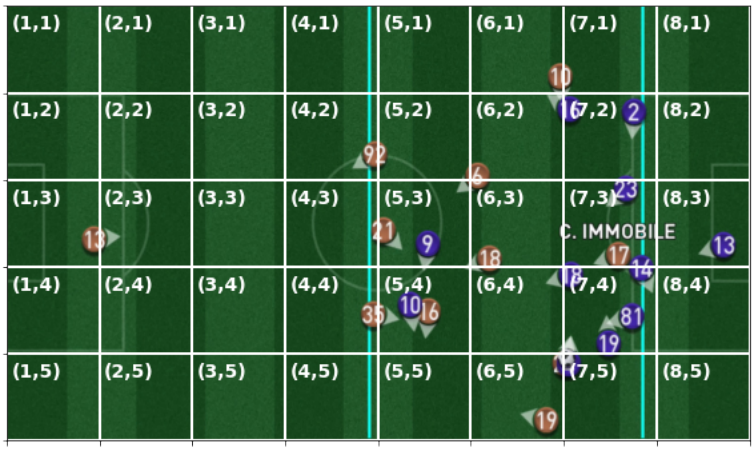


Figure 4-3: Field partitioning of the field into 40 regions of equal size. Each region is annotated with a specific discrete label, which is used for discretization purposes.

4-3-3 Reward function proposal

A reward gained at θ_t at time t can be computed using the reward function $\theta : \tilde{S} \rightarrow \mathbb{Z}$, which maps the state $\tilde{s} \in \tilde{S}$ to an integer value representing a form of immediate goal-related danger. The reward function uses the field division as proposed in subsection 4-3-2 and, subsequently, assigns a reward value to having possession of the ball in each one of the regions.

Recall the field division, where $F = \{f_i\}_{i=1}^{40}$. Let $d : F \rightarrow \mathbb{Z}_{\geq 0}$ be a function assigning a danger value d to each region $f \in F$. For example, region f_1 may have an annotated danger value

of $d(f_1) = 2$. Subsequently, regions containing the same danger value are bundled into n_θ different sets of regions $F_d \subset F$, such that:

$$F_d = \{f \in F : d(f) = d\}, \qquad d = \{0, 1, 2, \ldots, n_\theta\}. \tag{4-11}$$

Although implied, it is important to recognize that each $F_d \subset F$ contains distinct regions, such that:

$$F_i, F_j \subset F \Rightarrow F_i \cap F_j = \emptyset, \quad F_i \neq \emptyset, F_j \neq \emptyset, \qquad \forall i, j, \quad i \neq j.$$

Now consider two scenario's.

- *In scenario 1,* the user team is in possession of the ball.
- *In scenario 2,* the opponent is in possession of the ball.

Two separate and opposite metric functions $\theta_1 : \tilde{S} \to \mathbb{Z}_{\geq 0}$ and $\theta_2 : \tilde{S} \to \mathbb{Z}_{\leq 0}$ for scenario 1 and 2 respectively can be derived. They are formulated as follows:

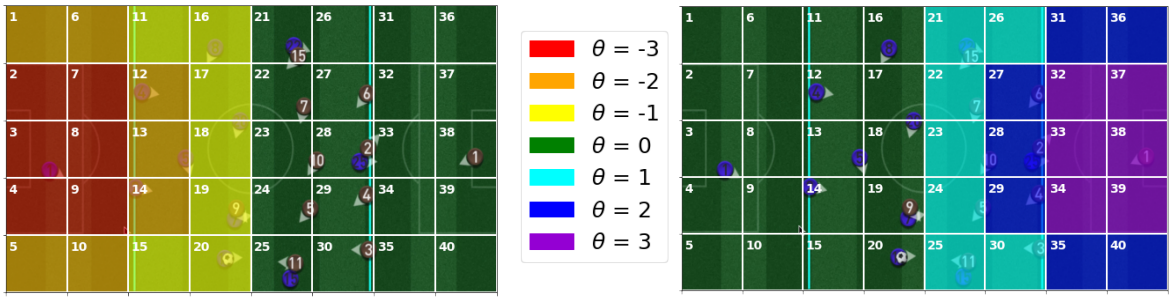
$$\theta_1(\tilde{s}) = \begin{cases} n_\theta & \text{if possession in regions } F_{n_\theta} \\ \vdots & \vdots \\ 2 & \text{if possession in regions } F_2 \\ 1 & \text{if possession in regions } F_1 \\ 0 & \text{otherwise} \end{cases}, \qquad \theta_2(\tilde{s}) = \begin{cases} -n_\theta & \text{if possession in regions } F_{n_\theta} \\ \vdots & \vdots \\ -2 & \text{if possession in regions } F_2 \\ -1 & \text{if possession in regions } F_1 \\ 0 & \text{otherwise} \end{cases} \tag{4-12}$$

A higher reward is given to having possession in more dangerous regions. The metric functions of Equation 4-12 can be merged into a single metric function $\theta : \tilde{S} \to \mathbb{Z}$ capturing all possible cases:

$$\theta(\tilde{s}) = \begin{cases} n_\theta & \text{if the user team in possession in attacking regions } F_{n_\theta} \\ \vdots & \vdots \\ 2 & \text{if the user team in possession in attacking regions } F_2 \\ 1 & \text{if the user team in possession in attacking regions } F_1 \\ 0 & \text{otherwise} \\ -1 & \text{if the opponent in possession in attacking regions } F_1 \\ -2 & \text{if the opponent in possession in attacking regions } F_2 \\ \vdots & \vdots \\ -n_\theta & \text{if the opponent in possession in attacking regions } F_{n_\theta} \end{cases} \tag{4-13}$$

To clarify, the annotated rewards for the regions on the pitch are flipped whenever possession switches sides. This means that F_{n_θ} will contain the most dangerous regions for both scenario's.

In this work, a design choice was made to have 7 different integer values of danger, such that $d = \{-3, -2, -1, 0, 1, 2, 3\}$. Consequently, regions containing the same danger value can be bundled into $n_\theta = 7$ different sets of regions $F_d \subset F$. A visualization of the corresponding reward annotation of the reward function is illustrated in Figure 4-4. The figures show both scenario's and visualize how the reward is distributed depending on the location of the player in possession of the attacking team.



(a) Scenario 1: The user team has possession

(b) Scenario 2: The opponent has possession.

Figure 4-4: Reward distribution of θ over regions $f \in F$ visualized. Two scenario's are sketched in the cases either team has possession of the ball. The value of θ of each region is indicated by various colors, where $\theta \in \{-3,-2,-1,0,1,2,3\}$.

4-3-4 Formulation of the Markov decision process model

The MDP model, as presented in subsection 4-2-3, given by $(\tilde{S}, A, \tilde{P}, \theta)$ can now be defined properly. First, $\tilde{S}$ is the state set where the states consist of the selected state variables as discussed in subsection 4-3-1. Second and third, $A = \{433\}$ is the action space, and $\tilde{P}$ are the set of unknown state transition probabilities, both discussed in subsection 4-2-3. Finally, $\theta : \tilde{S} \rightarrow \mathbb{Z}$ is the reward function as presented in Equation 4-12. The system description as presented by this MDP will be the basis of the prediction model, as will be further discussed in section 4-4.

4-4 Prediction Model Proposal

The MDP model enables the creation of the prediction model. The same model parameters as defined throughout section 4-3 will be used for the design of this prediction model. First, subsection 4-4-1 will present the definition and mathematical formulation of prediction metric as used in this work. Subsection 4-4-2 will then propose the prediction model, based on the proposed metric and the MDP model.

4-4-1 Metric proposal

It can be argued that goals are the only relevant metric within football because, at the end of a match, the team with the most goals wins and goes home with all the points. However, difficulties can arise when trying to find long-term patterns within data that lead towards increased probabilities of scoring goals. The build-up towards goals are influenced by a huge amount of weighted factors. What type of player movements will yield the highest goal scoring chances when the opponent team is playing a certain way? What type of passing combinations will lead to the highest probabilities of scoring a goal? When should a team play high pressure or use an offside trap in order to increase scoring possibilities? Finding a function which encompasses all factors accurately might be a difficult task, which even an ANN might not be able to handle. Another disadvantage of choosing goals as the metric to be predicted is that goals only occur sporadically and, consequently, a lot of game data is required to find relevant patterns in the build-up towards a goal. Problems may arise as available data of football games is scarce.

Metric requirements

In light of the reasons mentioned above, it was chosen not to select goals as the metric to be predicted. Instead, it was chosen to design a different metric. A thoughtful approach was taken to find a metric by setting certain requirements, based on the current literature and logical reasoning. The metric requirements are as follows:

- **MR01**: The metric shall rely on available data and shall include tracking data.
- **MR02**: The metric shall be able to quantify performance on a team level and shall be strongly correlated with goals.
- **MR03**: The metric shall occur a reasonable amount of times during a match.
- **MR04**: The metric shall be designed and constructed within reasonable time.

The first metric requirement **MR01** is based on the promising performance potential of including tracking data into a prediction algorithm, as found in the heritage in subsection 2-1-1. The requirement **MR02** naturally follows from the importance of the ultimate football metric, which is goals. Furthermore, the heritage of subsection 2-1-1 showed that the predictions in football can be made most accurately when considering metrics on a team-level. **MR03** follows from the data properties needed to train a ANN, thoroughly explained in subsection 3-2-2. Finally, **MR04** aims to keep in mind the time boundaries of this thesis project. Large data sets with an immense amount of variables can quickly lead to complex metrics and, as a result, writing an algorithm for such a metric could be very time-consuming.

Apart from well-known metrics (e.g., the number of passes of a team, the shots on target, the possession percentage, etc.), a lot of research has been conducted in capturing data into accurate and understandable Key Performance Indicators (KPI's). Especially the rise in the availability of tracking data of the past decade has lead to the development of more advanced KPI's [17], which can be linked to implications such as goal scoring chances, territorial advantages, and successful passing networks. A large review of KPI's was conducted during this work and is as presented in Appendix A-2. These metrics were used as inspiration for the metric as designed in this project.

Metric selection and definition

After reviewing a broad range of metric possibilities and keeping in mind the boundaries of the above requirements, it was chosen to create the Expected Danger (ED) as metric for further research. The metric is inspired by the work of Cervone et al. (2014) [15], who have proposed a framework for calculating the expected possession value (Expected Possession Value (EPV)) in basketball. The EPV is a real-time estimate of the expected points obtained at the end of a single possession phase. They propose a two-level Markov chain model in which the continuous-time movements of players and ball, and the discrete-time events (e.g. shots, turnovers, passes) are combined hierarchically for real-time computation of the EPV. To date, the EPV still remains a popular measure and is still used in the American NBA.

The metric is defined as follows:

Definition 4-4.1. *The Expected Danger (ED) is defined as a form of expected attacking danger during any moment during a football game and is based on the evolution of the state vector over time. The ED is a real number and expresses the likelihood that possession of the ball will be held within dangerous territories for either the user team or the opponent team over a given prediction horizon.*

In mathematical terms, the expected danger function $ED^h : \tilde{S} \rightarrow \mathbb{R}$ is defined as the expected average reward over a fixed time interval $[t+1, t+h]$, where t is the current time, h is the fixed prediction horizon, and time T represents the final whistle of the game:

$$ED^h(\tilde{s}) \quad = \quad \mathbb{E}\left[G_t^h \mid \tilde{s}\right] \quad = \quad \mathbb{E}\left[\frac{\sum_{k=1}^{\min(h,T-t)} \theta_{t+k}}{\min(h, T-t)}\right]. \tag{4-14}$$

As shown in Equation 4-14, G_t^h is the actual average reward and is computed by summing the rewards θ_t over time interval $[t+1, t+h]$ and adjusting for the length of the horizon h. It is similar to the undiscounted sum of rewards as used in many reinforcement learning algorithms and can also be defined separately. Given that $t + h \leq T$:

$$G_t^h = \frac{1}{h} \sum_{k=1}^{h} \theta_{t+k}, \tag{4-15}$$

where θ_t is defined as the reward at time t. Furthermore, it is important to recognize that since $\theta : \tilde{S} \rightarrow \{-3, -2, -1, 0, 1, 2, 3\}$, it follows that $G_t^h \in [-3, 3]$ and therefore also $ED^h \in [-3, 3]$ for any horizon h and any $\tilde{s} \in \tilde{S}$.

4-4-2 Predictor function proposal

Making accurate predictions is only possible whenever the state transition probabilities can be estimated. As discussed in subsection 3-1-1, a common technique for the estimation of the state transition probabilities is based on the maximum likelihood estimator. However, the major issue of MDP model of this project is its size. Although a higher-order MDP gives possibilities to include time dependencies, it does come at the cost of an increased number of parameters, i.e., the unknown transition probabilities. In comparison to a regular MDP, which consists of $(n-1)n$ parameters, the total number of parameters of a MDP of order m increases exponentially to $(n-1)n^m$. This potentially huge amount of parameters can cause considerable estimation problems and is therefore not recommended for problems with a large state space and a value m chosen large [74]. In the case of this project, the total number of parameters is equal to $(n-1)n^m \approx 2 * 10^8$. The estimation of the transition probabilities using the maximum likelihood estimator was therefore discarded for this project. Instead, an ANN approach for the computation of the ED was pursued and will be proposed in the following.

One of the major advantages of an ANN approach for within-game predictions is that only a small, relevant subset of the set of transition probabilities need to be estimated. The ANN will learn to identify the most important features within the web of interconnected states $\tilde{S}$, rewards θ and state transition probabilities $\tilde{P}$ of the MDP to, ultimately, compute a value for the ED. In contrast to the theoretical MDP model, which has full knowledge of the dynamics of the abstracted system, the ANN model works as a black-box model. The ANN will create an input-output function focusing solely on the mapping of the state $\tilde{s} \in \tilde{S}$ to the ED, thereby not revealing any additional information about the system. This process is illustrated in Figure 4-5, which shows the ANN model encapsulates the MDP without knowing its specifics.

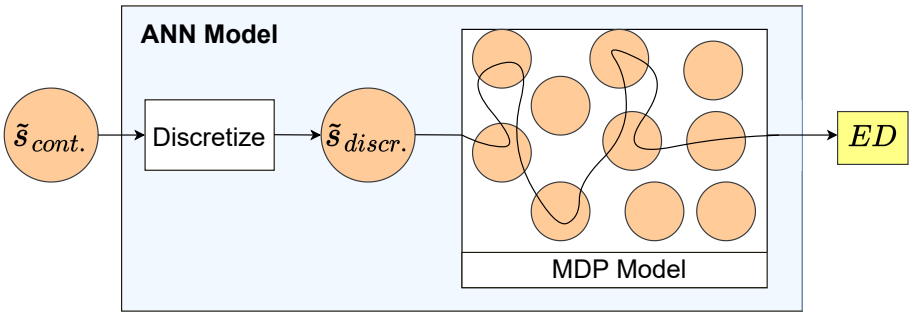


Figure 4-5: Schematic representation of how the ANN model operates relative to the MDP model.

As explained in subsection 3-2-2, there exist different classes of ANN's. A choice was made to design and build the prediction model by training a Multilayer Perceptron (MLP), i.e., a class of ANN explained in detail in section 3-2-2. A MLP model was chosen for the following four reasons. First, within the context of ANN's, the architecture of an MLP's is relatively basic (with only feed-forward signals), making them easier to tune and optimize as fewer hyperparameters exist. Second and third, they can be used for complex, non-linear problems and work great for large input data sequences. Fourth, fast predictions can be made when training is completed, thereby making them suitable for an online prediction algorithms. The objective for the prediction model can now be defined as follows.

Objective: *Train and tune a MLP for regression using labeled input-output pairs of the state $\tilde{s} \in \tilde{S}$ and the corresponding average reward G_t^h. After training, the MLP function shall be able to compute the ED; a real-time metric expressing the likelihood of having possession in dangerous territories over a future horizon.*

Training and tuning will be done using labelled input-output pairs, where the input layer will consist of the state $\tilde{s} \in \tilde{S}$ and the output value will be equal to the corresponding average reward $G_t^h \in [-3,3]$. A regression algorithm was used since the output is a continuous value. A graphical representation of the labelled examples is given in Figure 4-6.

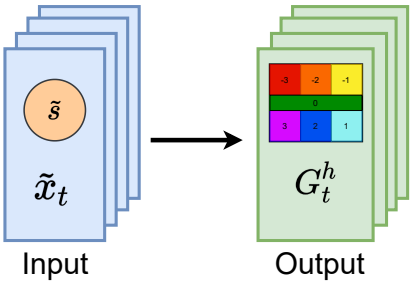


Figure 4-6: Labelled input-output examples.

The input variables represent the state of the game at $m = 4$ time instances, which includes the state of the game at current time t, and is sampled with a sampling time of $p = 5$ minutes between each state observation. The output variable represents the average reward over horizon $[t+1, t+h]$. Furthermore, the input vector is of size $n_x = 188$ and the output value is of size $n_y = 1$.
After the training phase, the predictor function created by the MLP will have learned to map unseen states $\tilde{s} \in \tilde{S}$ to the ED. A schematic overview of the trained MLP is illustrated in Figure 4-7.

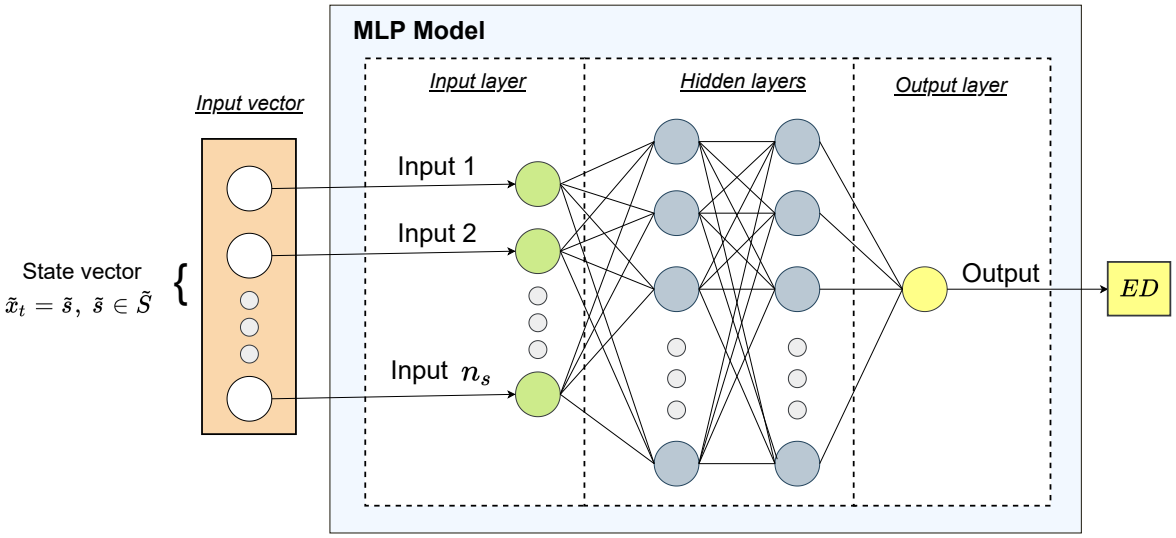


Figure 4-7: Schematic overview of a MLP used for regression. All entries of the state vector $\tilde{x}_t = \tilde{s}$, with $\tilde{s} \in \tilde{S}$, are used as inputs. The output will be the ED.

Different MLP's will be trained for the different horizons $h = \{5, 10, 15, 30, 45\}$. Since 5 horizons were selected, a total of 5 predictor functions will be trained. The MLP for $h = 5$ will then be trained using $G_t^{h=5}$ as an output value, the MLP for $h = 10$ using $G_t^{h=10}$ as an output value, and so on. The goal of the training phase is to find the right combination of nodes, connections, and weights such that the input-output relationship between any state $\tilde{s} \in \tilde{S}$ and the ED can be obtained. The predictor functions will all be trained using the parameters as defined in this chapter. All these parameters have been summarized in Table 4-2.

Parameter	Description	Value(s)
n	The length of the initial state vector $s \in S$.	47
m	Order of the higher-order MDP model, defined by the amount of concatenated states $s \in S$ that make up state $\tilde{s} \in \tilde{S}$.	4
n_s	The length of the updated state vector and any $\tilde{s} \in \tilde{S}$.	188
p	Sampling time, defining the temporal distance between two consecutive states $s \in S$ in $\tilde{s} \in \tilde{S}$.	5
n_r	Total number of regions in F, composed of 8 regions along the length and 5 along the width of field.	40
h	Prediction horizon, representing the time interval $[t+1, t+h]$ considered for prediction.	$\{5, 10, 15, 30, 45\}$
n_θ	Defines the maximum and minimum reward value, such that $\theta = [-3, 3]$.	3
d	Reward values, defined as the reward annotated to each region $f \in F$.	$\{-3, -2, -1, 0, 1, 2, 3\}$
G_h^t	Average reward, defined as the average reward over time horizon $[t+1, t+h]$	$[-3, 3]$
ED^h	Expected danger, defined as the expected average reward over prediction horizon $[t+1, t+h]$	$[-3, 3]$

Table 4-2: Description of the parameters of the MDP model.

Chapter 5

Datasets

Machine Learning (ML) models are only as good as the datasets that are used to create them. It is therefore of crucial importance to find and shape a dataset to the needs of the project. This chapter will discuss the process of finding and creating a suitable dataset, based on range of available data sources. Section 5-1 will first discuss the process and methodology of selecting the data source most suitable to the project. After the data source is selected, section 5-2 will describe how the dataset is created. Finally, an overview of the dataset will be given in section 5-3, which will describe its contents and highlight various interesting features.

5-1 Dataset Selection

Before selecting the source of the data, it is important to follow a methodological process of selection. Numerous criteria for the dataset are therefore created in subsection 5-1-1, and serve as an objective selection tool. Next, a selection of promising data sources and a dataset comparison will be made in subsection 5-1-2. At last, a short explanation will be given on how the datasets are going to be realized, which will be done in subsection 5-1-3.

5-1-1 Dataset requirements

Numerous dataset requirements were constructed tailored to the needs of the prediction algorithm as proposed in section 2-2.

- **DR01:** The dataset shall contain the tracking data of the players and the ball
- **DR02:** The dataset shall contain at least 15 games of data and be large enough to capture the relationship between the input and output variables
- **DR03:** The dataset shall contain data as realistic as possible

- **DR04:** The dataset shall contain the specific event data needed for the creation of the Expected Danger (ED)

Requirement **DR01** follows from the way the ED danger metric is constructed, that is, as a mapping of the positions of all players and ball. Dataset requirement **DR02** is based on algorithm requirement AR01, as given in section 2-2, which states that the algorithm shall be trained on a large-scale dataset of at least 15 games of data. The reason behind this requirement is to have a diverse profile of different football games. As a result, the Artificial Neural Network (ANN) can be trained on numerous games, thereby making it more robust to unseen data of other football games. In addition, a minimum number of games increases the chances of creating an accurate predictor, as ANN-based algorithms generally need a large amount of sample data to be able to generalize the data. Requirement **DR03** follows from the heritage as described in subsection 2-1-2 and is based on the notion that simulation data should only be considered for research whenever the gameplay of the simulator closely matches that of actual football gameplay. Even the most advanced football simulators experience the so called simulation-to-reality gap [47], and datasets based on actual football are therefore preferred. The final requirement **DR04** is straightforward and naturally follows from way the ED metric is defined, as proposed in subsection 4-4-1. This means that the dataset shall include information about the team in possession.

5-1-2 Candidate data sources and dataset selection

Based on the availability of data as described in the heritage in subsection 2-1-2, numerous candidate data sources were identified. The identified data sources consist of existing datasets as well as possible methods to create a new dataset. They are summarized in Table 5-1.

The selection process will take place by comparing and evaluating the different data sources, as presented in Table 5-1, based on the requirements as given in subsection 5-1-1. Three data acquiring approaches were considered and will be discussed and evaluated in the following part. The most straightforward approach is to try and find existing datasets containing either event data or tracking data and try to merge them. Two very large and publicly available datasets containing only event data were found [40] [46] and two existing datasets were found containing only tracking data [45] [46]. However, the datasets based on tracking data only contain 4 games of data and do therefore not meet the dataset requirement DR02.

Another approach is to use existing automatic event detection algorithms and automatic player and ball tracking algorithms to create or complement datasets. Various studies focusing on event detection were able to achieve high-accuracy performance in a few event classes, namely: passes, shots, and goals [37] [38] [39]. Similarly, two studies that developed a player tracking and detection algorithm [43], and a ball tracking algorithm [44] achieved reasonable performance with detection accuracies of 92.8% and 97.3% respectively. By combining three different existing data extraction algorithms for player-, ball, and event detection, it might be possible to create or complement a dataset. The drawbacks of this approach, however, are the flawed accuracy and the uncertainty whether the detection algorithms can be practically implemented within the time boundaries of the project.

The final approach is to use simulation data of the Google Research Football (GRF) or the eFootball dataset, which provide possibilities to indirectly extract the actual underlying data. Simulators provide the possibility to generate more data if needed, and allow controlled experiments to be executed. Furthermore, the heritage showed that state-of-art simulators, such as the eFootball

	Description	Type data	Size	References
1	Existing dataset of three games by the Norwegian club Tromsø IL, where only the home team got tracked.	Tracking data	Three 90-minute matches measured at 20 Hz.	Pettersen et al. [45]
2	Existing dataset of one match in the Nuremberg stadium in which all players got tracked	Tracking data	One 90-minute match measured at 200 Hz.	Muschler et al. [46]
3	Existing dataset of event data from the 2017/2018 season of the major European leagues	Event data	3,251,294 events	Pappalardo et al. [40]
4	Football events collection from season 2012/2013 to 2016/2017 of major European leagues	Event data	941,009 events	Alin Secareanu [1]
5	Semi-automatic player tracking and detection algorithm	Tracking data	Arbitrary (although video recordings and manual labelling are required)	Hurault et al. [43]
6	Automatic ball tracking algorithm	Tracking data	Arbitrary (although video recordings are required)	Najeeb et al. [44]
7	Automatic event detection algorithms for the classification of passes, shots, and goals	Event data	Arbitrary (although video recordings are required)	Imai et al., [37]; Morra et al. [38]; Khaustov and Mozgovoy [39]
8	A football simulator by Google Research Football	Tracking and event data	Arbitrary, but source code must first be altered for data transfer	Google Research Football [2]
9	eFootball (PES 2021), an advanced football simulator	Tracking data and limited event data	Arbitrary, but digital image processing algorithm must first be created for data extraction	Konami

Table 5-1: The list of candidate data souces.

simulator, are able to create highly realistic football environments. Simulation data was therefore thought to be representative for real football data and promising for further research.
Keeping in mind the dataset requirements and the advantages and disadvantages of the aforementioned approaches, a choice was made to only consider simulation data for this project. Both the GRF and the eFootball simulator were used for the extraction of data in this work.

5-1-3 Dataset creation

Two separate datasets were created, namely: the GRF dataset using the GRF simulator, and the eFootball dataset using the eFootball simulator. Both datasets will be made public and can be seen as contributions to the football research community. To the best of the authors' knowledge, these datasets will then be the largest collection of tracking data in football publicly available. Furthermore, the datasets do not only consist of tracking data, but also include detailed event data, thereby making the datasets even more interesting for further research. Both datasets will be briefly introduced and discussed in the following.

The GRF dataset

The GRF dataset was successfully created using the open-source GRF simulator. Extra code was written, and several adjustments were made in the source code with the idea to transfer and save the raw data to a separate data file. After a successful implementation, the GRF dataset was then created by simulating a total of 58 games. Although the simulator is rather simplistic and perhaps not as realistic as desired, it did provide a basic and controlled start of the project. Prediction results based on the GRF dataset will therefore be provided in section 7-2. However, this chapter will not further elaborate on how the GRF dataset was created, as the main focus of this work is based on the eFootball dataset. Nonetheless, if the reader is interested, an elaborate analysis of the creation of the GRF dataset can be found in Appendix B-1, which includes: a thorough analysis of the advantages and shortcomings of the GRF simulator in subsections B-1-1 and B-1-2, an explanation on how the dataset was created in subsection B-1-4, and the data validation process in subsection B-1-5.

The eFootball dataset

The eFootball dataset will be created using a state-of-art football simulator. Although the game is free-to-play, it is not open-source, and it is therefore not possible to directly obtain the data from the game itself. Consequently, a large part of the project has been dedicated to writing an algorithm which automatically creates a detailed dataset from video recordings. These video recordings can be obtained by recording the screen of a computer while playing the game. The recordings of simulated matches are then given as input to the algorithm which in turn gives as output the locations of all players and ball together with event data (i.e. passes, shots on target) and contextual match information (i.e. current minute, team formation). Section refdataset 2: eFootball simulator will describe in more detail how this dataset was obtained.

5-2 Creation of the eFootball Dataset

The eFootball dataset is derived from the game eFootball, previously known as Pro Evolution Soccer (PES). eFootball is a highly realistic game portraying real football matches. Players can play different game modes, including online and offline modes, where they can play against other players or the computer AI. The game is free-to-play but not open-source, thus access to the underlying data is restricted. Instead, the data was obtained by means of digital image processing techniques. A data extraction algorithm pipeline was designed solely for this project, able to automatically extract data from video recordings of the game.

A step-by-step analysis of the design of the algorithm will be given in the following. First, subsection 5-2-1 will give an impression of the gameplay and interface of the eFootball simulator to give a better understanding of how data can be extracted. Subsection 5-2-2 will then give a general overview of the entire algorithm, including a visual representation of the entire process. In the subsequent subsections of 5-2-3 all the way to 5-2-8, each segment of the algorithm will be explained in more detail. The running times of the algorithm are given in Table 5-5. Section 5-3 will then present both the final datasets.

5-2-1 The gameplay and user interface

A specific camera angle in the simulator allows the user to view the game from a top-view angle, in which all players have been simplified to circles and each player corresponds to a single circle on the pitch. Each circle is marked by the jersey number of the corresponding player and the color of the team's jersey. An example of the interface of the gameplay is depicted as a snapshot in Figure 5-1.

Figure 5-1: Snapshot of the interface of the gameplay of the eFootball simulator.

Besides the pitch and the players on the pitch, the user interface includes a couple of other interesting real-time features. Displayed at the top are the live score and the current minute of the game. Then there are different statistics displaced in the bottom left corner, namely: shots on goal, passes, touches, and ball winning. Furthermore, the player names and numbers of both teams can be viewed on each side of the pitch. All of the above were used for the creation of the final dataset.

A football game takes 90 minutes and is divided into two halves of 45 minutes. In the simulator, a 90-minute game lasts approximately 12 actual minutes, i.e., two halves of 6 minutes. The speed of the gameplay is the same as in a real football match. The time-ratio of the simulator compared to a real football match can thus be computed and is approximately 15:2, i.e., when one minute is played in an actual football match, seven and a half minutes have passed in the simulator. The game duration could not be adjusted in the simulator.

5-2-2 Algorithm pipeline overview

The complete data extraction algorithm consists of multiple segments. Each segment has its own function and uses information acquired from earlier segments. A schematic overview of the pipeline of the algorithm is given in the form of a block scheme in Figure 5-2. Each segment within the algorithm obtains its information by iterating over video images of the gameplay, extracting a relevant piece of data, and then saving the data into a file. The data gets added within these files with each iteration.

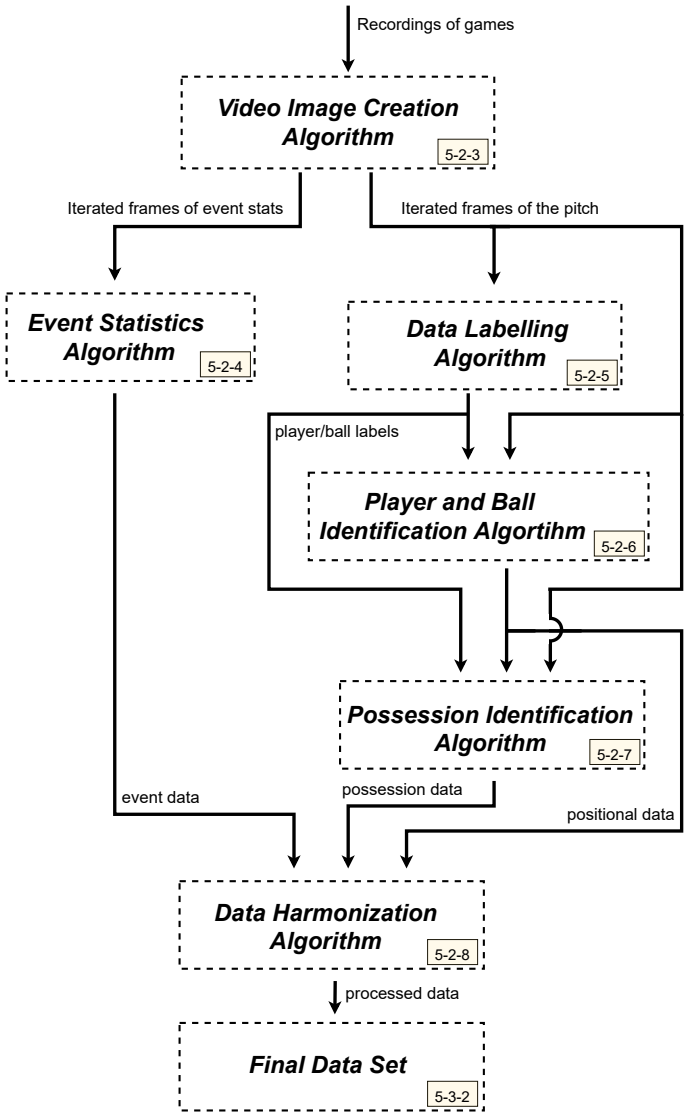


Figure 5-2: A general overview of the data extraction algorithm. The block scheme starts with video recordings of the simulator and ends with the final dataset.

The algorithm consists of six segments. The first segment is called the *'video image creation algorithm'* and converts video recordings into images, which are then used as input by all other segments. The second segment, *'the event statistics algorithm'*, extracts the match events occurring on the pitch (e.g., shot on target, goal) at each time instance. In parallel to the 'event statistics algorithm' run: the *'data labelling algorithm'*, which makes sure all players and ball for a single match are labelled correctly; *'the player and ball identification algorithm'*, with the task to identify and locate all players and ball in unseen image frames; and the *'possession identification algorithm'*, used to determine the possession team for each frame instance. Finally, the *'data harmonization algorithm'* removes errors from the data and fills in missing data through interpolation and extrapolation. The processed data then gets saved, which occurs in the last segment of the pipeline, namely the *'final dataset'*. Each segment will be explained in more detail in the subsequent subsections.

5-2-3 Video Image Creation Algorithm

The *video image creation algorithm* has the task of creating masked video images of the user interface and, subsequently, iterating over them. A series of steps were taken to achieve this task. A schematic overview of the process is given in Figure 5-3. First, screen recordings are made of individual games. A recording of a single game has a duration length of approximately 12 minutes. These video recordings are then converted to images. Two different masks were applied; one of the football pitch and the other one of the event statistics. The video-image-creation algorithm outputs both masks separately and iteratively, such that a focus can be put solely on the relevant parts of each image. The outputted images of the pitch are then given as a frame-by-frame input to three other segments, namely: the 'data labelling algorithm', the 'player-and-ball identification algorithm', and the 'possession identification algorithm', whereas the outputted images of the event statistics are iteratively given to the 'event statistics algorithm'. A graphical representation of this process can be observed in Figure 5-3.

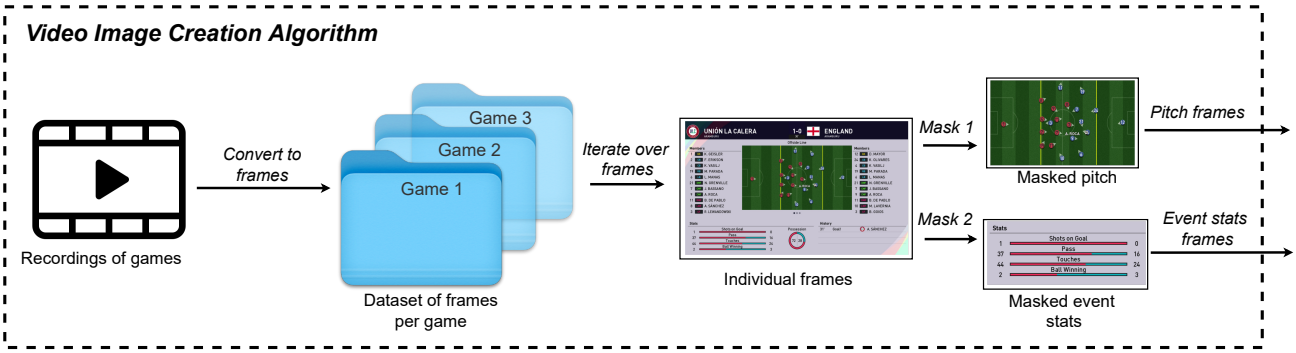


Figure 5-3: Block scheme of algorithm used to create masked images from game recordings.

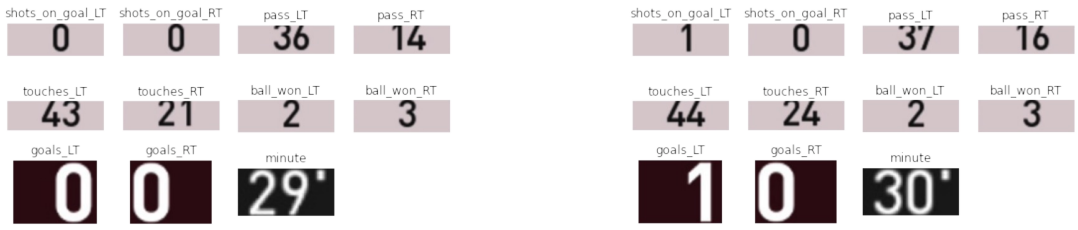
5-2-4 Event Statistics Algorithm

After establishing the desired framework for getting the individual event statistics frames, a dataset consisting of all event data can be created using the *event statistics algorithm*. The input of the algorithm are images of event statistics of individual games from the video-image-creation algorithm, which are given to the algorithm in frame-by-frame manner. The output of the algorithm is the event dataset of a given game.

A series of steps is taken to extract the relevant knowledge from the input image. First, a mask is created such that each individual event statistic is depicted in a single image. A total of 11 event statistics are masked and tracked during the game, namely: shots on goal (2x), passes (2x), touches (2x), ball won (2x), goals (2x), and the time in minutes. Note that most statistics are being counted twice due to both teams having their own counters for each statistic. An example of these masks is illustrated in Figure 5-4a.

Furthermore, the algorithm keeps track of the occurrences of events by linking a counter to each statistic. At the beginning of each match, all counters are initialized to zero. Also, after each iteration, the masked statistics are cached for comparison with the statistics of the next iteration. The counting process starts whenever two consecutive masked event images can be compared with one another, as illustrated in Figure 5-4. When two consecutive images differ, a corresponding

counter is incremented by a value of one. By iterating over all images and comparing all consecutive images of the masked statistics with one another, an accurate dataset can be created. A schematic overview of this process is illustrated in Figure 5-5.



(a) Event statistics at time t **(b)** Event statistics at time $t+k$

Figure 5-4: The masked numbers of all event statistics of two consecutive frames are compared with each other. Counters are updated based on differences in individual statistics.

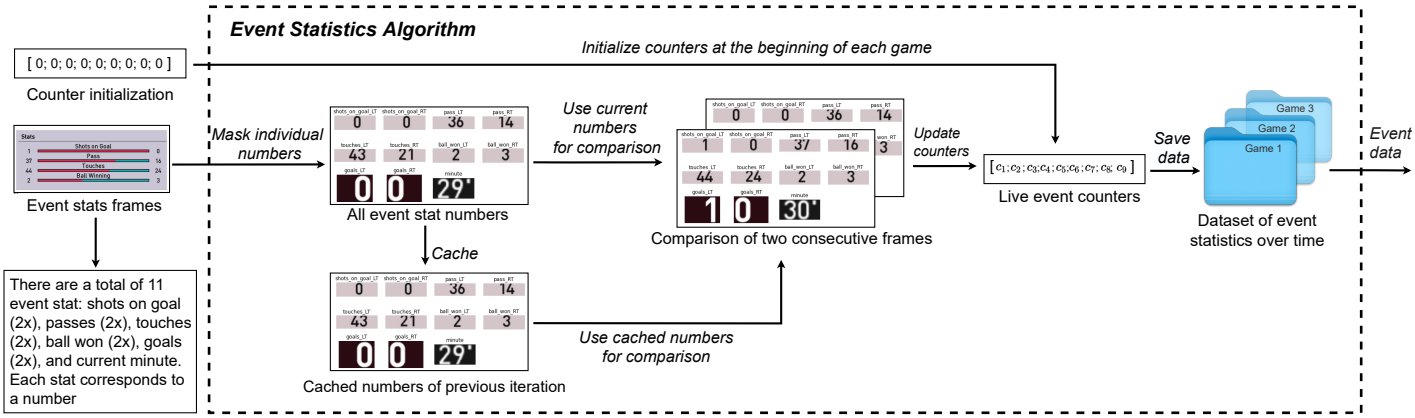


Figure 5-5: Block scheme of algorithm used to determine the real-time statistics of events during a match. Examples of event statistics are: goals, shots on target, touches. The input of the algorithm is a single image frame of the statistics, while the output is an update of the counters.

5-2-5 Data Labelling Algorithm

A semi-automatic *data labelling algorithm* will be designed to identify and label the templates of all players and ball. In this work, a template is defined as a square image which represents either a distinct player on the pitch or the ball in a given game. The algorithm takes as input the image frames of the pitch from the video-image-creation algorithm and outputs the labelled templates of all players and ball. Identification and labelling of the players is done once at the beginning of each game. This is required because the templates of the player change each game due to varying jersey colors and numbers. The template of the ball, however, is the same throughout all matches and was therefore only identified and labelled once. Once the labelled templates of a single game are collected, an automatic tracking algorithms will detect the players and ball for the remainder of the game.

The algorithm can be subdivided into two phases. The first phase consists of the identification of the circles of the players. The second phase deals with the labelling process of the players. Both

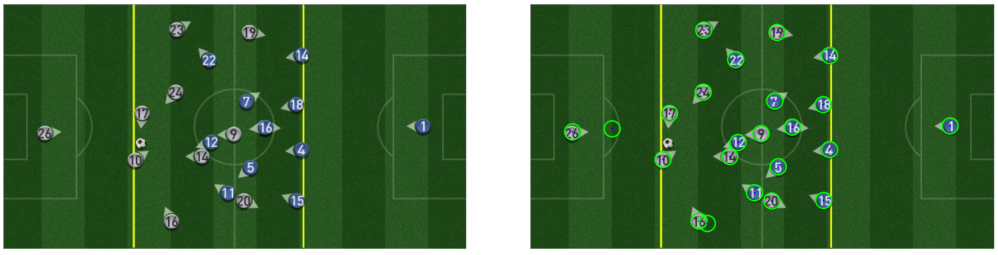
phases will be explained briefly. An overview of the entire data labelling algorithm can already be observed in Figure 5-8, which schematically describes all the taken steps.

First, the circles representing the players will be identified automatically using the OpenCV Hough Circle Transform (HCT) algorithm. The theoretical concept of the HCT algorithm, as well its parameters, are explained in the preliminaries of subsection 3-3-1. The algorithm takes as input the masked image of the pitch from the video-image-creation algorithm, and outputs the (x,y)-coordinate locations of all detected circles within the image. The specific input image for each game is chosen manually and should give a clear view of each individual circle without overlap. The parameters of the algorithm were tuned and optimized with the purpose of identifying all relevant circles in a given image frame of the pitch. The tuned parameter values are displayed in Table 5-2.

Method	minDist	param1	param2	minRadius	maxRadius
Hough gradient	25	100	9	25	27

Table 5-2: Optimized parameter values for the OpenCV HCT algorithm.

The minimum distance 'minDist' was set such that the same circles are not identified twice. The parameters 'param1' and 'param2', representing the identification thresholds, were set relatively high such that all circles in a frame could be detected. This comes at the cost of an increased detection of false positives. The minimum and maximum radii 'minRadius' and 'maxRadius' were set such that only the circles of the players are approved. Conveniently, all player circles have a size of approximately 26 pixels. An indication of the performance of the HCT algorithm can be seen in Figure 5-6, which shows how the player circles are detected in a given image.



(a) Input image **(b)** Output image

Figure 5-6: Example of how the HCT algorithm is able to detect the relevant circles from an input image of the pitch as shown in **(a)** to an output image in **(b)**. The (x,y)-locations of the centres of the detected circle are saved and given to the labelling algorithm.

Now consider the labelling phase. This section of the algorithm takes as input: an image of the pitch, and the (x,y)-locations of the detected player circles. It then outputs the labelled player-and-ball templates of a single game. Labelling all players accurately is extremely important, as these labels will be used for detection and tracking of all players for the entirety of a match. A single labelling error at this stage can therefore lead to wrongful detection of a player for a whole game. It was therefore chosen to manually label each player. Additionally, the template labels were are visually double-checked to ensure a 100% labelling accuracy.
In this part of the algorithm, the templates were visually shown in the algorithm interface such

that they could be manually labeled, as illustrated in Figure 5-7. Each circle was given a label which includes: a player ID, a team ID, and a player position. The set of 23 labeled circles (22 players and the ball) was saved. The labelled templates were then used as inputs to: the 'player and ball identification algorithm'; and the 'possession identification algorithm'. The whole process is visualized in a block-scheme in Figure 5-8.

(a) U-20 **(b)** O-5 **(c)** Ball

Figure 5-7: Examples of templates of players of opposing teams and the ball. Labels are given to each template, where the labels U-20 in **(a)** and O-5 in **(b)** refer to 'player #20 of the user team' and 'player #5 of the opposing team' respectively. The label in **(c)** refers to the ball.

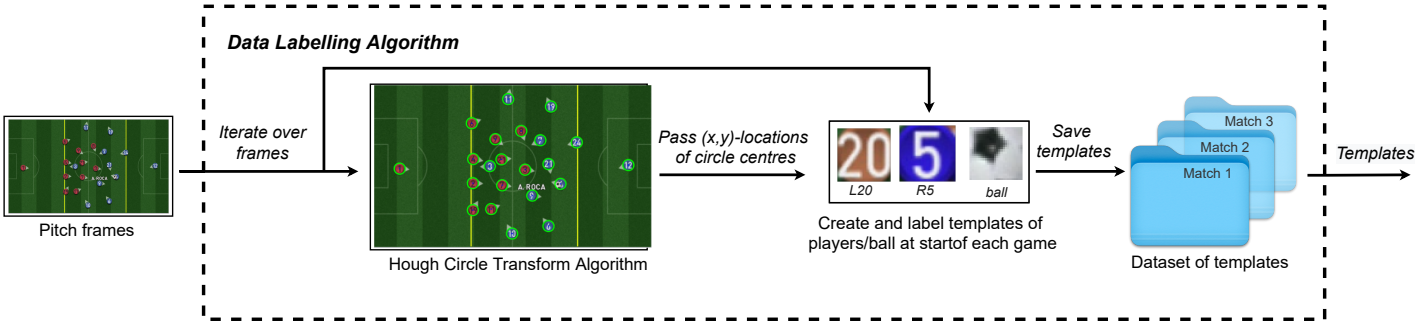


Figure 5-8: Overview of the labelling process of players and ball.

5-2-6 Player and Ball Identification Algorithm

During the course of the match, the unique circular templates of players and ball do not change appearance. As a result, the labelled templates allow detection possibilities of players and ball in unseen frames. The *player and ball identification algorithm* takes as input: the set of templates of a single game from the data-labelling algorithm, and the masked images of the pitches from the video-image-creation algorithm. The automatic detection of players and ball is done using a technique called template matching, as explained in subsection 3-3-2, which aims to locate the (x,y)-coordinates of a given template in an unseen frame. The output of the algorithm is a vector denoting the (x,y)-coordinates of all players and ball in a given image. A coordinate vector of all players and ball can then be made by iterating over all templates. A schematic overview of this process can already be observed in Figure 5-12. Each section of the algorithm will be elaborated on in subsequent parts.

Template matching

First, two template matching functions were constructed for: the detection of the ball, and the detection of players. The functions take as input: the template image T (i.e., a player or ball), and the

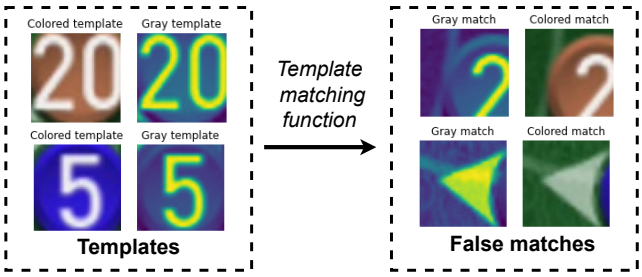
search image S (i.e., a masked image of the pitch). The output of the functions are a set of candidate matches of T in S, where values for a correlation metric reach above a certain threshold. The function parameters, as explained in detail in the preliminaries of subsection 3-3-2, are displayed in Table 5-3 and will be discussed in the following.

	Template image T	**Search image** S	**Method for acquiring** R	**Match threshold** $R(x,y)$
1	Ball template	Image of pitch	Normalized correlation coefficient	0.85
2	Player template	Image of pitch	Normalized correlation coefficient	0.91

Table 5-3: Input parameters of the template matching algorithm.

Two separate functions for the detection of players and ball were designed. The first function focuses on the detection of the ball template T_{ball} within search image S, while the second function deals with the detection of all player templates T_{player} within search image S. The normalized correlation coefficient was chosen as the correlation metric $R(x,y)$ for both functions as this is the most commonly used correlation metric for template matching. The values of $R(x,y)$ are collected in result matrix R, where high values indicate a higher probability of an actual match. Finally, thresholds were set to reduce the probability of false positives. Such thresholds are especially important whenever players in S move close to one another, as illustrated in Figure 5-9a. As a result of player junctions, the template images can often not be found in the search image. Without proper thresholds, the template matching algorithm will still detect false positives, as depicted in Figure 5-9b. If the reader is interested, the process of selecting suitable thresholds is explained in more detail in Appendix B-2-1.

(a) Junction of players in S



(b) Examples of false positives

Figure 5-9: **(a)** shows players moving over each other in the search image, thereby making it impossible to detect certain templates T. Consequently, in the absence of the actual T in S, false positives might be detected. Examples of false positives are given in **(b)**, where the images in the columns represent (from left to right): colored T, gray-scale T, a gray-scale match in S, the gray-scale match converted back to a colored image.

Verification of candidate matches

The template matching functions require the three-layered, colored input image, to be converted to a one-layered, gray-scale image, as this reduces the computational intensity. However, this

comes at the cost of an increased probability of detecting false matches, especially when it involves color. In the case of the templates of players, which are used as input to functions, this often results in such false matches, as is illustrated in Figure 5-10.

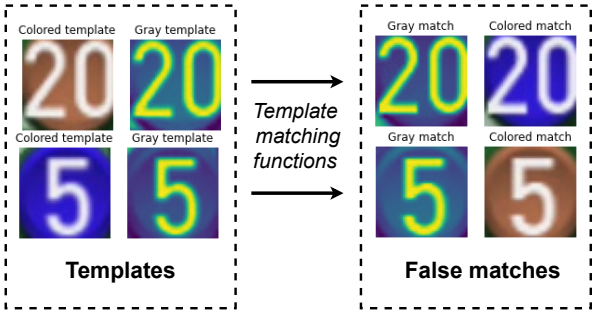


Figure 5-10: False matches of player templates due to the inability of the template matching functions to distinguish various colors. The columns in the image represent (from left to right): colored T, gray-scale T, a gray-scale match in S, a colored match in S.

A solution was found through an extra verification process, in which the colors of the candidate match are compared to the colors of the actual jersey. This process is schematically visualized in Figure 5-11 and will be briefly elaborated upon in the following. Before an actual match of T in S is found, a set of candidate matches are provided by the functions. The functions provide a list of candidates matches by sliding a template T over the search image S and collecting the (x,y)-locations where $R(x,y)$ reaches above the set threshold. The candidate matches are then sorted based on their correlation value $R(x,y)$. Subsequently, a verification process takes place where each one of the candidate matches gets verified based on their color, starting with the candidate match with the highest value for $R(x,y)$. Then, when a candidate matches passes the color verification test, it is selected as the best match and the detection of that template stops. In the case no candidate match passes the color verification test, it is likely the template of that player is hidden behind another one, and a NaN-value is passed for its location.

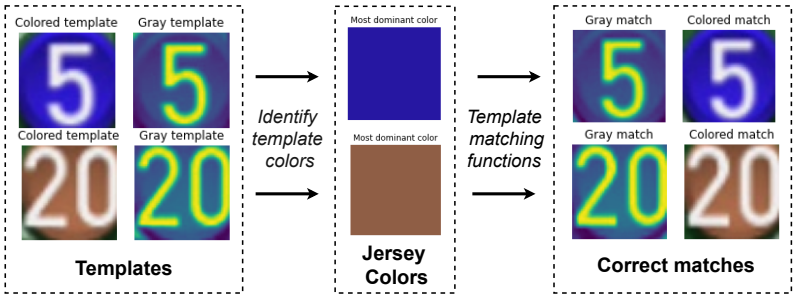


Figure 5-11: Visualization of two examples passing through the template matching algorithm after candidate matches have been verified based on their color.

This detection process is repeated for each template within each frame of all games. The (x,y)-coordinates of the centre of each template T in S are saved in a separate file. Finally, by iterating over all games, the tracking data of all players and ball can be collected. An overview of the algorithm code is given in Figure 5-13, which summarizes the whole player-and-ball detection process as described in earlier parts. The whole process is also schematically visualized in a block-scheme in Figure 5-12.

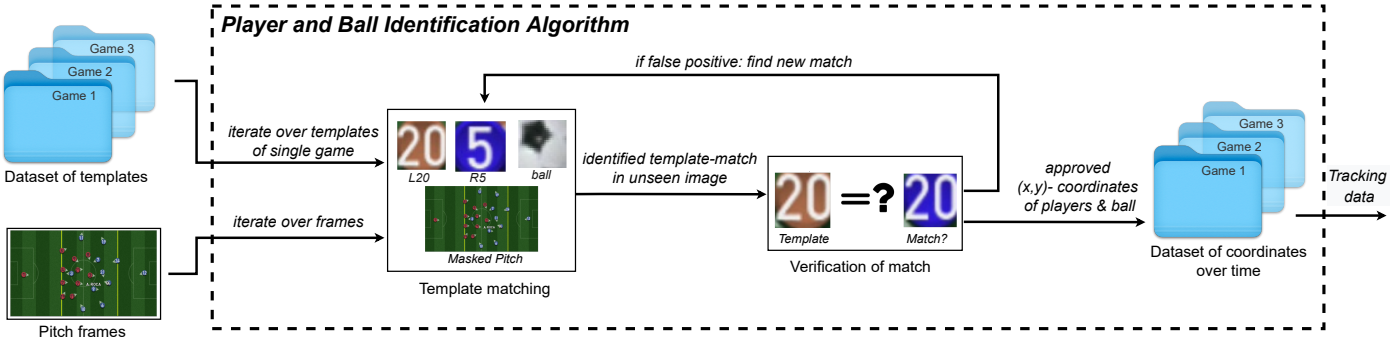


Figure 5-12: The process of identifying the players and ball on unseen images using a template matching algorithm, depicted as a block scheme.

Algorithm code overview

```
1: Import dataset
2: For i = 1,.....,number of games:
    3: Import templates of players and ball of game
    4: For j = 1,.....,number of frames in game:
        5: For k = 1,.....,number of templates:
            6: Obtain list of candidate matches using template matching functions
            7: Sort list of candidate matches based on their R(x,y) value,
               i.e., their probability of being a match
            8: For m = 1,....., number of candidate matches:
                9: Return (x,y)-location of candidate match of the first
                   candidate match passing the color verification test
```

Figure 5-13: Overview of code of the player-and-ball detection algorithm.

5-2-7 Possession Identification Algorithm

The aim of the *possession identification algorithm* is to identify which team has possession of the ball in a given frame. The algorithm takes as input: an image of the pitch, a labelled player template of either team, and the (x,y)-coordinate of the ball. The output is the team in possession. The algorithm is constructed based on the fact that the ball always moves on top of all other players, and is therefore always completely visible. Now, consider a scenario where a player is in possession of the ball. In this case, the ball always sits on top of the player. Then, by analysing the colors of the area just around the ball and, subsequently, linking these colors to the jersey colors of one of the teams, it becomes possible to deduce the team in possession.

The algorithm consists of three different parts. The first part takes as input: an image of the pitch, and the (x,y)-coordinate of the ball in that image. It then zooms in on the ball and creates a template of the ball in that image. The ball template is then used to create a 'possession template', by cutting out the area around the ball and removing all other pixels from the image. The possession template is then used as output. This process can be observed in Figure 5-14.

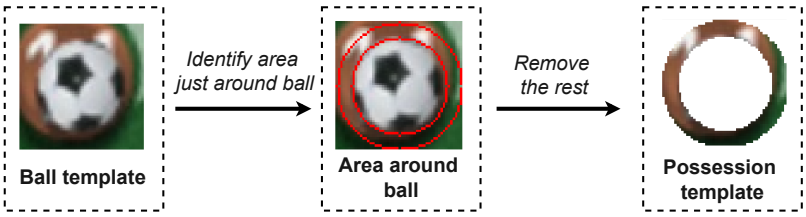


Figure 5-14: Process of creating a possession template from the ball template.

The second part of the algorithm runs in parallel with the first part. It takes as input a single template of both teams and outputs a list of dominant colors of both templates. This process is illustrated in Figure 5-15.

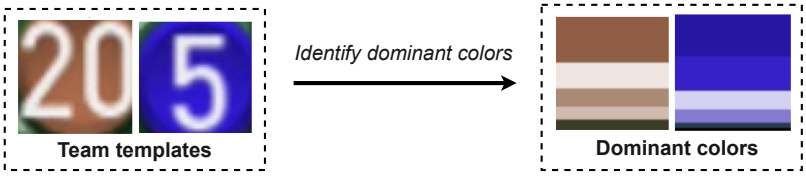


Figure 5-15: The process of extracting the dominant colors from the team templates.

Finally, the third part of the algorithm combines the earlier parts to deduce which team is in possession for a given frame. The possession template is given as input. The dominant colors of the possession template are then identified and compared to: the dominant colors of the jersey of the user team, the dominant colors of the jersey of the opposing team, and the color of the grass. In this comparison process, the most dominant color of the possession template is compared first to the colors of the team jerseys. Then the second most dominant color is compared, then the third, etc. When a match is found, the algorithm terminates and outputs the corresponding team. The algorithm has four possible outputs for the possession team, which are as follows: 'the user team', 'the opposing team', 'the grass', and 'NaN'. A NaN-value is only given whenever no match was found. An overview of this process is illustrated in Figure 5-16.

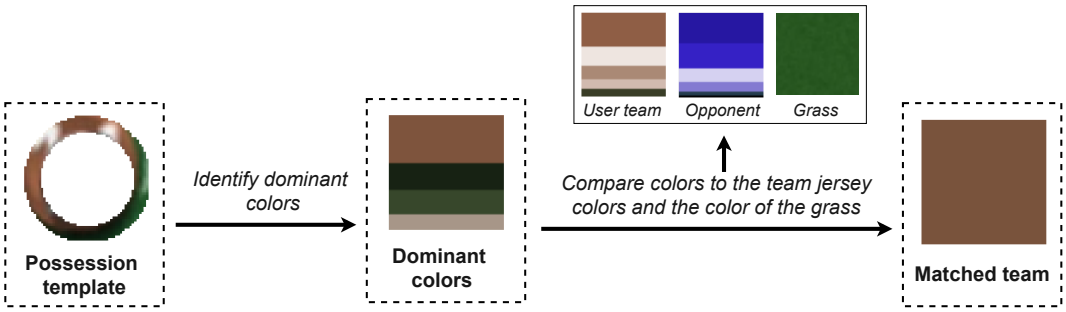


Figure 5-16: The process of finding the team in possession based on the possession template and the dominant colors of both team jerseys.

This process is repeated for all frames. The data is saved in a separate file. A schematic overview of the whole algorithm is depicted in Figure 5-17.

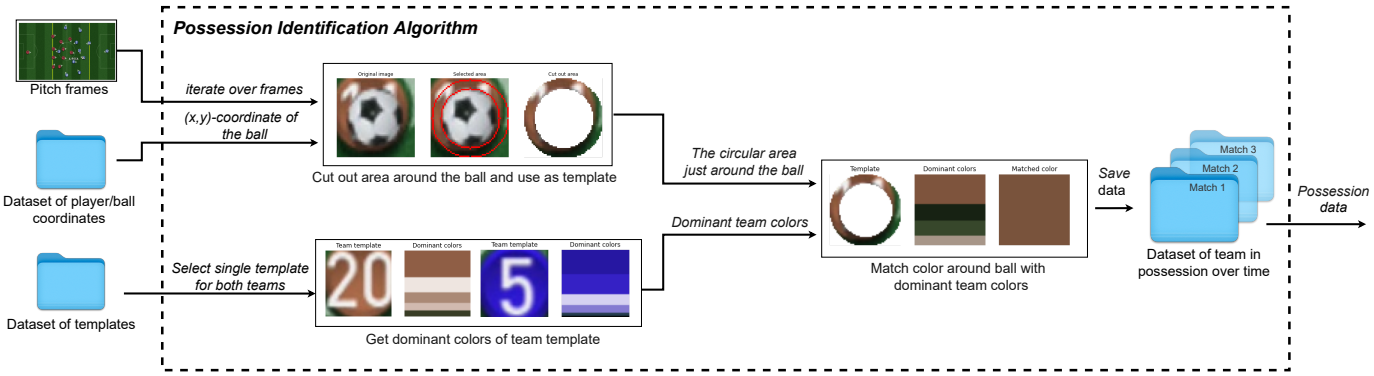


Figure 5-17: Schematic overview of the algorithm used to deduce the team in possession of the ball. The algorithm takes as input an unseen masked image of the football pitch, a labelled template of both opposing teams, and the (x,y)-coordinate of the ball.

5-2-8 Data Harmonization Algorithm

The event data, the possession data, and the tracking data all get merged into a large dataset in the *data harmonization algorithm*. The function of this algorithm is to merge all data types, remove errors, remove non-relevant information, validate the data and add missing values. The non-relevant information refers to the parts of the dataset corresponding to non-relevant frames, such as the frames: before the match started, from the menu during half-time, and after the match. Furthermore, missing data is approximated by interpolating between known points in time. Extrapolation is applied whenever data was missing from the start or at the end of a half. All will be discussed in the following parts.

Data validation

A subdivision can be made in the validation of: event data, possession data, and tracking data. The validation of event data is straightforward and can be done by comparing the extracted values with

the actual value. To be more precise, the extracted event vector $[e_1, e_2, \ldots, e_{11}]$ at any time t can be compared to the actual values of those events $[e'_1, e'_2, \ldots, e'_{11}]$ from the corresponding image frame at time t. When the event vectors are the same, the data is fully accurate. The actual accuracy of the event data will be given and discussed further section 7-1.
The validation of the possession data and the tracking data is done by checking how often 'the player-and-ball identification algorithm' and 'the possession identification algorithm' were able to correctly match a template T in search image S. In other words, it checks how many missing values (i.e., NaN-values) are present in the data. To prevent false matches, all detection thresholds were chosen very conservative such that interpolation of missing data will not lead to the identification of completely wrong trajectories. The amount of missing data is given as percentages of the total amount of cases per match and shown in Table 5-4. These percentages therefore don't directly quantify detection accuracy, although they do give a good indication of what the lower bound of the detection accuracy will be.

Game no.	Players location	Ball location	Possession team
1	82.3 %	80.9 %	82.6 %
2	85.6 %	87.0 %	84.2 %
4	88.1 %	88.1 %	84.7 %
5	84.4 %	87.4 %	80.2 %
6	83.4 %	85.7 %	84.9 %
7	75.2 %	90.4 %	89.6 %
8	84.5 %	89.7 %	82.7 %
9	75.1 %	90.1 %	87.8 %
10	75.6 %	90.9 %	88.8 %
11	79.4 %	87.4 %	84.6 %
14	81.0 %	92.3 %	90.8 %
19	78.2 %	86.5 %	86.5 %
20	79.2 %	90.0 %	90.0 %
22	75.8 %	87.7 %	87.7 %
23	76.5 %	86.0 %	86.0 %
24	75.1 %	91.4 %	91.4 %
total=16	80.0 %	88.2 %	86.4 %

Table 5-4: Number of times the locations of players and ball as well as the possession team were identified by the detection algorithms prior to interpolation and extrapolation. The values are given as a percentage of all cases per match.

A total of 25 games were simulated. However, not all detection percentages were as desired. For example, in some cases, the detection algorithms were only able to detect the players approximately 60% of the time. The reason for such poor performance follows from the fact that it is not possible to select the jersey of the opposing team. The games are simulated online, and eFootball only allows you to select the jersey of your own team. Consequently, the colors of the jersey of the opposing team may be similar to the user team's jersey or similar to the color of the grass. This lead to detection problems for the detection algorithms in some games. As a result, it was chosen to discard 9 games where detection percentages reached below values of 75%. As can be observed from Table 5-4, a total of 16 games remain for further research. The detection percentages of the discarded games have been left out, as they will not be further used in this work. Nonetheless, the detection percentages of all simulated games can still be observed in Appendix B-2-2.

Interpolation and extrapolation

Missing values within the tracking data are filled in using interpolation and extrapolation methods. Both will be discussed in the following. Interpolation of missing (x,y)-coordinate data is done through a combination of cubic spline interpolation and linear interpolation between the bounds of a set of known points. Cubic spline interpolation is a method where splines $f : [x_t, x_{n+t}] \rightarrow \mathbb{R}$ over time interval $[t, t+n]$ are constructed of third-degree piece-wise polynomials [79]. Cubic spline interpolation takes into account the first and second derivative in the points where adjacent polynomials touch. In terms of the movements of players, this means that velocity and acceleration are included into the equation. It therefore makes sense to use cubic spline interpolation, as this yields the desired smoothness of the data in a relatively simple manner while also avoiding undesired oscillations in the interpolated data, which may occur when even higher-order polynomials are chosen. However, whenever the (x,y)-coordinate locations of players/ball are unknown for an extended period of time, cubic spline interpolation may cause some problems. An example of such an interpolation problem occurs when a player is last observed accelerating towards the edge of the pitch, then goes missing for an extended period of time, and later re-emerges at the other side of the pitch. The errors that follow include: location values far outside the pitch; velocities higher than humanly possible; and unrealistic, cumbersome player trajectories. Therefore, it was chosen to combine cubic spline interpolation with linear interpolation. Cubic spline interpolation was applied for cases when players go missing for a brief period of time. Furthermore, interpolated values were discarded whenever they reached unrealistic velocities or accelerations or when locations were observed outside the boundaries of the pitch. Linear interpolation was applied for all remaining missing values, where all linearly interpolated points were equally spaced in time.
The remaining missing values at both ends of the data, so at the start or end of a half, are obtained through extrapolation by assuming the player or ball stays positioned at their last known location. Similarly, missing values in the possession data were interpolated and extrapolated by assuming the last known team in possession stays in possession for all missing values in a game.

Merging the data

The eFootball dataset can now be created by merging the updated event data, possession data, and tracking data into a single dataset, consisting of 16 games. The data properties of the eFootball dataset will be given and elaborated on in subsection 5-3-2. The actual performance of the entire data extraction algorithm pipeline will be evaluated in section 7-1.

5-2-9 Running times

Running times of the algorithms could be adjusted by changing the input frequency of the image frames. Different input frequencies were tried and tested for different segments of the algorithm. The input frequency of the frames for the 'the player and ball identification algorithm' and 'possession identification algorithm' was set at approximately 4 frames/second, which includes enough data to accurately describe the trajectories, while at the same time keeping the total computational time relatively low. The image input frequency of the 'event statistics algorithm' was set at a value of 8 frames/second. This value is set higher because the event statistics algorithm needs to iterate over almost all frames in order to detect changes in subsequent frames. Furthermore, the

running time of the event statistics algorithm is not very high and could therefore handle a higher frame input frequency. Both parts of the algorithm were run in parallel to speed up the process. The running times of all individual algorithms per match, as well as the running time of the entire algorithm, can be observed in Table 5-5.

Algorithm name	**Avg. manual input per game in [min]**	**Avg. algorithm running time per game in [min]**	**Avg. total running time per game in [min]**
Video Image Creation	19	20	39
Event Statistics	-	9	9
Data Labelling	17	0	17
Player and Ball Identification	-	66	66
Possession Identification	-	5	5
Data Harmonization	3	0	0
Complete Algorithm	39	101	140

Table 5-5: Running times of all segments of the algorithm per game, classified in manual input times and running times of the algorithm itself. The running times of each segment of the algorithm were rounded to the nearest minute.

The running times are only an indication of how long it takes for a new game of data to be generated. It takes an estimated total of 140 minutes to create a dataset of a single match, of which 39 minutes have to be done manually, and 101 minutes are done automatically by the algorithm.

5-3 Final Datasets

This section will give a description of the contents of both datasets. Subsection 5-3-1 will describe the GRF dataset and subsection 5-3-2 will elaborate on the eFootball dataset.

5-3-1 GRF dataset

The GRF dataset covers the state of 58 games over time. The state includes, but is not limited to: the exact (x,y)-coordinates of the players of both teams, the [x,y,z]-coordinates of the ball, a tiredness factor of the players of both teams, the team in possession of the ball (i.e., no team, user team, opponent), the current game mode (i.e., normal play, kick-off, goal-kick, free-kick, corner, throw-in, penalty), the score of the match, the time, the players positions, and the team's formations. If interested, a classification of the entire state can be observed in Table B-1 in Appendix B-1-3. Various relevant data features were extracted and are listed Table 5-6.

Data feature	User team	Opponent	Total
Total number of states/ time-steps	-	-	168,056
Total number of games played	56	56	56
Total number of games won	26	10	36
Total number of draws	20	20	20
Total number of goals	46	18	64
Average amount of goals per game	0.82	0.32	1.14
Average percentage of ball possession per game	56 %	44 %	-

Table 5-6: GRF dataset features.

5-3-2 eFootball dataset

The eFootball dataset consists of 16 games of data, where each game of data lasts 90 in-game minutes, which converts to 12 actual minutes. Each game of data consists of a series of states collected over time. The state is sampled at around 4 frames per second or, when converted to within-game seconds, approximately 0.5 frames per in-game seconds. The state consists of: the extracted (x,y)-coordinates of the ball and the players of both teams, the team in possession, the score, a time label; the touches of both teams, the balls won of both teams, the passes given by both teams, the shots on target by both teams, the player positions of both teams, and the team's formations. Some relevant data features of the eFootball dataset are displayed in Table 5-7.

Data attribute	User team	Opponent	Total
Total number of states/time instances	-	-	53,022
Total number of games played	16	16	16
Total number of games won	6	4	10
Total number of draws	6	6	6
Average amount of goals per game	1.2	0.9	64
Average amount of shots on target per match	4.4	2.8	7.2
Average amount of passes per match	105	97	202
Average amount of touches per match	152	138	290
Average amount of balls won per match	15	13	28
Average percentage of ball possession per game	54 %	46 %	-

Table 5-7: eFootball dataset features.

Chapter 6

Design of Prediction Model

This chapter will describe the design and discuss the design choices of the prediction model based on the model proposal as presented in subsection 4-4-2. Section 6-1 will first give an overview of the entire algorithm used within the prediction model. Section 6-2 will elaborate on the training and hyperparameter tuning process and will discuss various network design choices. The final prediction models will then be selected and validated in section 6-3. Finally, the conclusions of the chapter are summarized in section 6-4.

6-1 Algorithm Pipeline Overview

First, an overview of the entire prediction algorithm pipeline will be given. The same pipeline was used for both the Google Research Football (GRF) dataset as the eFootball dataset. The algorithm is split up into multiple sections. Each section has its own purpose and interacts with other parts of the algorithm, as visualized in Figure 6-1. A brief introduction to each section will be given in the following.

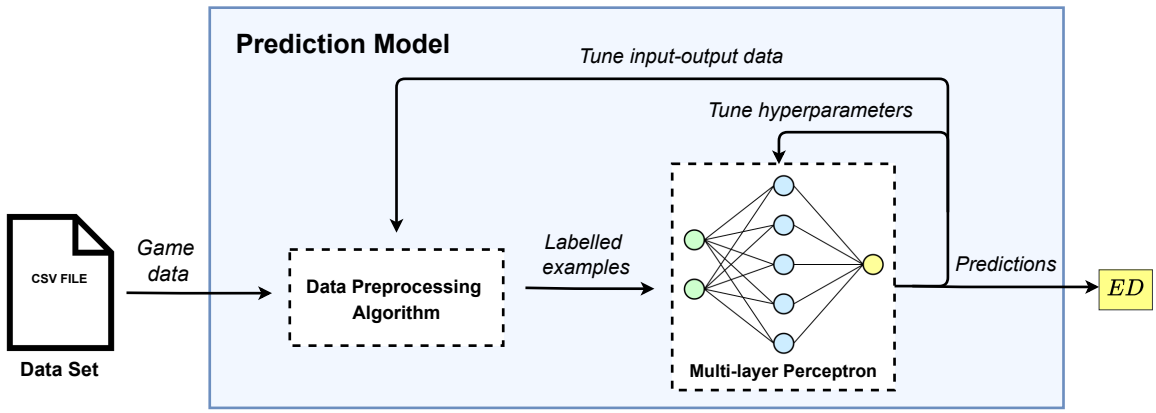


Figure 6-1: Overview of the design of the Multilayer Perceptron (MLP) prediction algorithm.

During the preprocessing of the data, the game data is converted to labelled input-output examples. These labelled examples are then fed into the MLP during the training process. Weight and bias parameters are adjusted each iteration until stopping conditions are reached, and the model is fitted on the dataset. Different models with different hyperparameters are trained, and prediction results are analyzed. The model with the best performance on the validation dataset is then chosen as the final prediction model. All parts will be further elaborated on in the subsequent parts of this chapter.

6-2 Training and Tuning

This section will discuss the design choices that were made in the creation process of the prediction model. The mathematical notations of the used parameters and symbols of the prediction model are presented first in subsection 6-2-1. Subsection 6-2-2 will then discuss the general design choices of the prediction model. Subsection 6-2-3 elaborates on the preprocessing procedure of the data. Subsections 6-2-4 and 6-2-5 will then explain how the models were trained and which hyperparameters were used.

6-2-1 Notation

A list of the notations for parameters of the MLP in this work are given in the following, based on the theory as explained in subsection 3-2-2. m_s is the number of the examples in the dataset, n_x is both the input size and the size of the state vector, n_y is the output size, n_l are the number of hidden layers, $n^{[l]}$ is the number of neurons of the l^{th} layer. $X \in \mathbb{R}^{n_x \times m}$ is the input matrix of the neural network, $x^{(i)} \in \mathbb{R}^{n_x}$ is the i^{th} example represented as a column vector, $Y \in \mathbb{R}^{n_y \times m}$ is the label matrix or output matrix, $y^{(i)} \in \mathbb{R}$ is the output label for the i^{th} example. The weight matrix $W^{[i]} \in \mathbb{R}^{n_x \times n^{[l-1]}}$ is the collection of all the weights in the l^{th} layer, given the network is fully connected. $w_i^{[l](k)}$, $a_i^{[l](k)}$, and $b_i^{[l](k)}$ are the weight vector, activation output, and bias respectively of the i^{th} neuron in the l^{th} layer of the k^{th} example. All the parameters of the model (of all layers) are then collected as the weights $W = \{W^{[0]}, W^{[1]}, \ldots, W^{n_l}\}$ and the biases $b = \{b^{[0]}, b^{[1]}, \ldots, b^{n_l}\}$.

6-2-2 Building the model

No publicly available, pre-trained Artificial Neural Network (ANN) was found for making within-game predictions using tracking data in football (or any other sport). Therefore, it was chosen to build a MLP from scratch, which involves trying to optimize the weights W and biases b during the training phase, as well as tuning the hyperparameters of the model. The parameters in W and b were trained according to the training methods described in subsection 3-2-2. The input and output layer of the network follow from the model as proposed in section 4-4. To summarize, the model proposes that the input layer consists of n_x neurons, where the input to each neuron is equal to a single state variable of $\tilde{s} \in \tilde{S}$. The number of hidden layers n_l is given as a hyperparameter to the model. Finally, the output layer consists of a single neuron and will be trained using the average reward over time interval $[t+1, t+h]$, denoted as G_t^h. For the design of the network, the Keras [1]

[1] https://keras.io/api/

open-source software library was used within the Python coding environment, which provides built-in tools for the design and training all kinds of Machine Learnings (MLs) models.

6-2-3 Division of labelled examples

The labelled input-output examples were created from the datasets, following the data preprocessing procedure as described in subsection 4-4-2. The new datasets of labelled examples were randomly shuffled and split into three smaller sets before the training phase was initiated, namely: the training set, the validation set, and the test set. Each set has its own purpose, but they all contribute to generating an unbiased prediction model. The function of the training set is to train the weights and biases. The function of the validation set is to evaluate a trained model and update its hyperparameters. Finally, the test set is meant for unbiased evaluation of the performance of the final model, which were already trained on both the training set and the validation set. The validation and test sets needs to be big enough for unbiased evaluation, but at the same time not take too many examples away from the training set. For smaller data sets of about 1,000-100,000, a rule of thumb is to divide the train/val/test as 60/20/20, or only even exclude a separate test set and divide following a 70/30 division. For larger data sets >1,000,000, a rule of thumb is to take a 98/1/1 division [60]. Based on the rules of thumb and the sizes of both datasets, as presented in Table 5-6 and Table 5-7, a decision was made to split following an 80/10/10 division. Table 6-1 shows the exact amount labelled examples for each part of the training-, tuning-, and testing process.

	Training examples	**Validation examples**	**Test examples**	**Total examples**
GRF dataset	123,166	15,396	15,396	153,958
eFootball dataset	34,003	4,250	4,250	42,503

Table 6-1: Division of labelled examples for both datasets.

6-2-4 Training the predictor functions

A total of 10 final MLP's were trained and tuned for all horizons and the two different datasets. The hyperparameters of the models for both datasets were only tuned for horizon $h = 15$. Transfer learning (section 3-2-2), a technique where hyperparameters of one ANN can be transferred to a similar ANN, was applied to train the MLP's for all other horizons $h = \{5, 10, 30, 45\}$. This technique could be applied as the input-output examples: have identical structure, are from the same dataset, and serve the same objective. Transfer learning was applied as it saves time and resources. The characteristics of how all finals models were created can be observed in Table 6-2. In the case of the MLP's with $h = 15$ (models 3 and 8), various hyperparameters were tested on a total of 100 and 120 different models for the GRF and eFootball datasets respectively and results were analyzed. Ultimately, the model achieving the best performance was chosen as the final model. The average/approximated running times of the individual model algorithms can also be observed in the final column.

MLP number	dataset	Prediction horizon h [mins]	Transfer learning	Models trained	Running times [mins]
1	GRF	5	yes	1	5
2	GRF	10	yes	1	5
3	GRF	15	no	100	500
4	GRF	30	yes	1	5
5	GRF	45	yes	1	5
6	eFootball	5	yes	1	2
7	eFootball	10	yes	1	2
8	eFootball	15	no	120	140
9	eFootball	30	yes	1	2
10	eFootball	45	yes	1	2

Table 6-2: Creation characteristics and of all final models.

6-2-5 Hyperparameter tuning and network design choices

A common practice in hyperparameter tuning is choosing random values from a predefined set of values. Choosing hyperparemeters using a fixed step size is generally avoided, as more information about each hyperparameter is gathered when different values are chosen for all hyperparameters in each attempt. When having n different hyperparameters, each attempt will sample its hyperparameters from a n-dimensional space. When aiming for faster convergence, a course-to-fine sampling scheme can be chosen. Coarse-to-fine sampling works by zooming in on a subset of the n-dimensional hyperparameter space where results are most promising during the training process and, subsequently, sample more densely within this new subset space. Following this approach, a focus can be put on values of hyperparameters that a have higher probability of being optimal. In this work, course-to-fine sampling was performed solely on the learning rate α as this is often regarded as the most important hyperparameter to tune [60].

Before the random sampling of the hyperparameters can take place, an appropriate scale needs to be chosen for each hyperparameter. For some hyperparameters, it is perfectly reasonable to pick values uniformly at random from a certain set of values. For example, when picking the amount of hidden neurons in layer l, one can choose a value in the uniform set: $n^{[l]} = \{50, \ldots, 100\}$, or the total number of hidden layers $n_l = \{2, 3, 4\}$. In other cases, such as the learning rate α where small values of α are desired to prevent unstable behavior, it is more reasonable to sample uniformly at random from a logarithmic scale. The reason for implementing a logarithmic scale for sampling is to sample more densely whenever parameters are close to zero, as this causes way more relative change to the possible outcome than values closer to one. Otherwise, too many resources will be spent on too high learning rates. Following this procedure, the range of possible outcomes can be managed in a more time efficient manner.

Different hyperparameters were tuned during the training and testing phases of model 3 and model 8, as shown in Table 6-2. The hyperparameter values of model 3 and model 8 were later used for the training of models 1,2,4,5 and models 6,7,9,10 respectively (i.e., transfer learning). The selection of hyperparameters and some additional design choices are summarized in Table 6-3. In the following, an elaboration will be given for the featured design choices and hyperparameters. Each will be discussed in detail.

Name	Symbol	Fixed or sampled	Distribution/ choice	Reference
learning rate	α	sampled	$10^r, r \in [-5,-1]$	-
activation function	$g(z)$	fixed	ReLu	Agarap [80]
weight initialization	-	fixed	He	He et al. [81]
number of hidden layers	n_l	fixed	2	-
number of hidden neurons	n_h	sampled	$[50, n_{h,max}]$	-
loss function	$\mathscr{L}(y,\hat{y})$	fixed	MSE	-
mini-batch size	-	sampled	$\{32,64,128\}$	-
number of epochs	-	variable	GRF: $[25,120]$	-
			eFootball: $[25,200]$	-
optimization algorithm	-	fixed	Adam	Kingma and Ba [63]
Adam's decay rate 1	β_1	sampled	$1-10^r, r \in [-3,-1]$	-
Adam's decay rate 2	β_2	fixed	0.999	-
Adam's correction term	ϵ	fixed	10^{-8}	-

Table 6-3: List of hyperparameters and various design choices of the prediction model.

Learning rate

Learning rate α is arguably the single most important hyperparameter and should always be tuned. The learning rate determines the proportionate amount the weights change each time they are updated. Generally, a larger learning rate means faster learning, but an increased risk of arriving at a suboptimal set of weights. Inversely, a smaller learning rate leads to an increased chance in arriving at a more optimal or possibly even a global optimum set of weights, but at the cost of a significantly longer time to train. At the extreme case of choosing a learning rate too large, it becomes possible to fall into a positive feedback loop where high weights induce high gradients, causing a large update of the weights. Ultimately, numerical overflow can occur when the weights increase consistently each update. On the other hand, when a learning rate is too small, weights may never converge or might get stuck at a local minimum or saddle point. Typical values for the learning rate are on a logarithmic scale, where a logarithmic scale from 0.1 to 10^{-5} is most common [82]. It was therefore chosen to choose the learning rate as follows,

$$\alpha = 10^r, \text{with} \quad r \in [-5,-1],$$

where r is picked from a uniform distribution when training each model. Learning rate decay was not applied to any of the models, as favorable results were already obtained in its absence.

Activation function

When a network's architectures gets large, a Rectified Linear Unit (ReLu) function is most commonly used as an activation function in neurons, as it was found by Krizhevsky et al. (2012) that

convergence of the cost function gets accelerated by a factor of 6 when compared to classical activation functions (e.g., sigmoid, Tanh). The ReLu is given by the following ramp function,

$$g(z) = z^+ = \max(0, z), \tag{6-1}$$

and outputs $g(z)$ whenever $z > 0$, and $g(z) = 0$ otherwise. The ReLu was chosen as activation function for all neurons for all models.

Initialization of weights

Weights should be initialized to prevent each neuron being identical and computing the same optimization problem, as explained in subsection 3-2-2. The bias b can be equal to zero, as a non-zero weight W initialization already solves the problem. Commonly used weight initializers are: random normal, random uniform, Xavier, Normalized Xavier, and He [60]. For ReLu activation functions, He weight initialization [81] is most commonly applied and was therefore chosen for all models.

Number of hidden layers

In 1991, Hornik [83] proposed a theorem stating that a single hidden layer in a neural network with a finite amount of neurons has the capability of approximating continuous functions and find any input-output relation. However, although it *can* approximate any input-output function, it does not define how easy or difficult it is to do so. Hinton, Osindero, and Teh (2006) [84] then found that MLP's with more than a single hidden layer were able to learn complex functions much better thank MLP's with only a single hidden layer. This means that different layer architectures have different capabilities, as theorized in Table 6-4 [85].

No. hidden layers	Result
none	Only capable of representing linear separable functions or decisions.
1	Can approximate any function that contains a continuous
2	Can represent an arbitrary decision boundary to arbitrary accuracy with rational activation functions and can approximate any smooth mapping to any accuracy.
>2	Additional layers can learn more complex features within the data, such as complex temporal sequences or images, where different data dependencies can be identified layer by layer.

Table 6-4: Capabilities of different layer architectures

The prediction algorithm of this work tries to approximate a function $ED : \mathbb{R}^{188} \rightarrow \mathbb{R}$, which should be computable using a two-layer network. The amount of layers was therefore fixed for the prediction algorithm and chosen to be:

$$n_l = 2.$$

As a side note: in the case that m was chosen large, (i.e., $m >> 0$), it would have been wise to choose $n_l > 2$, as the temporal nature and complexity of the problem would have increased massively.

Number of neurons

Although many authors have attempted to find a for an optimal amount of neurons , experience has found that there is no general formula for finding the optimal number of neurons for a given network. Each distinct ML task should be seen as its own problem, and a network should be designed by iteratively finding the right amount of neurons. However, there are certain guidelines for choosing the distribution for the sampling of this hyperparameter. An upper limit for the number of neurons in the hidden layers can be obtained using the work of Yotov, Hadzhikolev, and Hadzhikoleva (2020) [86]. They found that under certain conditions, the maximum amount of neurons in a multi-layer network can be quantified as follows,

$$n_{h,max} = \text{floor}\left[\frac{\sqrt{(n_l+n_x+2)^2+4(n_l-1)(m_s-1)}-(n_l+n_x+2)}{2(n_l-1)}\right] \tag{6-2}$$

where n_x is the number of input neurons, m_s is the number of samples in the training dataset, and n_l are the number of hidden layers. The conditions under which Equation 6-2 can be seen as reliable were satisfied, thereby creating a maximum for the total number of hidden neurons. A minimum amount of 50 neurons was taken, as a network with fewer neurons was thought to lack the potential of finding an accurate function approximator within reasonable time. To conclude, for each model the number of neurons was taken as follows,

$$n_h \in [50, n_{h,max}]$$

Loss function

The Mean Squared Error (MSE) is the most widely used loss function for regression problems and should be evaluated first [60]. The MSE is defined as the average of the squared residuals between the actual values y and the predicted value $\hat{y}$, given by the following equation:

$$\mathscr{L}(y,\hat{y}) = \frac{1}{N}\sum_{i=0}^{N}\left(y-\hat{y}_i\right)^2. \tag{6-3}$$

Generally, the MSE is used whenever the to be predicted values are normally distributed around a mean and when far-off predictions should be significantly penalized more than close predictions. In the case of this project, the Expected Danger (ED) is bounded in the interval $[-3,3]$, where $ED = 3$ and $ED = -3$ are extreme cases indicating a total attacking dominance of a single team over horizon h. Furthermore, given the user team and the opponents are of equal strength, the distribution of the ED over time will be centered around a mean of zero. It is therefore assumed that ED will be normally distributed whenever the sample size is large enough. Moreover, a more robust prediction algorithm can be created when outliers are penalized more heavily, thereby reducing the chances of large prediction errors. It was therefore chosen to use the MSE loss function for all models.

Mini-batch size

The concept of a batch is used whenever the entire dataset is used to compute the gradient for a single iteration, as weights are only updated when the algorithm has gone through all training

examples, thereby making convergence slow. This process is valid and yields accurate results, but can be extremely time-consuming when the dataset is large. Mini-batches are therefore generally used during the training phase, as they speed up the process by updating weights and biases in each mini-batch instead of the whole batch. A mini-batch is defined by a subset of the training dataset, which is then used to compute the gradient of the loss function and update the parameters. The process of using mini-batches during training can be briefly explained as follows. For k mini-batches of fixed size, the following practices are repeated:

1. forward propagation on $X^{\{k\}}$ for mini-batch k,
2. cost function $J(X^{\{k\}}, Y^{\{k\}})$ computation,
3. backpropagation for computation of gradients dW and db,
4. update of parameters W and b using gradients.

Common choices for the size of a mini-batch are: $\{32, 64, 128\}$ [60]. The above set of values were chosen as sampling values for all models.

Epochs

The process of running through the entire dataset a single time (all mini-batches) is called an *epoch*. Typically, networks are trained for multiple epochs until the minimization of the loss function stagnates or stopping conditions are reached. In this work, a trial and error technique was applied to find a reasonable value for the epochs. By looking at the rate of convergence of the loss function over the number of epochs of various models,

$$\text{number of epochs for GRF MLP's} = 120$$

$$\text{number of epochs for eFootball MLP's} = 200$$

were found to be appropriate values as a vast majority of the models have then converged to a steady-state value for the loss function.

Early stopping

Early stopping is used to prevent overfitting of the data, while also saving time by stopping further training of models where the loss function has already converged to a steady-state value. Early stopping prevents overfitting because during training, there exists a point when the model stops generalizing the data and begins to learn the statistical noise instead. This creates an increased generalization error, which, in turn, results in less accurate predictions on unseen data. In practice, early stopping is easy to implement and should therefore always be considered. In this work, early stopping was applied whenever: (1) the value of the loss function value did not improve for 10 consecutive epochs, and (2) at least 25 epochs were executed. In combination with early stopping, the number of epochs in this work can be regarded as a hyperparameter.

Optimization algorithm

Most points of zero gradient in the cost function are saddle points. In a very high dimensional space, all dimensions need to be bent downwards for it to be a local or global optimum. Therefore, it is much more likely to end up in a saddle point than a local or global minimum. A fortunate outcome is that the chances of getting stuck in a local optimum are actually very slim when having a high-dimensional space. When local optima are not really a problem in deep learning, then what is? Plateaus can slow down learning immensely. When gradients are close to zero, it can take a really long time for an optimization algorithm to get off a plateau. To counter this, variations on gradient descent algorithms, as briefly mentioned in subsection 3-2-2, can improve and speed up learning, such as: RMSprop [87], gradient descent momentum optimization [82], and Adam optimization [63].

In this work, Adam optimization was chosen as the optimization algorithm as it is very frequently used and has been proven to be very effective for many ANN's with a broad variety of architectures. It is an improvement of the traditional gradient descent and is a combination of the other optimization techniques of RMSprop and gradient descent with momentum. Both these optimization techniques will be explained briefly. Gradient descent with momentum optimization makes use of a technique with a moving average, with the advantage of cancelling the powerful effects of downwards or upwards gradients. The gradient descent with momentum is particularly effective in situations with high curvatures, noisy gradients, or little but consistent gradients, speeding up the learning rate significantly [88]. RMSprop is an optimization technique to go more accurately towards a minimum point each iteration using an automatically adaptable step size for each input variable. This adaptive step size is obtained through a decaying moving average of the partial derivatives.

Adam optimization algorithm combines the techniques of gradient descent with momentum and RMSprop. For each *mini-batch* $(X^{\{k\}}, Y^{\{k\}})$ and epoch t, the parameters w and b in each neuron get updated according to the following equations:

$$\begin{aligned}
V_{dw} &= \beta_1 V^*_{dw} + \left(1-\beta_1\right)\frac{\partial J(w,b)}{\partial w} & S_{dw} &= \beta_2 S^*_{dw} + \left(1-\beta_2\right)\frac{\partial^2 J(w,b)}{\partial w^2} \\
V_{db} &= \beta_1 V^*_{db} + \left(1-\beta_1\right)\frac{\partial J(w,b)}{\partial b} & S_{db} &= \beta_2 S_{db} * + \left(1-\beta_2\right)\frac{\partial^2 J(w,b)}{\partial b^2} \\
V^{corr}_{dw} &= \frac{V_{dw}}{1-\beta_1^t} & S^{corr}_{dw} &= \frac{S_{dw}}{1-\beta_2^t} \\
V^{corr}_{db} &= \frac{V_{db}}{1-\beta_1^t} & S^{corr}_{db} &= \frac{S_{db}}{1-\beta_2^t}
\end{aligned} \tag{6-4}$$

$$w := w - \alpha\frac{V^{corr}_{dw}}{\sqrt{S^{corr}_{dw}}+\epsilon} \qquad b := w - \alpha\frac{V^{corr}_{dw}}{\sqrt{S^{corr}_{dw}}+\epsilon} \tag{6-5}$$

In Equation 6-4, V^*_{dw} and S^*_{dw} are defined as the previous value of V_{dw} and V_{dw}, computed during the last mini-batch iteration. Apart from the learning rate α, Adam's optimizer includes multiple hyperparameters, namely: β_1, β_2, and ϵ. The gradient descent with momentum method is represented on the left-hand side of Equation 6-4, which shows recursive equations and therefore implies a moving average through the influences of past values. Thus, β_1 can be seen as a value

indicating how much weight should be given to past values. The RMSprop technique is linked to right-hand side of Equation 6-4, and shows how the correction terms S_{dw}^{corr} and S_{db}^{corr} are computed to adjust the weight updates each mini-batch iteration in Equation 6-5. Default hyperparameter values are $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$ and were also chosen as fixed values for all models. The value for β_1 was chosen as a tuning hyperparameter, sampled at random from a logarithmic scale $(0.9 - 0.999)$ as follows,

$$\beta_1 = 1 - 10^{r}, \text{with} \quad r \in [-3, -1],$$

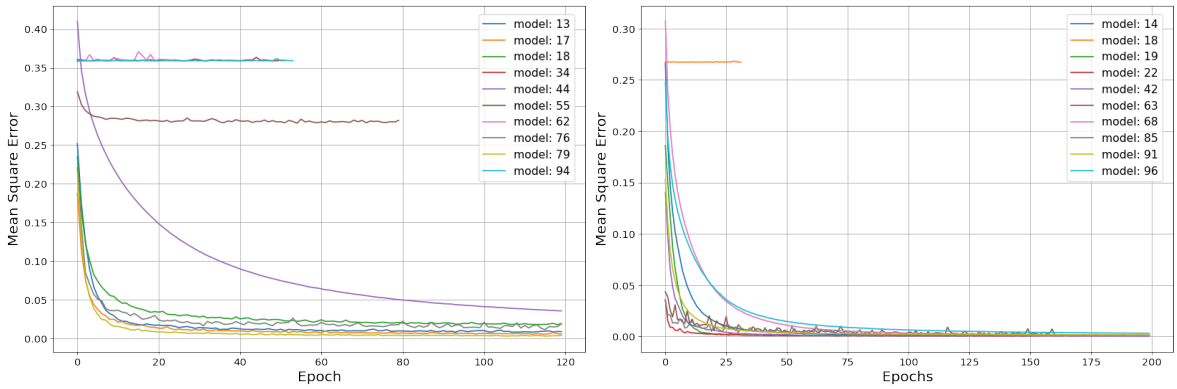
where r is picked uniformly at random.

6-3 Model Selection and Validation

The hyperparameters of the models with a horizon of $h = 15$ were tuned and validated on the validation datasets, which consist of 10% of the total amount of training examples of the initial dataset. This was done for both the models of the GRF dataset as the eFootball dataset. The two highest performing networks with their hyperparameters of both data sets were then chosen for further evaluation, as will be discussed in subsection 6-3-1. Subsequently, a separate validation of the fitted models over all horizons will be made in subsection 6-3-2.

6-3-1 Network selection

The networks were validated by evaluating the convergence of the MSE loss function over the number of epochs, as is graphically displayed in Figure 6-2. The performances of the 100 models fitted on both data sets will be evaluated.



(a) The GRF dataset. **(b)** The eFootball dataset.

Figure 6-2: Validation loss function (MSE) over epochs of 10 randomly chosen models with different hyperparameters fitted on (a) the GRF dataset and (b) the eFootball dataset.

Consider the performance of the 10 highlighted networks with different hyperparameters fitted on the GRF dataset and eFootball dataset during the training process, as shown in Figure 6-2a. Note that the performance of only 10 randomly chosen models are highlighted, but all other models were evaluated similarly. In both cases, it becomes clear that the loss function decreases steadily when the number of epochs increases, which was to be expected as the weight and bias parameters

get updated numerous times each epoch. Furthermore, the number of epochs was tuned and fixed to a maximum value of 120 by observing the convergence rates in both graphs. Both graphs show that the loss functions of almost all models converge towards a steady-state value for all models within the chosen number of epochs, thus validating this choice. Moreover, in some cases, the early stopping procedure forces the training to stop. This phenomenon is visible in both graphs. Training abruptly stop whenever no improvements are made for a certain amount of consecutive epochs. Finally, the parameters and hyperparameters of the model with the best performance in both the GRF dataset and the eFootball dataset are chosen as the best models and will be evaluated further for prediction in section 7-2. All other networks were discarded. The hyperparameters of the models with the best performance are displayed in Table 6-5.

dataset	α	n_l	n_h	n_{h1}	n_{h2}	Mini-batch size	β_1
GRF	3.78e-4	2	268	180	88	32	0.996
eFootball	3.03e-4	2	198	112	86	64	0.998

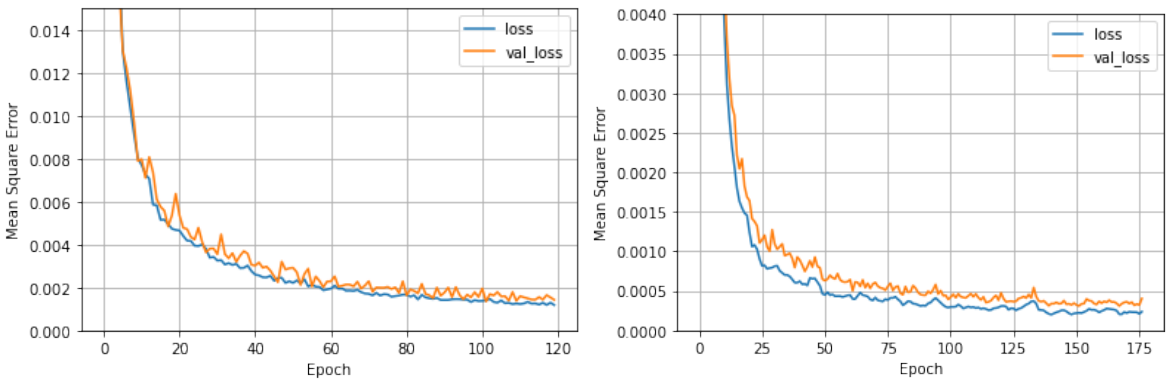
Table 6-5: Hyperparameters of the models with the best performance regarding the loss function within 120 epochs. The training process of both data sets were performed as separate entities.

6-3-2 Network validation

Within each epoch of training, the loss function gets computed after each mini-batch, after which the weight and bias parameters get their updates. In parallel to this process is the cross-validation of the values of the loss function through the validation set. The model will set this fraction of the initial dataset aside, will not use it for training, and evaluate the unbiased loss at the end of each epoch. This means two separate values for the loss functions are computed at the end of each epoch.
In ML, the *loss* generally refers to the value of the loss function of a network trained on the training set, while the *validation loss* refers to the value of the loss function of a network trained on the validation set. It is expected that the loss at the end of each epoch is lower than the validation loss, as weights and biases are updated based on the loss. A less biased performance evaluation can then be made by checking the validation loss. Conclusions from the evaluations of the loss and validation loss can really narrow down the possible solutions for improvements of the network's performance. For example, when the loss still decreases, it makes sense to train longer and increase the amount of epochs. Another example is when the validation loss is much higher than the loss. Generally, this is a sign of overfitting, as the parameters of the model are able to memorize the limited amount of training data. Possible solutions then are to: increase the size of the training data, use regularization techniques (e.g., drop-out, early stopping, L_2 regularization), or try different network architectures.

Validation of best performing networks with horizon $h = 15$

All the models as presented in Figure 6-2 describe the course of the loss and the validation loss over the number of epochs of the models with horizon $h = 15$. The loss and validation loss of the two networks with the highest performance as defined in Table 6-5 are now displayed in Figure 6-3. Performance of both models is showcased on both the training set and the validation set.



(a) The GRF dataset.

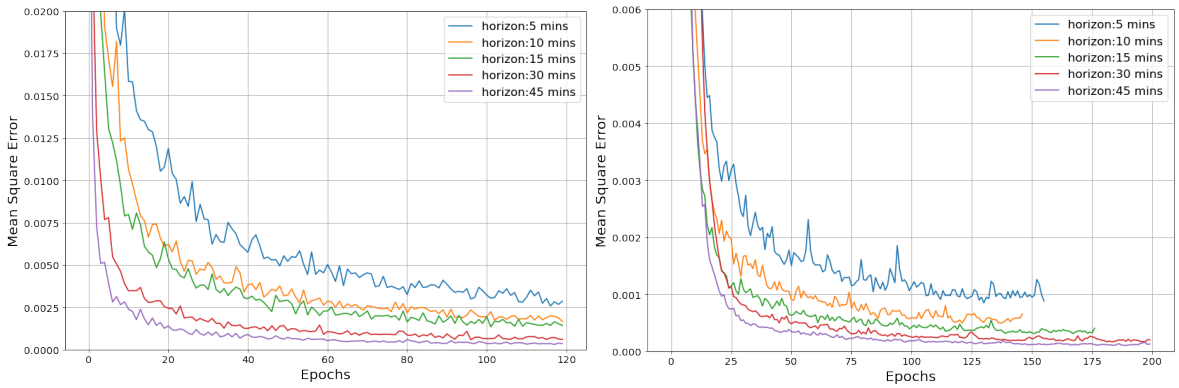
(b) The eFootball dataset.

Figure 6-3: Validation loss function and the loss function over epochs of the models with the highest performing hyperparameters and horizon $h = 15$ fitted on (a) the GRF dataset and (b) the eFootball dataset.

The loss function performance and the value loss function performance of both the GRF model in Figure 6-3a and the eFootball model in Figure 6-3b show similar performance and will therefore be analyzed together. In both cases, desired behavior is achieved as the value loss function converges towards a low steady-state value. A noticeable difference between the performance, however, is that the value loss function of the eFootball model converges to a much lower value than the value loss function of the GRF model. To be more precise, the MSE of the final eFootball model is about 3 times lower than that of the final GRF model. One would expect the GRF model to be outperforming the eFootball model, since the GRF data is much more simplistic and, as a result, should be more predictable. A possible explanation is that the GRF simulation environment may include a significant amount of added stochastic noise, thereby making it harder for the model to generalize the data and make accurate predictions.
Other favorable behavior is the validation loss staying close to the loss in both cases. This shows the models were not overfitted on the training dataset and have similar performance on a different dataset. Naturally, performance on the validation set is slightly worse as it has not been fitted on this dataset.
Finally, a noticeable observation is the slight fluctuations in the loss functions and the more heavy fluctuations in the value loss functions. The reason for the slight fluctuations in the loss functions in both graphs follows from the curvatures of the highly-dimensional cost function and the parameter updates of Adam's optimizer. After each mini-batch, the weights and biases are updated based on the relatively few training examples in that mini-batch. Weights and biases then may get updated based on an inaccurate gradient direction and, as a result, a decrease in performance can be obtained. If this happens on and off for an entire epoch, a slight upwards fluctuation upwards can occur. The most probable reason for the more heavy fluctuations in the value loss functions is that the model is able to generalize 'too much' on the examples in the mini-batch. Consequently, when the model is cross-validated on the unbiased validation set, a more noticeable decrease in performance is achieved.
To conclude, both models seem to perform well on both the training data and the validation data and were therefore used as final predictions models. Prediction results on the unbiased test set will be analysed in section 7-2.

Validation of networks with different horizons

Transfer learning is used to create prediction models with different horizons. The hyperparameters of the best performing networks with horizon $h = 15$ were saved and used to train new networks with the horizons $h = \{5, 10, 30, 45\}$, where h is measured in minutes. The networks were validated by evaluating only their *validation loss* over epochs, as displayed in Figure 6-4.



(a) The GRF dataset.

(b) The eFootball dataset.

Figure 6-4: Validation loss function over epochs of the final prediction models with horizons $h = \{5, 10, 15, 30, 45\}$ fitted on (a) the GRF dataset and (b) the eFootball dataset.

Again, analysis on the models of the GRF dataset as well as the eFootball dataset will be done together as similar performance is realized, as can be observed by comparing Figure 6-4a and Figure 6-4b. Desirable behavior is shown as the performance of all models gradually convergences towards relatively low steady-state values. A noticeable observation is that the MSE of shorter horizons shows to be worse than the MSE of longer horizons in both cases. However, it can not be said with certainty at this point whether short-term predictions are actually worse, since the predicted values of the ED might be differently distributed for different horizons.

The difference in distributions can be best explained by means of an example. In a scenario where team A is about to have a dangerous attack, a predictor with $h = 5$ will likely output a high value for the ED as the chances are high that possession will be held in dangerous regions in the coming minutes. At the other end of the spectrum, a predictor function with $h = 45$ will evaluate the same scenario based on a long-term evaluation of both teams. This means that a prediction is created largely based on the overall strengths of a team and not so much on the quality of a single attack. Considering these two extreme cases, it becomes clear that predictions with $h = 5$ will have a much lower average MSE than predictions with $h = 45$ and also have a different distribution. After all, when predictions rely too much on the predicted outcome of a single attack, chances will be higher that the predictions will be far-off. Naturally, the predictors of horizons $h = 10$, $h = 15$, and $h = 30$ will lie somewhere in between, as can be observed in both plots in Figure 6-4.

6-4 Conclusion

The objective was to train and tune various MLP's for different horizons on both the GRF dataset and the eFootball dataset. When trained and tuned, the MLP predictor functions will be able to

compute the ED for a given state $\tilde{s}$ and horizon h. Five different horizons were chosen for further research, namely $h = \{5, 10, 15, 30, 45\}$. For $h = 15$, a multitude of MLP models were trained following a standard tuning procedure, in which hyperparameters were sampled from various distributions, as displayed in Table 6-3. Out of all the models with varying parameters and hyperparameters, the best performing model was chosen as the predictor function for $h = 15$. The hyperparameters of this best performing model, as given in Table 6-5, were then used to train the models for all other horizons $h = \{5, 10, 30, 45\}$, using a concept known as transfer learning. During training and validation, the MSE of the MLP's is computed after each epoch. The results are promising, as all values show convergence towards a low steady-state value. However, the final MLP models may include a bias as the models were trained, tuned and evaluated on the same datasets. Chapter 7 will therefore elaborate on the prediction performance based on a separate test set.

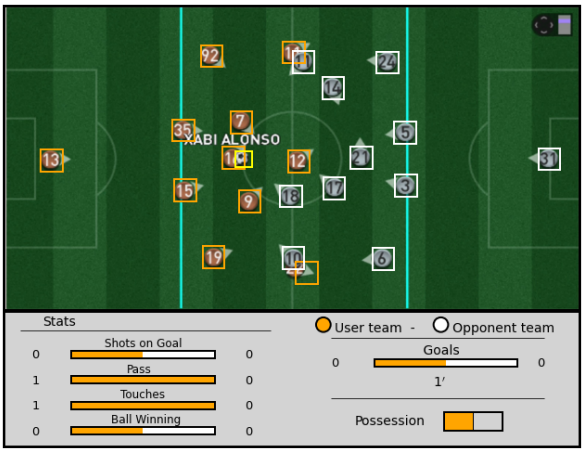
Chapter 7

Results

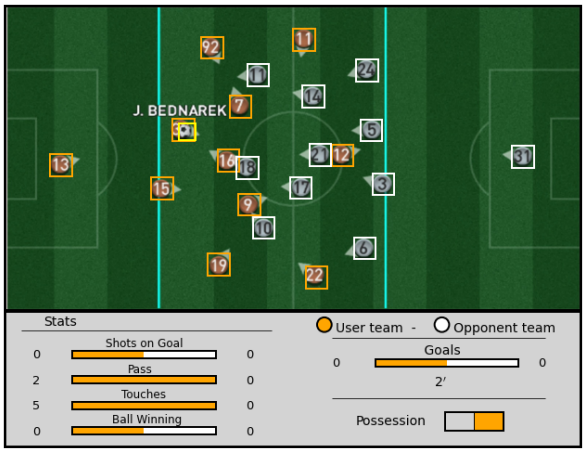
This chapter analyses the performance of the entire prediction pipeline used in this project, which includes the data extraction algorithms, as well as the Multilayer Perceptron (MLP) predictor functions. First, the performance of the data extraction pipeline will be evaluated in section 7-1. Section 7-2 will then analyze the prediction performance of the predictor functions. This analysis includes a comparative analysis of the distributions of the predicted values and the actual values in subsection 7-2-1, a prediction error analysis in subsection 7-2-2, and an evaluation of how predictions were made online subsection 7-2-3. Finally, a summary of the achieved results and conclusions is presented in section 7-3.

7-1 Data Extraction Performance

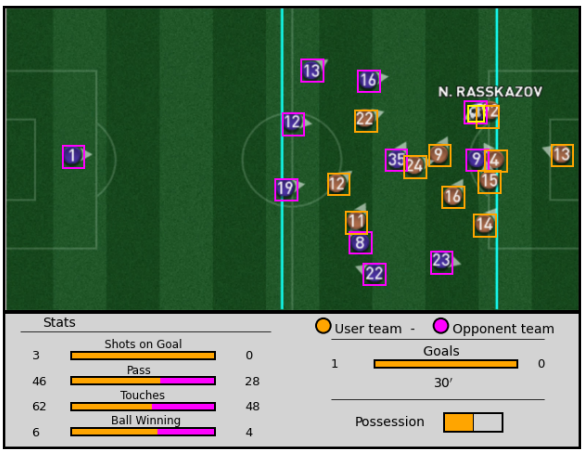
This section discusses the performance of the data extraction algorithm that was used within the eFootball simulation environment, as touched upon in chapter 5. The extraction of the data consists of: the detection of players and ball, the identification of the team in possession, the extraction of various event statistics, and the interpolation and extrapolation of missing data. For each game, performance was checked using visual inspection of randomly sampled frames by plotting the extracted data on top of the original frame. Detection of players on both teams was visualized using a colored box around each player such that differences within both teams could be identified. The ball was given a bright yellow box for all games. Furthermore, a separate interface was designed for the display of the extracted event data (i.e., shots on goal, touches, ball winning, goals, minute), and the identified team in possession. A selection of these frames will be presented that is indicative of the data extraction performance of all other frames. These frames have been selected to give the reader an impression of the accuracy of the algorithm, and can be observed in Figure 7-1.



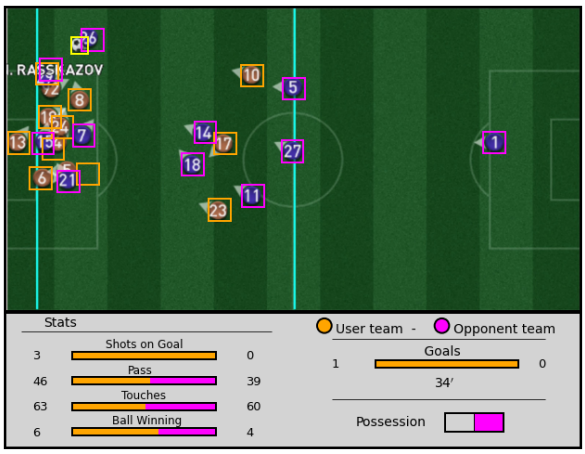
(a) Kick-off by user team 1



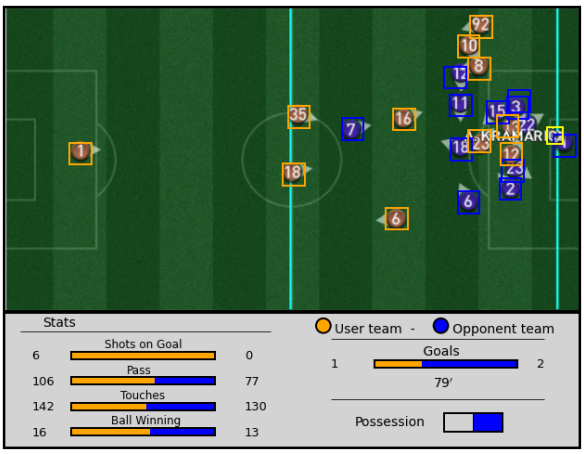
(b) Kick-off by user team 2



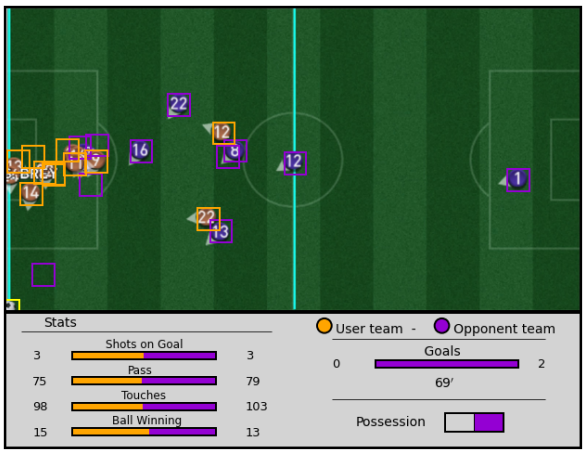
(c) Attacking play by opponent team



(d) Corner by opponent team



(e) Attacking play by opponent team



(f) Corner by opponent team

Figure 7-1: Examples of frames of different games. The detected players on the both teams have their own distinct colored boxes around them, while the ball is boxed in by a yellow square. The detected team in possession is displayed by its color in the bottom-right part of the images, while other extracted event statistics are displayed in the rest of the designed interface.

The following observations and conclusions were made based on visual inspection of a large variety of frames over all games in the eFootball dataset:

- *The algorithm performs very well on the detection and tracking of the ball.* No false matches were found and a very high detection accuracy was obtained. Examples of perfect detection of the ball are given in all sub-figures of Figure 7-1. High performance could be obtained for two reasons. First, the ball always moves on top of the pitch and the players, and is therefore always visible. Second, the configuration of the pixels in the template of the ball does is rather unique and can therefore be identified relatively easy. Furthermore, the template does not change over the course of different matches and could thus be perfected in a way that only the relevant pixels remained within the template.

- *Players can be detected almost perfectly when they do not overlap with one another.* Whenever the templates of players are completely visible without any overlap, the algorithm accurately finds their locations, as can be observed in Figure 7-1a and Figure 7-1b. In these scenario's, the templates can be matched perfectly by the algorithm in almost all cases.

- *Player detection becomes less accurate whenever players move behind each other for an extended period of time.* Whenever players are hidden behind other players, the algorithm uses the last known locations and calculates the likely trajectories using interpolation techniques. In the case that players are hidden behind other players for a brief amount of time, the algorithm is able to accurately compute these trajectories in most cases, as can be seen in Figure 7-1c, Figure 7-1d, and Figure 7-1e. However, inaccuracies occur in some cases when the players are hidden over a longer period of time, as can be seen in Figure 7-1d and Figure 7-1e.

- *Players are not detected well in the initial phase of the following events: corners, free-kicks, penalties, and kick-offs.* The reason is that whenever such a scenario takes place, all players are re-located instantaneously into certain pre-defined positions. Whenever players are then re-located behind other players, the algorithm acquires their positions based on their last known location and interpolation techniques. Consequently, the acquired locations of players can be highly inaccurate until the moment they re-appear. For example, a player might get moved across the entire field, only to be re-located behind another player for an extensive period of time. In this case, the algorithm interpolates between the two last known locations in time and is therefore not able to capture the actual trajectory of the player properly. Such an example is illustrated in Figure 7-1e, which shows how not all players are correctly detected when a corner is taken. Player detection becomes accurate again whenever the player moves out of the crowd without any other overlapping players.

- *Detection of the team in possession is reasonably accurate.* By identifying and comparing the colors around the ball with the colors of the team jerseys, an accurate extraction of the team in possession could be obtained. This is showcased in the bottom-right section of all sub-figures of Figure 7-1, where all of the possession teams were identified correctly. However, sometimes it was unclear from the frames which team is actually in possession, as the frames only provide a top-view. In such uncertain cases, the algorithm assumes the team in possession is the team which most closely matches the colors, although this may not always be the case. An example of such an uncertain case can be observed in Figure 7-1c.

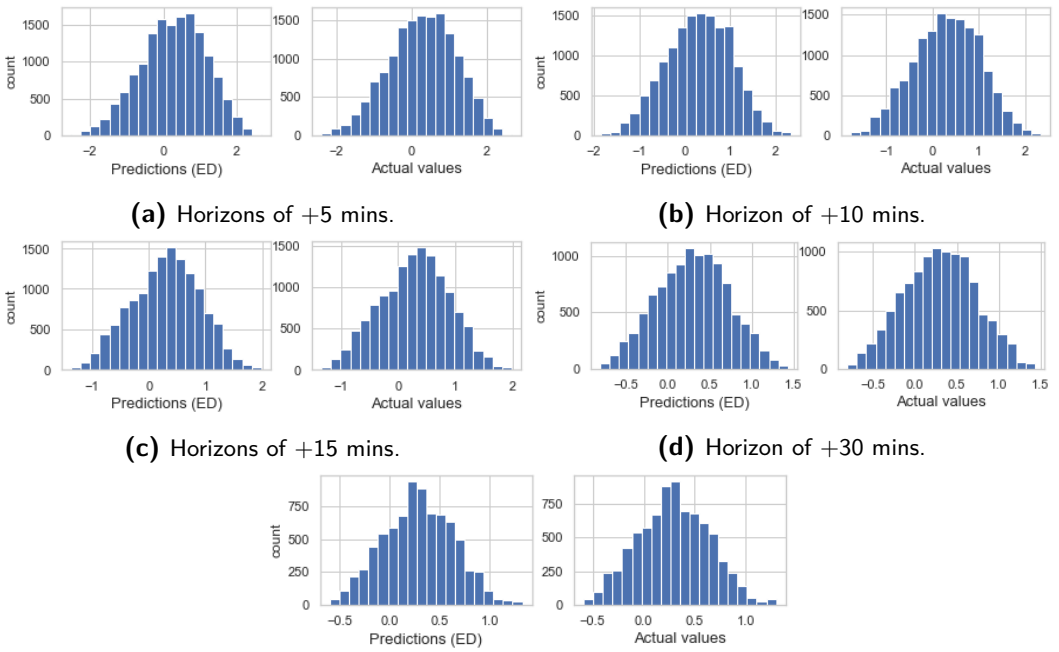
- *Detection of events is reasonably accurate.* The extracted event statistics at the end of each game were compared with the actual statistics as presented in the interface of the simulator. This comparison showed the statistics are similar in most cases, although some did not match exactly. One reason for these slight inaccuracies comes from the fact that the sampling rate of the frames going into the algorithm was not high enough for perfect tracking of the numbers. For example, when two touches occur within the time boundaries of two consecutive frames, the algorithm only adds one to the 'touches counter' instead of two. Other reasons for the inaccuracies were not further explored, as the prediction algorithm does not use this particular part of the data. Nonetheless, an example of correct event detection can be seen in Figure 7-1a and Figure 7-1b, which shows two frames closely after each other. In the time between the frames, 4 touches and 1 pass occur, and the timer (i.e., current minute) increases by a value of one, while all other statistics stay the same and have a value of zero.

7-2 Prediction Performance

This section will discuss the performance of the predictor functions for the computation of the Expected Danger (ED). All predictor functions were created by training and tuning different MLP's. As discussed in section 6-2, an 80/10/10 division was used for the training, validation, and testing of the network. The remaining 10% of the data will now be used for testing purposes. Results and evaluations of the predictions on this test set will be presented in subsection 7-2-1 and subsection 7-2-2. Predictions were made on both the Google Research Football (GRF) dataset and the eFootball dataset and will be analyzed in the following parts. Furthermore, a new dataset consisting of a single game was generated within the eFootball simulation environment to test and evaluate the prediction performance online, which will be discussed in subsection 7-2-3.

7-2-1 A comparative analysis of the distributions

The predictions and actual values of the ED were computed across all horizons using the data from the test sets of the two simulators. The distributions of the predictions and actual values will be compared in the following parts. A visualization of the distributions of all horizons of the GRF dataset is given in Figure 7-2, while Figure 7-3 shows the distribution of the eFootball dataset. Table 7-1 then shows various relevant statistics of the distributions as well as some comparative metrics, which are as follows: the sample size, the mean, the Standard Deviation (STD), the Shapiro-Wilk statistic, the Kolmogorov-Smirnov (K-S) statistic, the minimum value, the first quartile or 25th percentile (Q1), the median, the third quartile or 75th percentile (Q3), and the maximum value. Note that the sample size of each distribution differs per horizon, as different labelled examples need to be made for each horizon. For example, labeled examples with a horizon of $h = 45$ can only be created from data from the first half, as data from the second half is used as the prediction horizon. Both the Shapiro-Wilk statistic as the K-S statistic will be explained after the figures.
A number of distribution properties are also depicted using boxplots, as illustrated in Figure 7-4. A boxplot gives a summary of a distribution of data by visually showing the minimum value, Q1, the median, Q3, and the maximum, thereby giving a perfect overview for comparative analysis. Outliers are excluded from the boxplot itself, but are plotted separately as points. Finally, a comparative analysis of the distributions will be given, where conclusions will be drawn based on the properties of the different distributions.



(a) Horizons of $+5$ mins.

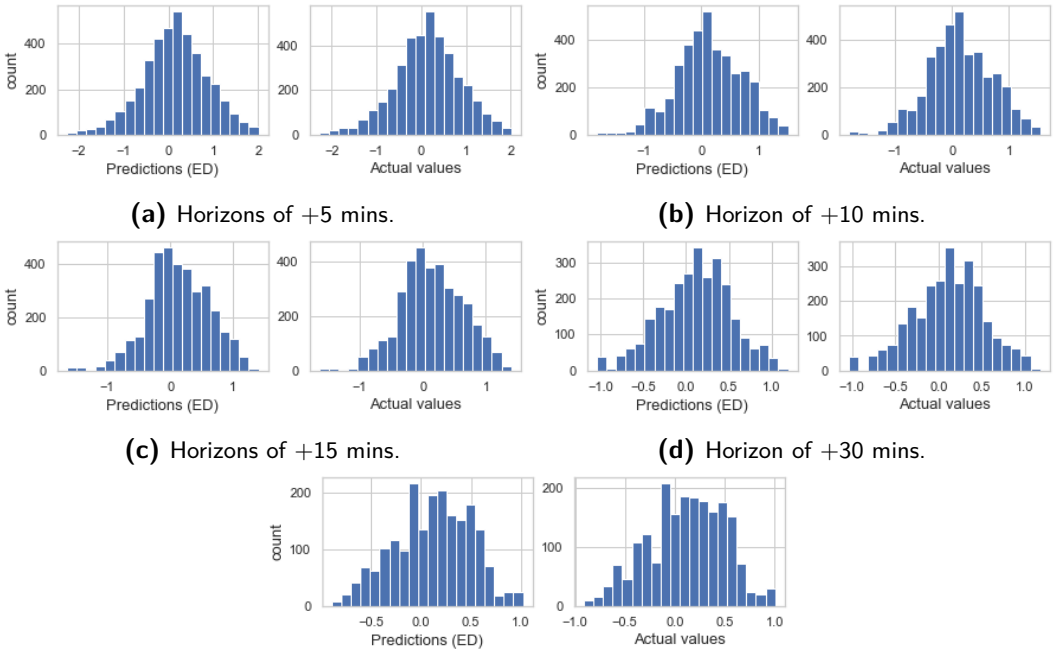
(b) Horizon of $+10$ mins.

(c) Horizons of $+15$ mins.

(d) Horizon of $+30$ mins.

(e) Horizons of $+45$ mins.

Figure 7-2: Plots of distributions of the predicted values (ED) and the actual values (G_t^h) for all horizons of the models trained on the GRF dataset.



(a) Horizons of $+5$ mins.

(b) Horizon of $+10$ mins.

(c) Horizons of $+15$ mins.

(d) Horizon of $+30$ mins.

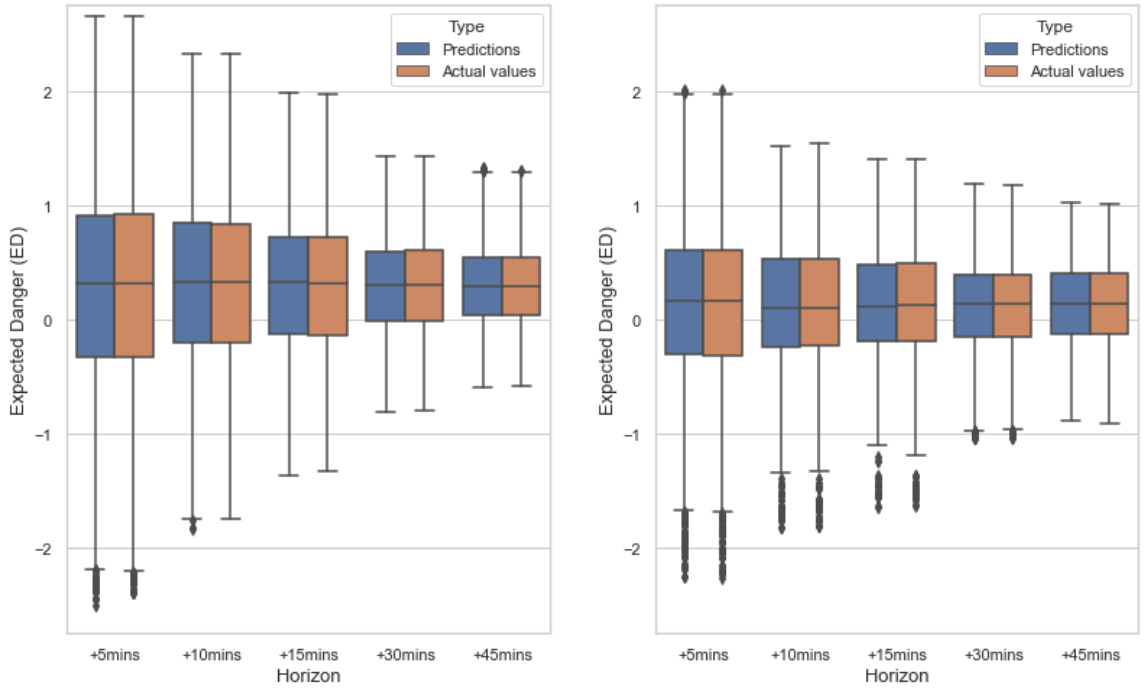
(e) Horizons of $+45$ mins.

Figure 7-3: Plots of distributions of the predicted values (ED) and the actual values (G_t^h) for all horizons of the models trained on the eFootball dataset.

Dataset	Horizon	Sample size	Mean		STD		W		K-S	
			Preds.	*Actual*	*Preds.*	*Actual*	*Preds.*	*Actual*	*Stat.*	*p-value*
GRF dataset	*+5*	15,125	0.268	0.273	0.884	0.887	0.994	0.995	0.008	0.725
	+10	14,341	0.304	0.301	0.717	0.719	0.996	0.996	0.006	0.981
	+15	13,333	0.297	0.287	0.598	0.598	0.995	0.995	0.011	0.407
	+30	10,645	0.291	0.294	0.426	0.429	0.995	0.995	0.009	0.754
	+45	7,957	0.290	0.291	0.347	0.347	0.997	0.997	0.006	0.999
eFootball dataset	*+5*	4,001	0.141	0.140	0.730	0.732	0.996	0.996	0.008	0.999
	+10	3,757	0.134	0.137	0.567	0.569	0.995	0.995	0.011	0.950
	+15	3,512	0.129	0.144	0.493	0.500	0.994	0.994	0.009	0.999
	+30	2,786	0.112	0.112	0.414	0.418	0.994	0.994	0.007	0.999
	+45	2,028	0.114	0.113	0.382	0.395	0.990	0.990	0.010	0.999

Min.		Q1		Median		Q3		Max.	
Preds.	*Actual*	*Preds.*	*Actual*	*Preds.*	*Actual*	*Preds.*	*Actual*	*Preds.*	*Actual*
-2.26	-2.27	-0.308	-0.315	0.162	0.167	0.604	0.605	2.01	2.01
-1.82	-1.80	-0.237	-0.231	0.104	0.105	0.527	0.526	1.51	1.54
-1.64	-1.64	-0.191	-0.186	0.117	0.119	0.485	0.488	1.41	1.40
-1.04	-1.03	-0.156	-0.152	0.132	0.131	0.393	0.391	1.19	1.17
-0.892	-0.910	-0.130	-0.127	0.137	0.138	0.409	0.408	1.02	1.00
-2.50	-2.34	-0.330	-0.328	0.310	0.311	0.911	0.922	2.65	2.66
-1.84	-1.75	-0.198	-0.203	0.325	0.322	0.840	0.838	2.33	2.32
-1.37	-1.33	-0.129	-0.140	0.325	0.314	0.721	0.714	1.98	1.98
-0.807	-0.803	-0.015	-0.014	0.299	0.296	0.593	0.602	1.43	1.43
-0.591	-0.587	0.041	0.039	0.283	0.286	0.542	0.539	1.33	1.30

Table 7-1: Properties of the distributions of the predicted and actual values of the ED. The distributions of the predictions and the actual values of each of the 10 predictor functions are considered.



(a) Boxplots of GRF predictor functions

(b) Boxplots of eFootball predictor functions

Figure 7-4: Boxplots of the distributions of the predicted values and actual values of the ED over all horizons of both the GRF predictor functions in **(a)** and the eFootball predictor functions in **(b)**.

The following observations were made based on the distributions, as illustrated in Figure 7-2 and Figure 7-3, the distribution properties as summarized in Table 7-1, and the boxplots as presented in Figure 7-4:

- *The distributions closely follow normal distributed behavior.* As can be seen in all plots of Figure 7-2 and Figure 7-3 the distributions look like they come from a normal distributed population, although the shapes look less like a normal distribution whenever the sample size is relatively low, e.g., the eFootball predictor function with $h = 45$. For verification, a test was performed on each distribution to check whether the sample data comes from a population with a normal distribution. Based on Monte Carlo simulations, it was found that the Shapiro-Wilk outperforms other tests of normality statistics (e.g., the Anderson-Darling test, the Lilliefors test, and the Kolmogorov-Smirnov test) [89]. The Shapiro-Wilk test computes a statistic in the $[0,1]$ range, with 1 being a perfect match. The test statistic is given by the following equation:

$$W = \frac{\left(\sum_{i=1}^{n} a_i x_{(i)}\right)^2}{\sum_{i=1}^{n} (x_i - \bar{x})^2}, \tag{7-1}$$

  where $\bar{x}$ is the mean of the sample and a_i with $i = \{1,2,\ldots,n\}$ are weight coefficients [90]. Based on the results of W, as given in Table 7-1, it was found that the distributions closely resemble normal distributed populations.

- *The distributions are centered around a non-zero mean.* The mean values of the distributions from the GRF test set are all in the $[0.2, 0.4]$ range, and the mean values from the distributions from the eFootball test set are all in the $[0.1, 0.2]$ range, as can be observed in Table 7-1. It was to be expected that the distributions were going to be centered around a non-zero mean. A non-zero mean means that one team has had the upper edge over the other team in terms of the ED. The value of the mean therefore quantifies the relative strength of the user team compared to its average opponent. A high mean value indicates that the user team has outperformed its opponents in terms of the ED, whereas a low mean can be linked to worse performance. In football, one team will always have an upper edge over the other, even if it is just the thinnest of margins. This behavior was thus expected.

- *The distributions of the predictions and the actual values are very similar for each horizon.* As can be observed in the Table 7-1 and the plots of Figure 7-2 and Figure 7-3, the distributions of the predictions and actual values seem to closely match each other. The K-S test, a test for measuring the differences between two distribution, was performed for verification purposes and includes a null-hypothesis and a p-value [91]. The test aims to compute the probability that two sample sets were drawn from the same, but unknown, probability distribution by computing the K-S statistic. The K-S statistic quantifies the maximum distance between the cumulative distribution functions of the two samples and is computed as follows:

$$D_{n,m} = \sup_x \left| F_{1,n}(x) - F_{2,m}(x) \right| \tag{7-2}$$

  where F_1 and F_2 are the cumulative distribution functions of the first and the second sample respectively, where $F_1 \in [0,1]$ and $F_2 \in [0,1]$, and *sup* is the supremum function. The null-hypothesis is that the samples are from the same distribution, which is rejected whenever $D_{n,m}$ reaches above a certain value. The p-value then quantifies the probability that the cumulative distribution functions are as far apart as observed. If the p-value is small (close to 0), a conclusion can be made that the samples are not from the same distribution. If the p-value is large (close to 1), we cannot conclude anything, but chances will be high that the samples are from the same distribution.
  The p-values in the case of the predictions made on the GRF dataset, as presented in Table 7-1, show varying values within the $[0.4, 1]$ range. No decisive conclusions can therefore be made about the similarities of the distributions. From the p-values as presented in Table 7-1, which are mostly very high for both the GRF- and the eFootball predictor functions, it can be deduced that the distributions of the predictions and actual values are very similar for all horizons that satisfy $p \geq 0.95$.

- *The mean of the distributions of the ED seems to converge towards a steady-state value whenever the horizon gets larger.* As can be seen in Table 7-1, the mean of the distributions seems to stabilize towards values of approximately 0.29 and 0.13 for the GRF- and eFootball predictor functions respectively when the horizon goes to $h = 45$. A reason for this behavior might follow from the law of large numbers, which states that the observed sample mean of a sample of independent and identically distributed random variables approaches the expected value of the true population whenever the sample size goes to infinity [92]. In this project, the rewards are not independent variables because subsequent rewards in time do, in fact, influence each other. However, whenever the horizon is chosen very large, as is the case with $h = 45$, the far-off rewards are not likely to be influenced much by the current state. Consequently, the expected value for far-off rewards will be similar to the expected average value

of the set rewards that is linked to the state set $\tilde{S}$.
In mathematical terms, given that h is chosen very large, we can approximate the ED as the expected reward in the following way:

$$ED^h(\tilde{s}) = \mathbb{E}[G_t^h \mid \tilde{s}] = \frac{\mathbb{E}[\theta_{t+1}] + \mathbb{E}[\theta_{t+2}] + \ldots + \mathbb{E}[\theta_{t+h}]}{h} \approx \frac{h\mathbb{E}[\bar{\theta}]}{h} = \mathbb{E}[\bar{\theta}], \tag{7-3}$$

where $\bar{\theta}$ is defined as the average reward over all states in $\tilde{S}$.

- *The STD of the distributions decreases whenever the horizon is chosen larger.* As is perfectly visualized in the boxplots of Figure 7-4, the STD's of the distributions decrease gradually whenever the horizon gets larger. The mean values and median values, however, do not seem to show a particular upwards or downwards trend whenever the horizon increases. This observation can be explained based on the fact that short-term predictions will vary between a much larger range of values than long-term predictions. The average, absolute reward can take values on much larger values whenever the horizon is short, since a single play during the game will have a much higher influence on the average reward. For example, when the user team is in an attack and has possession of the ball in regions with a high danger value, the average reward will increase much more whenever the horizon only considers the next 5 minutes instead of the next 45 minutes.

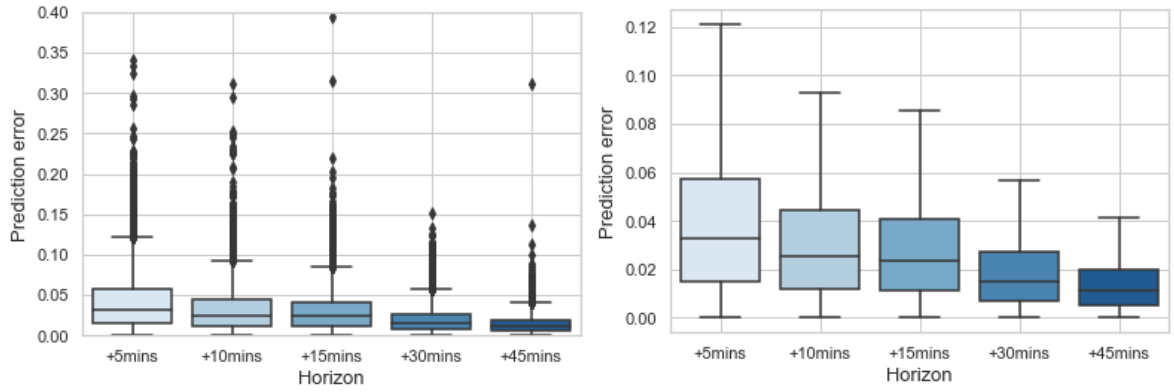
7-2-2 Prediction error analysis

The predictions errors, i.e. the difference between the predicted values (ED) and the actual values (G_t^h), will be analyzed in the following parts. As touched upon in section 3-2-3, the most common performance metrics for regression-based predictions are the: Mean Absolute Error (MAE), the Mean Squared Error (MSE), and the R-square (R^2). The MAE is given by the average error of the predictions, and is therefore a very intuitive representation of the error of the ED. However, it should be noted that the models of different horizons have different distributions. Decisive conclusions from a direct comparison of the MAE between models of varying horizons can therefore not be made. The MSE is similar to the MAE, however, in contrast to the MAE which gives each prediction error the same weight, the MSE penalizes larger errors more and therefore serves as an additional performance evaluation tool. Finally, the R-square is a value in the set $[0, 1]$, where a value close to 1 indicates a low error between the predicted values and the actual values. R-square works well on models based on systems with lots of dependent variables, and can thus be a useful measure.

All the aforementioned metrics for all predictor functions have been computed and are displayed in Table 7-2. Furthermore, the entire distributions of the prediction error over all horizons have been visualized using boxplots in Figure 7-5 and Figure 7-6.

dataset	Prediction horizon h [mins]	MAE ($*10^{-2}$)	MSE ($*10^{-4}$)	R^2
GRF dataset	+5	4.07	28.0	0.992
	+10	3.15	17.0	0.993
	+15	2.90	14.7	0.995
	+30	1.92	6.28	0.996
	+45	1.41	3.54	0.997
eFootball dataset	+5	2.25	8.60	0.994
	+10	1.97	6.77	0.995
	+15	1.42	3.49	0.998
	+30	1.01	1.90	0.998
	+45	0.91	1.45	0.999

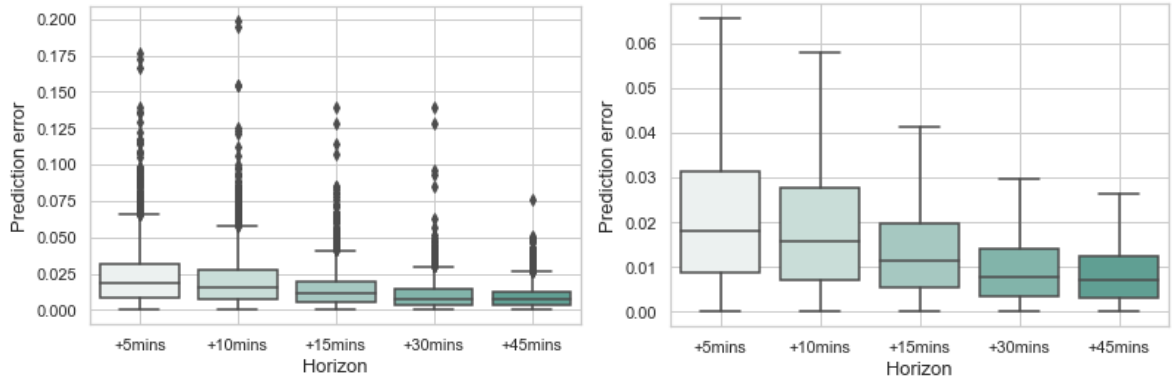
Table 7-2: Prediction performance metrics of the predictor functions of both the GRF- and the eFootball dataset over all horizons.



(a) Zoomed-out with outliers

(b) Zoomed-in without outliers

Figure 7-5: Boxplots of the prediction errors zoomed-out with outliers in **(a)** and a close-up without outliers in **(b)** over all horizons trained on the GRF dataset.



(a) Zoomed-out with outliers

(b) Zoomed-in without outliers

Figure 7-6: Boxplots of the prediction errors zoomed-out with outliers in **(a)** and a close-up without outliers in **(b)** over all horizons trained on the eFootball dataset.

Based on the predictions made on the test set, the following observations can be made from Table 7-2, Figure 7-5, and Figure 7-6.

- *The overall prediction error decreases whenever the horizon gets larger.* A downwards trend in the prediction errors can be observed from the MAE and MSE in Table 7-2, and also from Figure 7-5 and Figure 7-6. This was to be expected, as both the predictions and the actual values of the ED are much lower when the horizon is large, as followed from subsection 7-2-1. This does thus not directly imply that predictions are also better when the horizon is large, as the difference in the variance (or STD) of the distributions still needs to be accounted for. This will be further analyzed in subsection 7-2-3.
- *The predictions are very accurate.* The R-square evaluates how well the model has been able to fit the data and is a good indication of how accurate the predictions have been on the given dataset. Based on the values in Table 7-2, which are all close to 1, it can be deduced that the fitted values almost all fall close to the fitted regression line. This indicates that the predictions are very accurate, as can also be observed from the boxplots in Figure 7-5 and Figure 7-6, which show an overview of the distributions of the prediction error. Apart from the outliers errors, all prediction errors for the horizon $h = 5$ fall within the ranges $[0, 0.13]$ and $[0, 0.07]$ for the GRF and eFootball predictor functions respectively. These values are extremely low considering that the actual values for the ED fluctuate between $[-2.27, 2.01]$ and $[-2.34, 2.66]$ for the GRF and eFootball predictor functions respectively. Similar behavior can be observed for all other horizons.

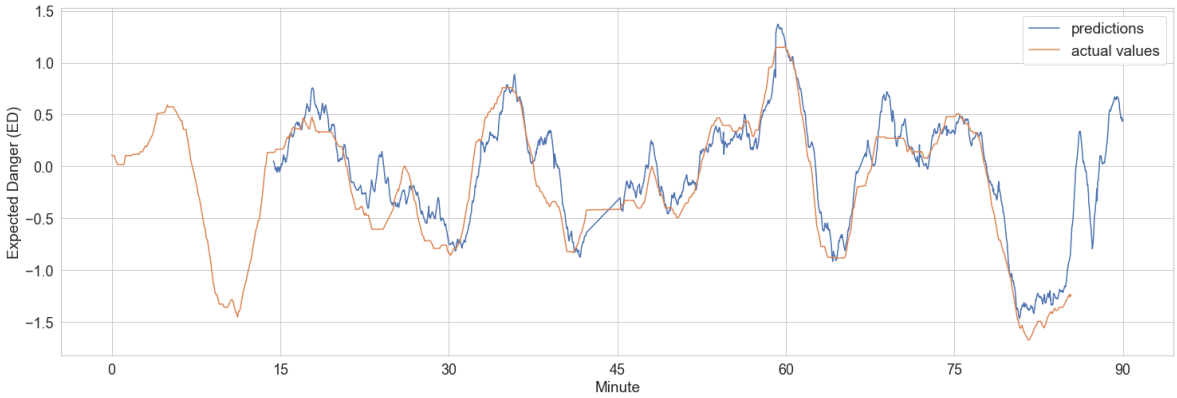
The observed prediction errors were not quite as expected because, typically, any field that tries to predict human behavior has much lower predictive performance values. A possible reason for the high accuracy of the predictor functions could be because there may be a certain bias included in the test set. The reason for this possible bias will be explained in the following. Recall that the initial dataset of all labelled examples were randomly shuffled and, subsequently, split following an 80/10/10 division for the training, validation and testing. Now consider two labelled examples in the initial dataset, which were sampled right after each other at time k and time $k+1$. The probability that these labelled examples contain the same or very similar values is rather high, as the sampling frequency was chosen at about 0.5 frames per in-game seconds. Consequently, a bias might be present whenever these labelled examples are split apart, and one is moved to the training set and the other to the test set. Whenever this scenario is repeated for all labelled examples, the test set might not include as many unbiased examples as one would desire. Instead, it would now contain a large collection of duplicates (or highly similar examples) from the training set. This is undesired as it does not present a new unknown situation for the predictor function to be tested upon. However, it would explain the high predictive accuracies of all the predictor functions.

In hindsight, a better dataset creation approach would have been to first split the labelled examples into chunks, where each chunk includes labelled examples over a consecutive time interval (e.g. time interval $[k, k+n]$ with n>1), and only then split the dataset into an 80/10/10 division. As a result, the test set would now contain more examples that are unknown to the predictor function. This would have at least reduced the suspicion of bias. Nonetheless, the prediction results can be seen as favorable, as it seems to be possible to make highly accurate within-game predictions within the limitations and assumptions of this project.

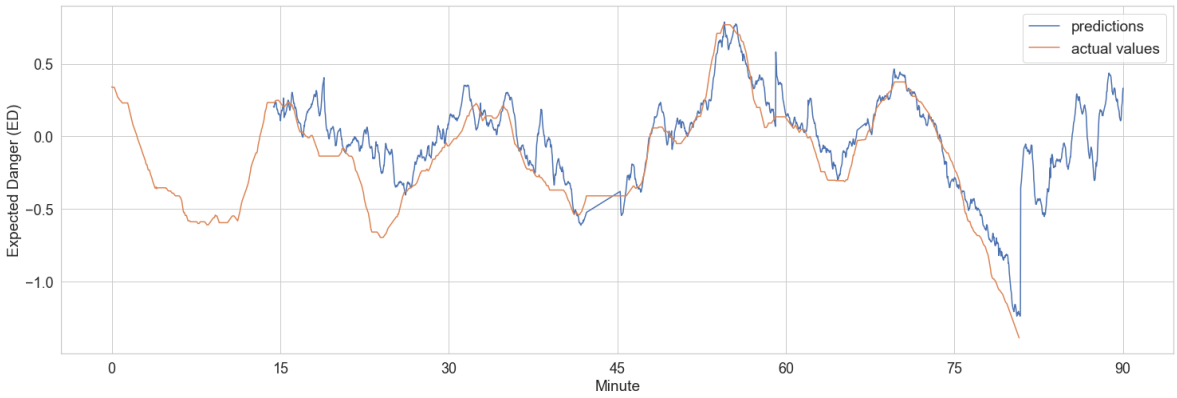
7-2-3 Online prediction performance

A final experiment was conducted to test the performance of all predictor functions online, using a dataset of a newly generated game. Only one extra game of data from the eFootball simulator was generated due to time constraints. The results will show the evolution of the ED over the course of the match, thereby revealing how the ED can be used online as intended.

The predictions, as well as the actual values, are plotted over the entire match as illustrated in Figure 7-7, Figure 7-8, and Figure 7-9. The predicted values only start after 15 minutes, as 15 minutes of prior data is needed as input to the predictor functions of all horizons. Similarly, the actual values of the ED end prematurely at a certain moment in time as the minutes thereafter are needed for its computation. Finally, the MAE's of all predictor functions are displayed in Table 7-3 and provide an intuitive representation of the prediction performance.



(a) Prediction horizon of +5 mins.



(b) Prediction horizon of +10 mins.

Figure 7-7: Expected Danger (the predictions) for the shorter horizons $h = \{5, 10\}$ over the course of a single match plotted together with the actual values (G_t^h).

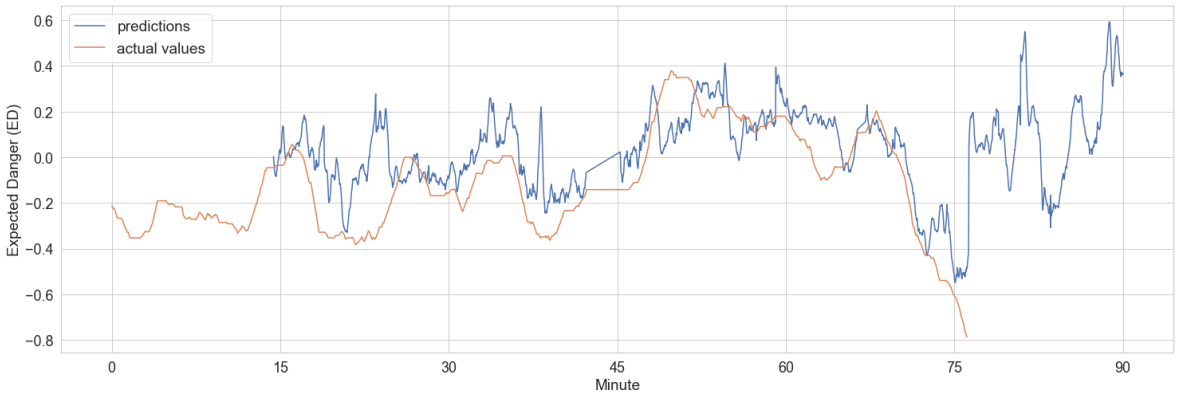
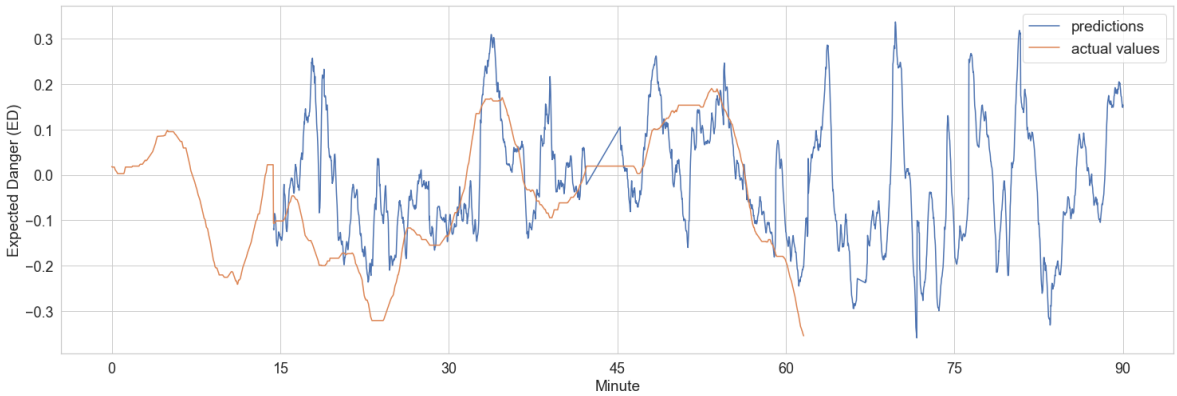
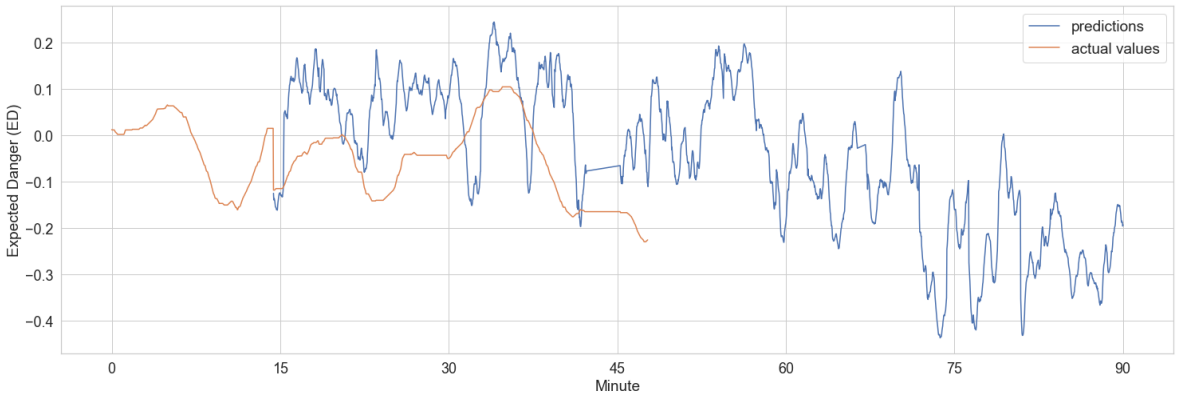


Figure 7-8: Expected Danger for the horizon $h = 15$ over the course of a single match plotted together with the actual values.



(a) Prediction horizon of $+30$ mins.



(b) Prediction horizon of $+45$ mins.

Figure 7-9: Expected Danger for the longer horizons $h = \{30, 45\}$ over the course of a single match plotted together with the actual values.

Prediction horizon h [mins]	**MAE**	**MSE**	R^2
+5	0.170	0.0463	0.88
+10	0.128	0.311	0.79
+15	0.135	0.0289	0.47
+30	0.130	0.0161	0.22
+45	0.215	0.0218	-2.16

Table 7-3: Mean absolute error of the predictions on the newly generated dataset consisting of a single game. The data was extracted from the eFootball simulator.

The following observations about the predictions based on the additional dataset of a single game can be made from Figure 7-7, Figure 7-8, Figure 7-9, and Table 7-3:

- *Predictions can be made reasonably accurate for short horizons.* From the plots in Figure 7-7 and Figure 7-8, it is clearly visible that the short-term predictions (i.e.,$h = \{5, 10, 15\}$) follow a similar trajectory as the actual values, thereby already showing that reasonably accurate predictions can be made online. This accuracy is quantified in terms of the MAE and MSE, which show fairly low values. For example, for the $h = 5$ predictor function, the MAE is equal to 0.17. Now recall from the observations that were made from the values in Table 7-1 that the training dataset of the eFootball predictor function of $h = 5$ is similar to the normal distribution of $N(0.14, 0.73)$. Based on the properties of this distribution, it can be deduced that the value of the MAE's indicates reasonably accurate prediction performance. Similar observations and conclusions can be made for the horizons $h = 10$ and $h = 15$.

- *Prediction performances decrease when the horizon is chosen larger.* The prediction performance becomes worse whenever the horizon gets larger (i.e.,$h = \{30, 45\}$), as can be seen when comparing all the plot and from Figure 7-9. This observation can also be made from the R-square values in Table 7-3, which clearly shows a decreasing trend when the horizon increases. The low R-square values indicate that the predictions are relatively poor. In the case of the $h = 45$ predictor function, the R-square value is even negative. This indicates that the prediction models have been generalized on data that is not similar to the new data, and is therefore not able to find the correct values. To be more precise, the predictor function has been trained on data where the user team generally outperforms the opponent team. However, in the case of this particular game, the opponent team seems to improve its overall ED, as the actual values are negative and mostly lie below the predicted values. This also explains why the R-square is negative.
  In addition, it should be noted that based on the MAE and the MSE it seems like the prediction performance does not decrease whenever the horizon is chosen larger. However, one should also take into account the decreasing variance of the ED whenever the horizon increases. For example, in the test set, the STD went from STD = 0.732 to STD = 0.395 for the $h = 5$ and $h = 45$ predictor functions, respectively (Table 7-1). However, the MAE and MSE have not improved in a similar trend as the STD, as can now be seen from Table 7-3, which further emphasizes that prediction performance decreases whenever the horizon gets larger.

- *Predictions can not be made accurately for longer horizons.* As can be observed from Figure 7-9, predictions do not seem to follow the actual values accurately, which is further emphasized by the low R-square values. A possible solution might be to increase the size of the original dataset such that it consists of more games. This way, the algorithm can be trained

on a more diverse set of football scenario's, such that it will hopefully recognize a wider range of relevant features when presented with an unseen scenario and improve its performance.

- *The prediction errors in this experiment are significantly higher than the predictions errors from the last experiment.* When comparing the performance metrics of this experiment as given in Table 7-3 with the performance metrics of the last experiment as given in Table 7-2, it can be observed that all metrics indicate worse performance. A plausible reason for this behavior is that the test set of the previous experiment may have a certain bias. As mentioned in subsection 7-2-2, this bias may have been caused by the way the labelled examples were shuffled during the train-, validation-, and test set division phase. Now that a new dataset has been generated, this assumed bias no longer exist. As a result, the prediction errors on the new dataset reflect a more realistic prediction performance.

7-3 Conclusions

This section will evaluate the performance of the entire prediction pipeline as used in this work. An evaluation of the data extraction algorithm was made based on the observations and conclusions made in section 7-1. Conclusions about the prediction pipeline and predictor functions were made based on the observations and analyses made in subsection 7-2-2 and subsection 7-2-3. The main takeaways are summarized in the following points:

- *The data extraction pipeline is able to extract highly accurate tracking data from the eFootball simulator.* The detection accuracy of the pipeline before the interpolation process already showed promising performance, in which players and ball were detected 80.0% and 88.2% of the time over all frames. Visual inspection of the obtained tracking data then concluded that the tracking data is highly accurate in the vast majority of the cases. Some inaccuracies were identified whenever players go missing for an extended period of time, or when the game abruptly moves into a new scenario, which is the case for certain events, such as corners and free-kicks. However, these scenario's only occur sporadically and do not last long whenever they do occur.

- *The data extraction pipeline is able to extract event data with a reasonably high accuracy.* Two different segments within the data extraction pipeline exist: one for the detection of the possession, and the other for the detection of all other events (i.e., shots on goal, passes, touches, balls won, goals, current minute). High accuracy was obtained for the detection of the team in possession, although a couple of questionable detection results were identified. The extracted data of all other events could be compared to the actual values at the end of each game and showed reasonably accurate values throughout the entire dataset, although inaccuracies were obtained due to the sampling rate being low.

- *The predictor functions trained and tested on the GRF dataset are able to achieve highly accurate prediction performance.* The predictor functions trained and tested on the GRF dataset have rounded MAE values of $\{0.04, 0.03, 0.03, 0.02, 0.01\}$ in terms of the ED and rounded R-square values of $\{0.992, 0.993, 0.995, 0.996, 0.997\}$ for the horizons, $h = \{5, 10, 15, 30, 45\}$ respectively. The values show highly accurate predictions were made and prove that the MLP's were trained effectively, as all the predicted values are closely located along the fitted regression line.

- *The predictor functions trained and tested on the eFootball dataset are able to achieve great performance with highly accurate predictions, while also being robust against a wide range of opponents with different strengths and weaknesses.* The predictor functions have rounded MAE values of $\{0.02, 0.02, 0.01, 0.01, 0.01\}$ in terms of the ED and R-square values of $\{0.994, 0.995, 0.998, 0.998, 0.999\}$ for the horizons $h = \{5, 10, 15, 30, 45\}$ on the test set of the eFootball dataset, which shows that the overall prediction error is very low and that the models were able to fit the presented data almost perfectly. Furthermore, the predictor function show robust performance against different opponents, since the test set includes data of 15 different games with 15 different opponents, all with their own strengths and weaknesses.

- *The predictor functions trained on the eFootball dataset and tested on an additional dataset featuring a single game using an online modelling approach are able to achieve fairly high predictive performance on low horizons, while predictive accuracy decreases whenever horizons are chosen larger.* The prediction trajectories of low-horizon predictor functions were able to follow the tracks of the actual values to a reasonable accuracy, with rounded MAE values of $\{0.2, 0.1, 0.1\}$ and rounded R-square values of $\{0.88, 0.79, 0.47\}$ for the horizons $\{5, 10, 15\}$ respectively. However, evaluation of the predictor functions of larger horizons show decreasing performance, with MAE values of $\{0.1, 0.2\}$ and R-square values of $\{0.22, -2.2\}$, which indicate that predictions cannot be made accurately. Given the assumptions and limitations of the project, the values show that online predictions in terms of the ED can be made to a decent accuracy when the prediction horizon is not chosen further than 15 minutes.

- *Prediction performance decreases when the prediction horizon is chosen larger.* The prediction experiments have shown that the prediction performance decreases whenever the horizon gets larger, as more uncertainty is involved when dealing with far-off predictions. Furthermore, a difference in the nature of the predictions was identified. In the case of short-horizon predictions, the predictor functions are able to use the given information of the past 15 minutes to read the current situation in the game and, consequently, make reasonably accurate predictions. Conversely, in the case of long-term predictions, the predictor functions focus more on the identification of the superior team, while the influences of current attacking/defending plays are penalized. Since the MLP's were trained on different games with opponents of varying strength, the predictors functions will not always be able to correctly identify the superior team.

- *The datasets used to develop the predictor functions need to be split into chunks of data, where each chunk includes consecutive labelled examples over a given time interval, to create a dataset more suitable for testing purposes.* A suspicion of bias was identified that may have resulted from the random shuffling process during the creation of the labelled examples, which could have led to numerous labelled examples being in both the training set and the test set, thus creating biased results. A possible solution is to group consecutive temporal labelled examples into chunks and, subsequently, shuffle and divide the chunks into new datasets for training, validating, and testing.

- *The predictor functions developed in this work need to be trained and tested on a larger dataset, consisting of a higher amount of games and covering more within-game plays, such that unseen input data can be recognized better.* The prediction results on the limited amount of games has been promising. It will therefore be interesting to see whether the MLP's will still be able to generalize the data whenever a wider range of games is involved.

Chapter 8

Conclusions and Recommendations

This chapter summarizes the main conclusions of this work and discusses recommendations for future research. Section 8-1 will first restate and answer the research questions of this work and discuss its main conclusions. Second, section 8-2 will provide recommendations for future work.

8-1 Conclusions

The main research question as formulated in section 1-2 is restated and will be answered below, based on the findings of this work:

Research question: *How can accurate online predictions in football be made with spatio-temporal data using learning-based algorithms?*

This work developed Artificial Neural Network (ANN)-based predictor functions suitable for making online predictions during a football game through a frame-by-frame quantification of the Expected Danger (ED); a comprehensive metric defining a form of relative danger. Results have shown that the predictor functions have been able to generalize the simulated spatio-temporal data of a selected number of games and, subsequently, use the acquired knowledge of these games, a priori, to make accurate online predictions on unseen frames relying solely on data of the past 15 minutes. Different prediction horizons were tried and tested, namely horizons of 5, 10, 15, 30, and 45 minutes. Great prediction performance was achieved over all horizons, thereby showing robustness of the prediction model against a wide range of opponents with different strengths and weaknesses. The predictor functions were also tested on an additional dataset, consisting of a single game of data that does not originate from the same distribution as the training data, with the purpose of testing online performance and robustness to unknown opponents. Experiments on this dataset demonstrated that reasonably accurate predictions can be made online as long as the prediction horizon is not chosen much larger than 15 minutes, although prediction capabilities could be further improved by training on a larger and richer dataset. The ED framework of this work has shown that online predictions can be made on a team-level and has opened doors for further research within the field of online predictive modelling and decision-making in football.

Similarly, the sub-questions of section 1-2 are restated and answered:

Research sub-question 1: *How can a theoretical, descriptive model describing the course of a football match be created?*

A descriptive model was made using a top-down modelling approach, where a general model was abstracted into a Markov Decision Process (MDP) model of reduced complexity. The MDP model has been able to mathematically formulate the complex dynamics of the game, while also enabling the creation of the prediction framework used throughout this work. Furthermore, the development of this MDP model has been of great importance in its ability to conceptualize the system and comprehend the complexity of the evolution of a football match. This comprehension of the system proved to be especially useful for the identification of problems during the creation of the black-box prediction models, as the models do not provide any additional comprehensible information about the system. The MDP model was therefore a crucial component of this research.

Research sub-question 2: *How can a representative, realistic dataset of football data be obtained, suitable for training and evaluating a learning-based algorithm?*

No suitable datasets or data sources within the field of real-life football were found meeting the necessary requirements for the training of a learning-based algorithm. Therefore, a decision was made to rely solely on simulation data, which allows the creation of large dataset and has the advantage of creating controlled experiments. Two novel datasets of both tracking- and event data were created using two separate football simulators, namely the relatively basic Google Research Football (GRF) simulator and the advanced eFootball simulator. The GRF dataset consists of 56 games and was extracted directly from the source code. It includes: detailed information about the positions, orientations and velocities of the players and ball; information about in-game events, such as penalties, free-kicks, goals, throw-ins, and goal-kicks; and some additional data on matters such as the fitness levels, yellow and red cards, the score, and the time.
The eFootball dataset was acquired using the data extraction algorithm pipeline as designed in this work, which is based on digital image processing techniques. The eFootball dataset consists of 16 games of data. It includes: the coordinate data of all players and ball over time; the team in possession; event data, such as the goals, touches of the ball, balls won, passes, and shots on target; and some additional data on the player positions, the formations, and the time. Both datasets are of larger size than any other comparable publicly available football dataset. They will both be made publicly available to incentivize further research within this field.

8-2 Recommendations

This section discusses the recommendations for future research. The recommendations have been divided into four different area's, namely: improvements on the current prediction pipeline, the quantification of online prediction uncertainty, decision-making, and bridging the domain gap between simulation data and real data. This is also the suggested order for future work.

The following recommendations are suggested to improve the current prediction pipeline:

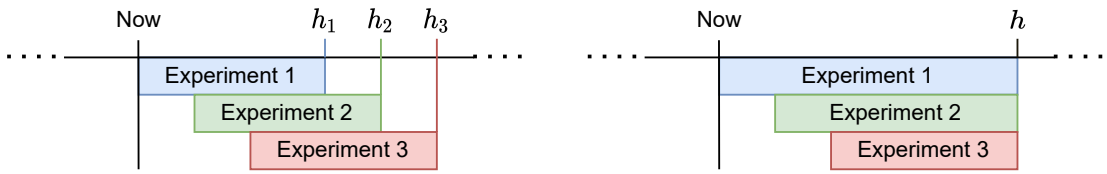
- **Improvements of the test set:** Two approaches can be taken to increase the differences between the test set and the training set. The first approach involves the bundling of successive labelled examples over a certain time interval into chunks, such that each chunk contains a

consecutive temporal sequence of states, which in turn represents a unique within-game play. As a result, the prediction model will be tested on unseen scenarios, thereby creating an unbiased evaluation of the ED. Another approach can be taken by simulating an additional number of games, extract the data, and use the extracted data for testing only. This way, the prediction model will certainly be tested on examples it has never seen before.

- **More specialized neural networks:** The usage of Recurrent Neural Networks (RNN's), which are specialized in sequence data such as spatio-temporal sequences, can lead to improvements in predictions and allows the learning algorithm to identify the time-dependencies of the states much easier. Instead of feeding an input vector of concatenated states to the network, the input of the network would now be a matrix where states are positioned in a side-by-side structure. Sequential models thereby allow the possibility of including a much larger amount of states into the input of the network, thus increasing the predictive potential of the algorithm, as the network is given much more information about the past spatio-temporal evolution of the game.

The following recommendations are relevant to the quantification of online prediction uncertainty:

- **Prediction interval experimentation:** A quantification of online prediction uncertainty can be made by experimenting with different prediction horizons. Two proposals will be discussed briefly in the following.
  The first approach involves experiments using a fixed-time interval and a varying horizon, as illustrated in Figure 8-1a. Given that the sampling frequency is chosen constant, a fixed-time interval will imply that the sum of rewards will consist of the same amount of values. This means that distributions and prediction metrics of the different horizons $h = \{h_1, h_2, h_3, \ldots\}$ can be compared better with each other. Metrics, such as the Mean Absolute Error (MAE) but also the variances of the distributions, can then be compared 1:1 with each other to quantify and evaluate how much uncertainty is involved in predictor functions of varying horizons.
  Another approach is to test how the initial starting point of the prediction interval influences the ED at time h, as illustrated in Figure 8-1b. As concluded in this work, short-term online values for the ED can be predicted more accurately than far-off values. However, this work has not attempted to quantify the extent of the uncertainty when short-term predictions are purposely left out of the computation of the ED. This experiment leaves room to do so.



(a) Fixed time interval with receding horizon. **(b)** Varying time intervals with fixed horizon.

Figure 8-1: Different options for prediction horizon experimentation for future work.

The following recommendations are made regarding the creation of a within-game decision-making model, where the coach or manager is now allowed to change its actions (e.g. the formation of the team) during the game. Two solution proposals were constructed for the decision-making model, which both involve an increased action space of the MDP model, such that $|A| > 1$. Both proposals build on the existing architecture of the prediction model of this work. Both proposals are provided as case studies in Appendix C and were not practically implemented due to the time limitations of the project. A short summary of the proposals of the two decision-making models is given in the following:

- **Decision-making model proposal 1:** The first decision-making model proposal adds an extra *input* to the existing Multilayer Perceptron (MLP) prediction model, as illustrated in Figure 8-2. The model will be trained with labelled examples that cover states of the game within time interval $[t, t+h]$, as well as the action $a \in A$ taken at time t. An important limitation of this model is that the action space is constrained in time in a way that actions can only be taken at certain pre-defined moments in the game.
  After training, the updated predictor function $ED : \tilde{S} \times A \rightarrow \mathbb{R}$ should then be able to compute the ED in a given state for all actions in $a \in A$. The action corresponding to the highest value for the ED will then be selected as the optimal action. An advantage of this model is that it can be created with relatively few adjustments to the MLP model. A disadvantage is that a lot of extra labelled examples need to be generated for training such that the MLP can attempt to learn the new state transition probabilities that exist when an action is taken in any given state. If the reader is interested, a more thorough explanation and analysis of this proposal is given in Appendix C-0-1.

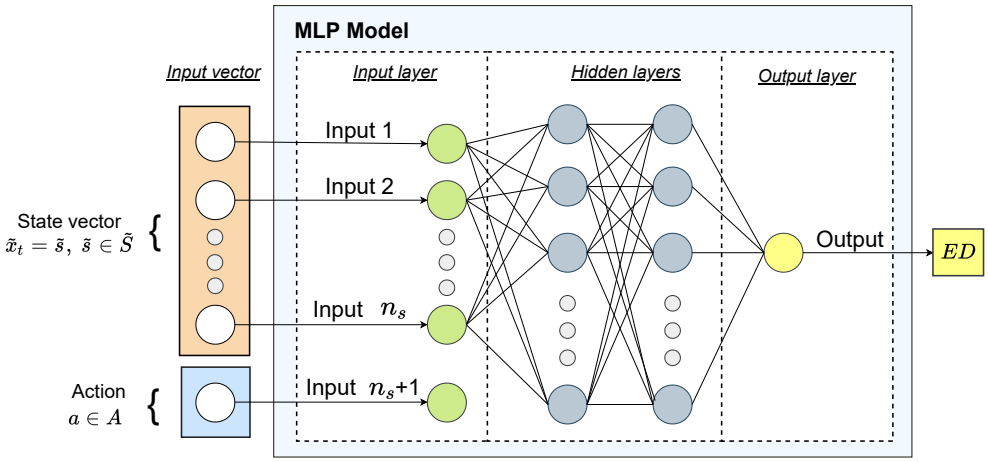


Figure 8-2: Solution proposal 1: extra input to the MLP model.

- **Decision-making model proposal 2:** The second decision-making model proposal adds an extra *output* to the existing MLP prediction model, where each output refers to the ED when a single action $a \in A$ is taken at time t, as illustrated in Figure 8-3. The rest of the network's architecture is kept the same. The model will be trained using a different training technique than applied in this work that is based on the work of Mnih et al. (2015) [93] about human-level control through Reinforcement Learning (RL), who were able to train an agent using RL to make it successfully execute challenging tasks within 2600 Atari

games. The training method consists of grouping successive labelled examples into chunks, where the examples in each chunk consist of only a single action taken, denoted as c_i^a. A chunk of examples where only the action $a = 433$ is taken might look as follows: $c_1^{a=433} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_{20}, y_{20})\}$. The MLP model will then be trained for each individual chunk, however, only the connections in the MLP that are connected to the output value of the action a of that particular chunk are kept intact. The architecture of the model during training will therefore be the same as the prediction model, and thus only have a single output value. The difference is that this output gate switches based on the action of each chunk of training data. This method resembles transfer learning techniques, however, instead of transferring the hyperparameters, the model now transfers both the hyperparameters and the parameters after training a model on each chunk.
After training, the weights of all output gates will be restored such that the amount of outputs is equal to the size of the action space. The updated predictor function $ED : \tilde{S} \rightarrow \mathbb{R}^{n_a}$ should now be able to compute different ED values corresponding to different actions taken at time t. The coach is then encouraged to choose the action corresponding to the highest ED. The disadvantage of this model is that it involves more complexity and will therefore be harder to design than the model of proposal 1. The main advantage is that the model is not dependent on the creation of an extremely large amount of labelled examples, and has the potential to make accurate predictions over different actions using only a relatively small amount of data. Solution proposal 2 is therefore preferred by the author. If the reader is interested, a more thorough explanation of this proposal is given in Appendix C-0-2, which also elaborates on the steps that must be taken to design the proposed model.

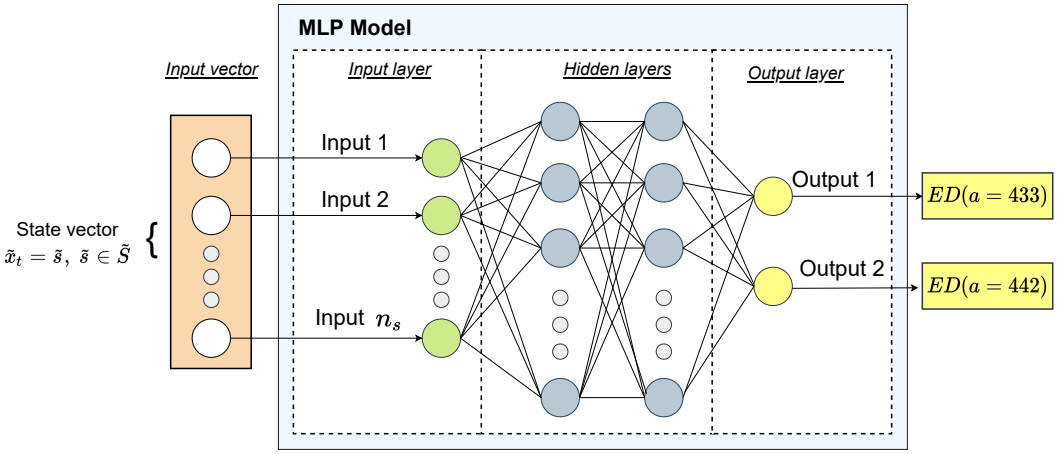


Figure 8-3: Solution proposal 2: extra output to MLP model.

The following recommendations can be linked to the ultimate goal of bridging the domain gap between simulation data and real data:

- **Usage of real football data:** The ANN-based predictor functions have been trained on simulation data of two different football simulators. The predictor functions were found to be robust against opponents of different playing styles and varying strengths. However, to adequately validate the performance of the predictions on real-life football, one must address the domain gap between simulated data and real data, and use real football data to make predictions. The prediction pipeline of this work can be used for these predictions.

Bibliography

[1] Michael Lewis. *Moneyball*. W.W. Norton, 2004.

[2] Hector Ruiz et al. ""The Leicester city fairytale?": Utilizing new soccer analytics tools to compare performance in the 15/16 & 16/17 EPL Seasons". In: *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* Part F1296 (2017), pp. 1991–2000.

[3] Kaan Koseler and Matthew Stephan. "Machine Learning Applications in Baseball: A Systematic Literature Review". In: *Applied Artificial Intelligence* 31.9-10 (2017), pp. 745–763. ISSN: 10876545. DOI: 10.1080/08839514.2018.1442991. URL: https://doi.org/10.1080/08839514.2018.1442991.

[4] Ronald Yurko, Samuel Ventura, and Maksim Horowitz. "NflWAR: A reproducible method for offensive player evaluation in football". In: *Journal of Quantitative Analysis in Sports* 15.3 (2019), pp. 163–183. ISSN: 15590410. DOI: 10.1515/jqas-2018-0010.

[5] Ronald Yurko et al. "Going deep: Models for continuous-time within-play valuation of game outcomes in American football with tracking data". In: *Journal of Quantitative Analysis in Sports* 16.2 (2020), pp. 163–182. ISSN: 15590410. DOI: 10.1515/jqas-2019-0056.

[6] Dragan Miljković et al. "The use of data mining for basketball matches outcomes prediction". In: *SIISY 2010 - 8th IEEE International Symposium on Intelligent Systems and Informatics* (2010), pp. 309–312. DOI: 10.1109/SISY.2010.5647440.

[7] Rahul Baboota and Harleen Kaur. "Predictive analysis and modelling football results using machine learning approach for English Premier League". In: *International Journal of Forecasting* 35.2 (2019), pp. 741–755. ISSN: 01692070. DOI: 10.1016/j.ijforecast.2018.01.003. URL: https://doi.org/10.1016/j.ijforecast.2018.01.003.

[8] Shuo Guan and Xiaochen Wang. "Optimization analysis of football match prediction model based on neural network". In: *Neural Computing and Applications* 0123456789 (2021). ISSN: 14333058. DOI: 10.1007/s00521-021-05930-x. URL: https://doi.org/10.1007/s00521-021-05930-x.

[9] O Balogun. "Artificial Neural Network Approach to Football Score Prediction". In: (2019), pp. 1–5. DOI: 10.28933/GJAI.

[10] Ryan Beal et al. “Optimising game tactics for football”. In: *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS*. Vol. 2020-May. May. 2020, pp. 141–149. ISBN: 9781450375184.

[11] Mat Herold et al. “Machine learning in men’s professional football: Current applications and future directions for improving attacking play”. In: *International Journal of Sports Science and Coaching* 14.6 (2019), pp. 798–817. ISSN: 2048397X. DOI: 10.1177/1747954119879350.

[12] Karl Tuyls et al. “Game plan: What AI can do for football, and what football can do for AI”. In: *Journal of Artificial Intelligence Research* 71 (2021), pp. 41–88. ISSN: 10769757. DOI: 10.1613/JAIR.1.12505.

[13] Javier Fernández et al. “Decomposing the Immeasurable Sport : A deep learning expected possession value framework for soccer”. In: (2019), pp. 1–20.

[14] Ryan Beal, Timothy J. Norman, and Sarvapali D. Ramchurn. “Artificial intelligence for team sports: A survey”. In: *Knowledge Engineering Review* 34 (2019), pp. 1–37.

[15] Dan Cervone et al. “POINTWISE: Predicting Points and Valuing Decisions in Real Time with NBA Optical Tracking Data”. In: *SLOAN Sports Analytics Conference* (2014), pp. 1–9. URL: http://dl.frz.ir/FREE/papers-we-love/sports_analytics/2014-ssac-pointwise-predicting-points-and-valuing-decisions-in-real-time.pdf.

[16] João Gustavo Claudino et al. “Current Approaches to the Use of Artificial Intelligence for Injury Risk Assessment and Performance Prediction in Team Sports: a Systematic Review”. In: *Sports Medicine* 5.1 (2019). ISSN: 21989761. DOI: 10.1186/s40798-019-0202-3.

[17] Mat Herold et al. “Attacking Key Performance Indicators in Soccer : Current Practice and Perceptions from the Elite to Youth Academy Level”. In: June 2020 (2021), pp. 158–169.

[18] Floris Goes et al. “Predicting match outcome in professional Dutch football using tactical performance metrics computed from position tracking data Predicting match outcome in professional Dutch soccer using tactical performance metrics computed from position tracking data”. In: (2019).

[19] C. Reep and B. Benjamin. “Skill and Chance in Association Football”. In: *Knowledge Engineering Review* 131.4 (1968), pp. 581–585. ISSN: 14698005. URL: https://pdfs.semanticscholar.org/a36c/694c79c3d38d19baf9d01a3677834289b340.pdf.

[20] Author I D Hill et al. “Association Football and Statistical Inference”. In: *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 23.2 (1974), pp. 203–208.

[21] Dimitris Karlis and Ioannis Ntzoufras. “Analysis of sports data by using bivariate Poisson models”. In: *Journal of the Royal Statistical Society: Series D (The Statistician)* 52.3 (2003), pp. 381–393.

[22] Gianluca Baio and Marta Blangiardo. “Bayesian hierarchical model for the prediction of football results”. In: *Journal of Applied Statistics* 37.2 (2010), pp. 253–264. ISSN: 02664763.

[23] Lars Magnus Hvattum and Halvard Arntzen. “Using ELO ratings for match result prediction in association football”. In: *International Journal of Forecasting* 26.3 (2010), pp. 460–470. URL: http://dx.doi.org/10.1016/j.ijforecast.2009.10.002.

[24] P O Donoghue et al. “Part II: Game activity and analysis”. In: *Journal of Sports Sciences* 22.6 (2004), pp. 500–520. ISSN: 0264-0414.

[25] Wojciech Sirko et al. “Continental-Scale Building Detection from High Resolution Satellite Imagery”. In: (2021), pp. 1–15. URL: http://arxiv.org/abs/2107.12283.

[26] Daniel Memmert, Koen A.P.M. Lemmink, and Jaime Sampaio. “Current Approaches to Tactical Performance Analyses in Soccer Using Position Data”. In: *Sports Medicine* 47.1 (2017), pp. 1–10. ISSN: 11792035. DOI: 10.1007/s40279-016-0562-5.

[27] Koen A.P.M. Lemmink and Frencken. “Tactical performance analysis in invasion games: Perspectives from a dynamical system approach with examples from soccer”. In: *Routledge handbook of sports performance*. London: Routledge, 2013, pp. 89–100.

[28] F Walter, M Lames, and T McGarry. “Analysis of sports performance as a dynamic system by means of relative phase”. In: *International Journal of Computer Science in Sports* 6 (2007), pp. 35–41.

[29] SBH Olthof, WGP Frencken, and KAPM Lemmink. “The older, the wider: on-field tactical behavior of elite-standard youth soccer players in small-sided games”. In: *Human Movement Science* 41 (2015), pp. 92–102.

[30] WGP Frencken, KAPM Lemmink, and N Delleman. “Oscillations of centroid position and surface area of soccer teams in small-sided games”. In: *European Journal of Sport Sciences* 11 (2011), pp. 215–23.

[31] WGP Frencken et al. “Variability of inter team distance associated with match events in elite-standard soccer”. In: *Journal of Sports Sciences* 30 (2012), pp. 7–13.

[32] Mike D Hughes and Roger M Bartlett. “The use of performance indicators in performance analysis”. In: (2002).

[33] Brian Burke. “DeepQB: Deep Learning with Player Tracking to Quantify Quarterback Decision-Making & Performance”. In: *Sloan Analytics Conference* (2019), pp. 1–13.

[34] Daniel Link, Steffen Lang, and Philipp Seidenschwarz. “Real Time Quantification of Dangerousity in Football Using Spatiotemporal Tracking Data”. In: (2016), pp. 1–16. DOI: 10.1371/journal.pone.0168768.

[35] Martijn Wagenaar et al. “Using deep convolutional neural networks to predict goal-scoring opportunities in soccer”. In: *ICPRAM 2017 - Proceedings of the 6th International Conference on Pattern Recognition Applications and Methods* 2017-Janua.Icpram (2017), pp. 448–455. DOI: 10.5220/0006194804480455.

[36] Manuel Stein et al. “How to make sense of team sport data: From acquisition to data modeling and research aspects”. In: *Data* 2.1 (2017).

[37] Tomoki Imai et al. “Play recognition using spatio-temporal relationship of football tracking data”. In: *2017 10th International Conference on Mobile Computing and Ubiquitous Network, ICMU 2017*. Vol. 2018-Janua. 2018, pp. 1–2. ISBN: 9784907626310. DOI: 10.23919/ICMU.2017.8330091.

[38] Lia Morra et al. *Slicing and Dicing Soccer: Automatic Detection of Complex Events from Spatio-Temporal Data*. 2020, pp. 107–121. ISBN: 9783030503468. DOI: 10.1016/b978-0-12-396501-1.00009-1.

[39] Victor Khaustov and Maxim Mozgovoy. “Recognizing events in spatiotemporal soccer data”. In: *Applied Sciences (Switzerland)* 10.22 (2020), pp. 1–12. ISSN: 20763417. DOI: 10.3390/app10228046.

[40] Luca Pappalardo et al. “A public data set of spatio-temporal match events in soccer competitions”. In: *Scientific Data* 6.1 (2019), pp. 1–15. ISSN: 20524463. DOI: 10.1038/s41597-019-0247-7. URL: http://dx.doi.org/10.1038/s41597-019-0247-7.

[41] Jia Liu et al. “Automatic player detection , labeling and tracking in broadcast soccer video”. In: *Pattern Recognition Letters* 30.2 (2009), pp. 103–113. URL: http://dx.doi.org/10.1016/j.patrec.2008.02.011.

[42] Chengjun Cui. “Player Detection based on Support Vector Machine in Football Videos”. In: 14.2 (2018), pp. 309–319.

[43] Samuel Hurault, Coloma Ballester, and Gloria Haro. “Self-Supervised Small Soccer Player Detection and Tracking”. In: *MMSports 2020 - Proceedings of the 3rd International Workshop on Multimedia Content Analysis in Sports* (2020), pp. 9–18.

[44] Huda Dheyauldeen Najeeb. “Tracking Ball in Soccer Game Video using Extended Kalman Filter”. In: (2020), pp. 78–82.

[45] Svein Arne Pettersen et al. “Soccer video and player position dataset”. In: *Proceedings of the 5th ACM Multimedia Systems Conference, MMSys 2014* (2014), pp. 18–23. DOI: 10.1145/2557642.2563677.

[46] Christopher Mutschler, Holger Ziekow, and Zbigniew Jerzak. “The DEBS 2013 grand challenge”. In: *DEBS 2013 - Proceedings of the 7th ACM International Conference on Distributed Event-Based Systems* (2013), pp. 289–294. DOI: 10.1145/2488222.2488283.

[47] Nicols Cruz and Javier Ruiz-Del-Solar. “Closing the Simulation-to-Reality Gap using Generative Neural Networks: Training Object Detectors for Soccer Robotics in Simulation as a Case Study”. In: *Proceedings of the International Joint Conference on Neural Networks* (2020).

[48] Paul Gagniuc. *Markov Chains: From Theory to Implementation and Experimentation*. 2017, pp. 1–256.

[49] A Frigessi and B Heidergott. *Markov Chains*. 2011.

[50] Robert Gallager. “Markov Chains with Countably Infinite State Spaces”. In: *The Springer International Series in Engineering and Computer Science* 321 (1996), pp. 149–186.

[51] Robert Gallager. “Discrete Stochastic Processes: Finite-state Markov chains”. In: *Massachusetts Institute of Technology: MIT OpenCourseWare* (2021).

[52] S. S. Melnyk, O. V. Usatenko, and V. A. Yampol’skii. “Memory functions of the additive Markov chains: Applications to complex dynamic systems”. In: *Physica A: Statistical Mechanics and its Applications* 361.2 (2005), pp. 405–415. ISSN: 03784371. DOI: 10.1016/j.physa.2005.06.083.

[53] Adrian Raftery. “A Model for High-Order Markov Chains”. In: 47.3 (1983), pp. 528–539.

[54] Chelsea C. White and Douglas J. White. “Markov decision processes”. In: *European Journal of Operational Research* 39.1 (1989), pp. 1–16. ISSN: 03772217. DOI: 10.1016/0377-2217(89)90348-2.

[55] Martijn Van Otterlo and Marco Wiering. *Reinforcement Learning and Markov Decision Processes*. Vol. 3. 2008, pp. 8–18. DOI: 10.1007/978-3-642-27645-3{_}1.

[56] Tom Mitchell. *Machine Learning*. McGraw-Hill, 1997, pp. 144–152.

[57] John Koza et al. *Automated Design of Both the Topology and Sizing of Analog Electrical Circuits Using Genetic Programming*. Springer, 1996, pp. 151–170.

[58] Ryo Matsumura et al. "Learning based industrial bin-picking trained with approximate physics simulator". In: *Advances in Intelligent Systems and Computing* 867 (1943), pp. 786–798. ISSN: 21945357. DOI: 10.1007/978-3-030-01370-7{_}61.

[59] Anthony C. Constantinou, Norman E. Fenton, and Martin Neil. "Pi-football: A Bayesian network model for forecasting Association Football match outcomes". In: *Knowledge-Based Systems* 36.December (2012), pp. 322–339. ISSN: 09507051.

[60] Andrew Ng. "Deep Learning Specialization". In: *Coursera* (2021).

[61] Zhiyuan Li and Sanjeev Arora. "An Exponential Learning Rate Schedule for Deep Learning". In: (2019), pp. 1–29. URL: http://arxiv.org/abs/1910.07454.

[62] Eyal Even-Dar and Yishay Mansour. "Learning rates for Q-learning". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 2111 (2001), pp. 589–604.

[63] Diederik P. Kingma and Jimmy Lei Ba. "Adam: A method for stochastic optimization". In: *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings* (2015), pp. 1–15.

[64] Popescu Marius-Constantin et al. "Multilayer perceptron and neural networks". In: *WSEAS Transactions on Circuits and Systems* 8.7 (2009), pp. 579–588. ISSN: 11092734.

[65] Frank Rosenblatt. *The perceptron : a theory of statistical separability in cognitive systems*. Buffalo, N.Y.: Cornell Aeronautical Laboratory, 1958.

[66] Ahmad Alwosheel, Sander van Cranenburgh, and Caspar G. Chorus. "Is your dataset big enough? Sample size requirements when using artificial neural networks for discrete choice analysis". In: *Journal of Choice Modelling* 28 (2018), pp. 167–182. ISSN: 17555345. DOI: 10.1016/j.jocm.2018.07.002. URL: https://doi.org/10.1016/j.jocm.2018.07.002.

[67] Miriam Steurer, Robert J. Hill, and Norbert Pfeifer. "Metrics for evaluating the performance of machine learning based automated valuation models". In: *Journal of Property Research* 38.2 (2021), pp. 99–129. ISSN: 14664453. DOI: 10.1080/09599916.2020.1858937. URL: https://doi.org/10.1080/09599916.2020.1858937.

[68] Pragnan Chakravorty. "What is a Signal?" In: *IEEE Signal Processing Magazine* 35.5 (2018), pp. 175–177.

[69] P.V.C. Hough. *A Method and Means for Recognizing Complex Patterns*. 1962.

[70] HK Yuen et al. "Comparative study of Hough Transform methods for circle finding". In: *Image and Vision Computing* 8.1 (1990), pp. 71–77. ISSN: 02628856. DOI: 10.1016/0262-8856(90)90059-E.

[71] John Canny. "A Computational Approach to Edge Detection". In: *IEEE 11th International Symposium on Biomedical Imaging (ISBI)* PAMI-8.6 (2006), pp. 679–698.

[72] N Kanopoulos, N Vasanthavada, and R.L. Baker. "Design of an image edge detection filter using the Sobel operator". In: *IEEE Transactions on Signal Processing* 23.2 (1991), pp. 358–367.

[73] R Brunelli. *Template Matching Techniques in Computer Vision: Theory and Practice*. Wiley, 2009.

[74] Wai Ching et al. *Markov chains*. Vol. 236. 5345. Springer, 2006, p. 291. ISBN: 9781461463115. DOI: 10.1038/236291a0.

[75] Carlos Cotta et al. "A network analysis of the 2010 FIFA world cup champion team play". In: *Journal of Systems Science and Complexity* 26.1 (2011), pp. 21–42. ISSN: 10096124. DOI: 10.1007/s11424-013-2291-2.

[76] Takuma Narizuka, Ken Yamamoto, and Yoshihiro Yamazaki. "Statistical properties of position-dependent ball-passing networks in football games". In: *Physica A: Statistical Mechanics and its Applications* 412 (2014), pp. 157–168. ISSN: 03784371. DOI: 10.1016/j.physa.2014.06.037. URL: http://dx.doi.org/10.1016/j.physa.2014.06.037.

[77] E. Arriaza-Ardiles et al. "Applying graphs and complex networks to football metric interpretation". In: *Human Movement Science* 57.September 2016 (2018), pp. 236–243. ISSN: 18727646. DOI: 10.1016/j.humov.2017.08.022. URL: https://doi.org/10.1016/j.humov.2017.08.022.

[78] José Gama et al. "Network analysis and intra-team activity in attacking phases of professional football". In: *International Journal of Performance Analysis in Sport* 14.3 (2014), pp. 692–708. ISSN: 14748185. DOI: 10.1080/24748668.2014.11868752.

[79] Martin Marsden. "Cubic Spline Interpolation of Continuous Functions". In: *Journal of Approximation Theory* 111.10 (1974), pp. 103–111.

[80] Abien Fred Agarap. "Deep Learning using Rectified Linear Units (ReLU)". In: *arXiv preprint arXiv:1803.08375* 1 (2018), pp. 2–8. URL: http://arxiv.org/abs/1803.08375.

[81] Kaiming He et al. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification". In: *Proceedings of the IEEE International Conference on Computer Vision* 2015 Inter (2015), pp. 1026–1034. ISSN: 15505499. DOI: 10.1109/ICCV.2015.123.

[82] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning (Adaptive Computation and Machine Learning series)*. The MIT Press, 2016, p. 434.

[83] Kurt Hornik. "Approximation capabilities of multilayer feedforward networks". In: *Neural Networks* 4.2 (1991), pp. 251–257. ISSN: 08936080. DOI: 10.1016/0893-6080(91)90009-T.

[84] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. "A Fast Learning Algorithm for Deep Belief Nets Geoffrey". In: *International Journal of Environmental Science and Development* 1554 (2010), pp. 341–346. ISSN: 20100264. DOI: 10.7763/ijesd.2010.v1.67.

[85] Jeff Heaton. *Introduction to Neural Networks for Java, 2nd Edition*. Heaton Research, 2008, pp. 214–218.

[86] Kostadin Yotov, Emil Hadzhikolev, and Stanka Hadzhikoleva. "Determining the Number of Neurons in Artificial Neural Networks for Approximation, Trained with Algorithms Using the Jacobi Matrix". In: *TEM Journal* 9.4 (2020), pp. 1320–1329. ISSN: 22178333. DOI: 10.18421/TEM94-02.

[87] Thomas Kurbiel and Shahrzad Khaleghian. "Training of Deep Neural Networks based on Distance Measures using RMSProp". In: *arXiv* (2017), pp. 1–6. URL: http://arxiv.org/abs/1708.01911.

[88] Ian Goodfellow and Yoshua Bengio. *Deep Learning (Adaptive Computation and Machine Learning series)*. 2016, p. 201.

[89] Nornadiah Mohd Razali and Yap Bee Wah. "Power comparisons of Shapiro-Wilk, Kolmogorov-Smirnov, Lilliefors and Anderson-Darling tests". In: *Journal of Statistical Modeling and Analytics* 2.1 (2011), pp. 13–14.

[90] Debashis Paul and Alexander Aue. "Random matrix theory in statistics: A review". In: *Journal of Statistical Planning and Inference* 150 (2014), pp. 1–29. ISSN: 03783758. DOI: 10.1016/j.jspi.2013.09.005.

[91] Vladimir Ivanovich Smirnov. *Kolmogorov–Smirnov Test*. 1939. DOI: 10.1007/978-0-387-32833-1{_}210.

[92] Michel Dekking. *A Modern Introduction to Probability and Statistics*. Springer, 2005, pp. 181–190. ISBN: 9781852338961.

[93] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518 (2015), pp. 529–533.

[94] Javier M. Buldú et al. "Using network science to analyse football passing networks: Dynamics, space, time, and the multilayer nature of the game". In: *Frontiers in Psychology* 9.OCT (2018), pp. 1–5. ISSN: 16641078. DOI: 10.3389/fpsyg.2018.01900.

[95] Javier López Peña and Hugo Touchette. "A network theory analysis of football strategies". In: (2012), pp. 1–6. URL: http://arxiv.org/abs/1206.6904.

[96] Bruno Gonçalves et al. "Exploring Team Passing Networks and Player Movement Dynamics in Youth Association Football". In: (2017). DOI: 10.1371/journal.pone.0171156.

[97] Jordi Duch, Joshua S. Waitzman, and Luís A. Nunes Amaral. "Quantifying the performance of individual players in a team activity". In: *PLoS ONE* 5.6 (2010), pp. 1–7. ISSN: 19326203.

[98] Florian Korte et al. "Play-by-play network analysis in football". In: *Frontiers in Psychology* 10.JULY (2019), pp. 1–10. ISSN: 16641078. DOI: 10.3389/fpsyg.2019.01738.

[99] Luca Pappalardo. "PlayeRank : Data-driven Performance Evaluation and Player Ranking in Soccer via a Machine Learning Approach". In: 10.5 (2019).

[100] Filipe Manuel Clemente et al. "Section III-Sports Training Using Network Metrics in Soccer: A Macro-Analysis". In: *Journal of Human Kinetics* 45 (2015), pp. 123–134. DOI: 10.1515/hukin-2015-0013. URL: http://www.johk.pl.

[101] Paolo Cintia et al. "The harsh rule of the goals: Data-driven performance indicators for football teams". In: *Proceedings of the 2015 IEEE International Conference on Data Science and Advanced Analytics, DSAA 2015* (2015). DOI: 10.1109/DSAA.2015.7344823.

[102] Tiago J Pina, Ana Paulo, and Duarte Araújo. "Network Characteristics of Successful Performance in Association Football . A Study on the UEFA Champions League". In: 8.July (2017). DOI: 10.3389/fpsyg.2017.01173.

[103] Paolo Cintia, Salvatore Rinzivillo, and Luca Pappalardo. "Football_Network_Teams". In: *Machine Learning and Data Mining for Sports Analytics workshop (MLSA'15)* (2015), pp. 1–9.

[104] Alina Bialkowski et al. "" Win at Home and Draw Away ": Automatic Formation Analysis Highlighting the Differences in Home and Away Team Behaviors Presented by : common y he d be ef ha eams shou d a m o " w n he r home games and draw he r away ones " W h he a d of a who e season p". In: (2014).

[105] Genki Ichinose, Tomohiro Tsuchiya, and Shunsuke Watanabe. "Robustness of football passing networks against continuous node and link removals". In: *Physics and Society* (2020), pp. 1–18. URL: http://arxiv.org/abs/2003.13465.

[106] Ton van den Boom and Manuel Mazo. *Modeling of Dynamical Systems*. TU Delft, 2021, pp. 7–24.

Part I

Appendices

Appendix A

Heritage

A-1 Research by Big Companies

Big commercial companies are also largely invested in data analysis and analytics in football. However, their data, studies and products are mostly not accessible for the wide public and if they are, a high price often has to be paid. Nonetheless, it is important to know what tools, techniques and methods have already been created in the industry, even if the theory behind them is not fully accessible. A short review about metrics and tools from big companies will be given, in which a particular focus will be put on real-time and ML-based metrics and tools. A loose classification was made of the football data companies. First, the largest data providing companies will be addressed, who offer ML-based metrics or tools. Second, companies who offer tools to gather tracking data and transfer real-time metrics will be covered. Third, companies offering video analysis tools will be covered.

A-1-1 Datasets and metrics based on machine learning techniques

Four companies were found offering large quantities of advanced football data and various advanced ML-based metrics. These companies are: Opta Sports, Statsperform, Sportlogiq, and Panini-Digital. All of them do not go into much detail about their approaches, other than stating they are 'AI-powered' or 'ML-based'. Opta Sports is the largest sports data company in the world, operating in more than 40 countries and analyzing more than 110,000 events on an annual basis. Opta Sports "are the only sports data business that collects and distributes full, time-stamped, contextual data live, featuring complete x,y-coordinates (as well as z-coordinates where applicable, such as shots in football), and a granularity of event type unique amongst data providers" [1]. Besides advanced data sets, they provide three advanced Machine Learning (ML) metrics to their customers, which can be used for evaluation and analysis purposes [2]. Two of these metrics are expected goals and expected assists. Expected goals measures the quality of a shot based on various variables. Each

[1] https://www.optasports.com/about/the-opta-difference/

[2] https://www.optasports.com/services/analytics/advanced-metrics/

shot gets a value in between 0 and 1 which cumulatively adds up to a player's or team's chances of scoring, indicating how many goals a player or team should have scored. Expected assists is analogous to expected goals, but goals are replaced with assists. Another metric is able to define different sequences of play in which events such as passes, dribbles, and shots are identified. More advanced metrics are offered by Opta's sister company Statsperform. Statsperform merged with Opta in 2019, but still have their own research departments. They provide advanced ML-based metrics to establish performance trends for teams and players, although not going into much detail about them. In addition, they provide a very interesting algorithm which allows live data insights to assist with in-game tactical decision-making on a minute-by-minute level. [3]. Other sports analytics company offering ML-based metrics are: Sportlogiq, who offer football tracking data and various ML-based metrics, although not going into much detail about them [4]; and PaniniDigital, who offers post-match statistical analyses and ML-based analytics through tracking data. In recent years, data sets have gotten increasingly advanced due to the rise in accurate and advanced GPS trackers. One of world leaders in providing GPS trackers for sports is Statssport. They have created a wearable tracking device ,the Apex Pro Series Pod, able to transfer 260+ metrics real-time [5]. In addition, they created the Apex Coach Series, which simplifies their metrics into a comprehensible format for coaches and allows user-friendly player and team performance analysis. Scisports is another provider of football data, including tracking data and various metrics. They offer advanced statistics and analytics to support performances and decisions. Their newest state-of-art analysis tool Inmotio allows real-time analysis and insights to assist in decision-making [6]. Other companies providing tracking data are: Catapult, who offers tracking data from both shoes, allowing real-time transfer of data and different metrics; and KINEXON, offering automated tracking data and basic metrics.

In order to make data analysis tools attractive to coaches and teams, it is common to use video analysis tools. Wyscout is a large company providing databases regarding performances of players and teams for coaches, keeping track of matches in more than 250 worldwide competitions [7]. Their main product is a video analysis tools providing video data and various metrics to help coach with analysing their team and other teams in a graphical way. Other companies involved in video analysis tools are: Second Spectrum, who offers real-time video augmentations with statistics; and Metrica Sports, who offer automated event detection, simple metrics, and video augmentations.

A-1-2 Conclusion

Various companies (Opta Sports, Statsperform, Sportlogiq, PaniniDigital, Scisports) provide detailed and advanced data sets as products. These data sets can be obtained by making use of state-of-art GPS trackers from various companies (Statssport, Scisports, Catapult, KINEXON), which are able to accurately transfer x,y,z-coordinates real-time, as well as a large variety of real-time metrics. Then, using state-of-art video analysis tools offered by numerous companies (Wyscout, Second Spectrum, Metrica Sports, Scisports), coaches are able to efficiently analyse their players and teams in between matches for effective offline tactical decision-making.

[3] https://www.statsperform.com/team-performance/football-performance/match-analysis/
[4] https://sportlogiq.com/en/technology
[5] https://statsports.com/apex/
[6] https://inmotio.eu/analysis-software/
[7] https://wyscout.com/coaches/

A-2 Key Performance Indicators

This section will show a series of Key Performance Indicators (KPI's), which were all reviewed before creating the Expected Danger (ED) metric. Some were used as inspiration. Three types of topological scales were identified, which were used to classify the different KPI's [94]. These scales are as follows: *the microscale*, which only includes analysis at the level of individual players; the *the mesoscale*, which describes small motifs between interactions of 3 to 4 players; and *the macroscale*, which considers the network as a whole. The findings are summarized in Table A-1.

Metric	Method	Description	Scale	References
Degree centrality	Graph theory	Calculates number of passes of each player	Microscale	Cotta et al. [75]
Eigenvector centrality	Graph theory	Measures a players' quantitative passing performance, while also giving consideration to its neighbors passing performance	Microscale	Cotta et al.[75]
Closeness centrality	Graph theory	Calculates the minimum number of steps to reach players	Microscale	López-Peña and Touchette [95]; Gonçalves et al. [96]
Betweenness centrality	Graph theory	A measure which considers how many times a player acts as a necessary bridge between other players.	Microscale	Duch et al. [97]; Gonçalves et al.[96]; Arriaza-Ardiles et al. 2018
Flow betweenness centrality	Graph theory	Betweenness centrality, but giving more emphasis to the sequential pattern of passing	Microscale	Korte et al. [98]
PlayeRank	Machine Learning	An advanced multi-dimensional performance indicator which incorporates role-aware evaluation of individual players using a ML approach.	Microscale	Pappalardo et al. [99]
Clustering coefficient	Graph theory	Measures the number of neighbors of a player that also have passed the ball between them	Microscale	López-Peña and Touchette [95]
Heterogeneity coefficient	Statistical	Relates the number of passes between players to sub-community existence, which hinders team performance.	Mesoscale	Clemente et al. [100]
Pass overabundancy index	Statistical	Relates overabundance of certain passes of subcommunities to team success	Mesoscale	López Peña and Sánchez Navarro [95]
H-indicator	Statistical	A set of 6 passing-related performance indicators summarized into one index	Macroscale	Cintia et al. [101].
Degree variability	Graph theory	A measure describing how much the degree centralities of players variate.	Macroscale	Pina et al. [102]
Average degree	Graph theory	Overall average of players' degree	Macroscale	Cintial et al. [103]
Network centroid	Euclidean metrics	Performance indicator describing the importance of the position of the centroid of the network	Macroscale	Bialkowski et al. [104]
Network robustness	Graph theory	Measure which considers the importance of individual players through analysis of link removal	Macroscale	Ichinose et al. [105]

Table A-1: Overview of candidate metrics for tactical performance analysis and predictive modeling.

Appendix B

Datasets

This appendix will explain how the Google Research Football (GRF) dataset has been created in section B-1. Subsequently, section B-2 will discuss parts of the eFootball dataset that were not included in the main part of the work.

B-1 The GRF Dataset

The first dataset was created using the open-source GRF simulator. The GRF simulator explores a new type of realistic gameplay in which they allow people to code their own players using an open-source simulator. The team encourages people to train agents using Reinforcement Learning (RL) techniques and let different teams compete with each other in a competition. The simulator can be used for realistic gameplay and gives options to test new tactics and ideas within real-life football. In addition, an extra option is provided in which the user can simulate a game between build-in AI's. The final dataset was created by simulating games using this build-in AI.

B-1-1 Internal mechanics of the build-in AI

The build-in AI behaves according to rule-based mechanics, which differentiates between an attacking and a defensive scenario. When attacking, the agent in possession of the ball is given a certain set of options, such as: dribbling forwards, passing backwards, shooting on target, etc. A rating is given to each option, including the position and velocity of the player and ball, the direction of the agent, the positions of its teammates, the predicted time before the opponent can pressure the ball, and its assigned role (e.g., attacking midfielder, keeper). The option with the highest rating is then executed by the agent at each time instance.
When defending, a single agent is designated to recapture the ball based on its proximity to the ball and its defensive position. Similar to the attacking scenario, the defensive agent then optimizes over a certain set of options (i.e. move towards the ball, slide tackle, move backwards to the goal), where each option is given a rating based on the state of the game. All agents, apart from the designated attacker on the attacking team and the designated defender on the defensive team,

will move according to more simplistic rule-based mechanics. For each individual agent, these rule-based mechanics are based on the current possession team, its the assigned role (e.g., keeper, left-winger), and the location of the ball.
No code lines were found defining different skill sets for agents at different positions. This gives the impression that all agents in the game are programmed with the same exact skill level. Examples of skills in football agents are: shot power, passing accuracy, and pace. Furthermore, changes in tactics, substitutions or formations can not be made before or during a match. Both teams have a fixed 1-4-3-3 formation (1 goalkeeper, 4 defenders, 4 midfielders, and 3 attackers) and tactics are fixed to a balanced playing style. The level of randomness involved in the dynamics of the game could not be retrieved from the code. However, unique plays and a large variety of different match outcomes do indicate there is at least some level of randomness involved. The full code of the build-in AI was can be found on Github [1].

B-1-2 Shortcomings and advantages of using the simulator for research

The internal mechanics of the game expose a number of shortcomings with using the Google Football simulator for research. Identifying the drawbacks of the simulator at an early stage is important as this already gives insights in potential flaws of the prediction results. The three main shortcomings with regards to goal of the project have been listed below.
First of all, both when attacking or defending, only a single designated player will move following more complex rule-based mechanics. The limited variability in the actions of all other players may result in a oversimplified version of a football game. Making accurate within-game predictions might therefore be relatively easy, as much of the subtle, complex dynamics of the game have gotten lost in the simulation process.
Second, it is unclear how much randomness is involved. However, the randomness implemented in the game will heavily influence the potential of within-game predictions. A simulator where random variables largely dictate game outcomes is much harder to predict than a simulator without any random influences. As a result, it will be hard, if not impossible, to link possible poor prediction accuracy's to either the randomness of the game or a failing prediction algorithm. Conversely, unreasonably high prediction accuracy's could be achieved as a result of the absence of randomness.
Third, finding optimal or sub-optimal tactics can only be realized by experimenting with various tactics during matches and observing outcomes. The inability of the simulator to customize the parameters of team tactics makes the simulator unfit for within-game decision-making, thereby limiting the potential for future work.
Nonetheless, the GRF simulator proved to be a great experimental tool for the initial phase of the work. It provides a huge amount of data, where a detailed, accurate description of the match is given at each moment in time. To the best knowledge of the author, no other publicly available source can be used to obtain such a large amount of detailed football data within such a short amount of time.

[1] `https://github.com/google-research/football/blob/master/third_party/gfootball_engine/src/onthepitch/AIsupport/AIfunctions.cpp`

B-1-3 Raw data

All the raw data from the simulator was extracted and transferred to a separate file. These raw observations can be observed in Table B-1. A distinction has been made in the nature of the states, depending on the cardinality of the state variable. Some states variables take values from a continuous set, in which $x_i \in \mathbb{R}^n$. Take for instance the coordinates of players, or the tiredness factor of players. Other state variables take values from finite sets $\{q_1, q_2, ...\}$, such as the vector denoting the amount of yellow cards or the score of the match. Another distinction has been made on the mechanism that drives the evolution of the state [106]. Time-driven states evolve at every tick of the clock, whereas event-driven states change at a given event within the environment. Furthermore, it must be noted that the data is simulated with a certain rate, which makes the system inherently discrete-time.

The raw state vector of the system can now be defined by concatenating the raw observations into a state vector of 190 values, or formally $x \in \mathbb{R}^n$ with n=190. The state vector can be used to define the general model.

Variable name	Description	Vector size	State type	Evolution Mechanism
$x_p(t)$	A vector containing the exact [x,y]-coordinates of the players of both teams.	44	Continuous	Time
$x_d(t)$	A vector containing the exact [x,y]-direction of the players of both teams.	44	Continuous	Time
$x_b(t)$	A vector containing the exact ball position and ball direction in [x,y,z] coordinates and the ball [x,y,z] rotation angles in radians.	9	Continuous	Time
$x_t(t)$	A vector containing the tiredness factor of the players of both teams $\{x_t \in \mathbb{R}^{22} \mid 0 \leq x_t \leq 1\}$.	22	Continuous.	Time
$z_o(t)$	A value denoting the team who owns the ball $z \in \{-1, 0, 1\}$ (no team, left team, right team).	1	Discrete	Event
$z_y(t)$	A vector denoting which players have yellow cards $z \in \{0, 1\}$	22	Discrete	Event
$z_a(t)$	A vector denoting which players are active $z_a \in \{0, 1\}$.	22	Discrete	Event
$z_m(t)$	An integer value denoting the current game mode $z_m \in \{0, 1, ..., 7\}$ (normal, kick-off, goal-kick, free-kick, corner, throw-in, penalty)	1	Discrete	Event
$z_r(t)$	A vector denoting the role of each player $z_r \in \{0, 1, ..., 9\}$ which correspond to goal keeper, centre back, left back, right back, defence midfield, central midfield, left midfield, right midfield, attack midfield respectively.	22	Discrete	Event
$z_s(t)$	A vector denoting the score of the match.	2	Discrete	Event
$z_e(t)$	An integer value denoting the amount of steps left until the end of the match (there is no extra time).	1	Discrete	Time

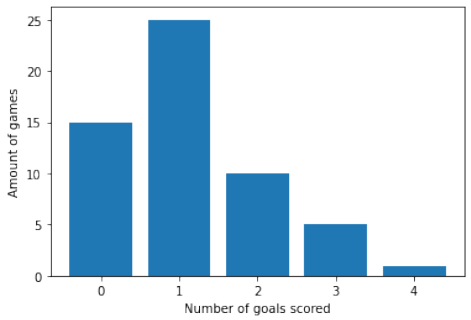
Table B-1: A classification of the raw observations/state of the Google Football simulator.

B-1-4 Creation of the dataset

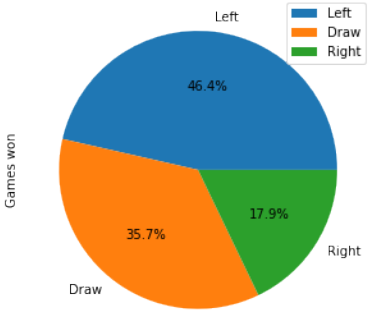
The simulator allows the option to graphically render matches such that the gameplay can be visualized. However, rendering does drastically increase simulation time and was therefore not used for creation of the final dataset. Using the build-in AI function, 58 games were simulated to be collected into the dataset. Each game consists of 3001 time steps, each with its own state vector describing the game. The creation of the dataset took approximately an hour. All games were simulated with: a fixed formation for both teams, a fixed playing style for both teams, no substitutions, no extra time, no significant quality differences between players, no significant quality differences between teams. The reduced variability of the data leads to easier generalization of the data compared to datasets from more advanced simulators. The size of the dataset was therefore deemed as sufficient as a starting point, but can be extended whenever needed. All the data was stored in a pickle file. The dataset will be made public to incentive research in this field.

B-1-5 Data validation

The data was checked for errors and flaws by means of basic data analysis and visualization of the data. First, it was checked whether the score outcomes of the matches played seemed reasonable. Teams should be able to score and win both over the course of 58 games. In most matches, only a single goal got scored, while zeros goals or two goals are common occurrences as well. The exact amount of goals scored in all games can be seen in Figure B-1a. An average of 1.1 goals per game was obtained, which is comparable to lower-table teams in the English Premier League [2]. The left team was able to win almost half the games, with a winning percentage of 46.4%, while the right team only won 17.9%, which seems a bit odd considering both teams are controlled by the same build-in AI. The distribution of the winning percentage of both teams is visualized in Figure B-1b.



(a) Histogram of goal distribution



(b) Chart of game winning percentages

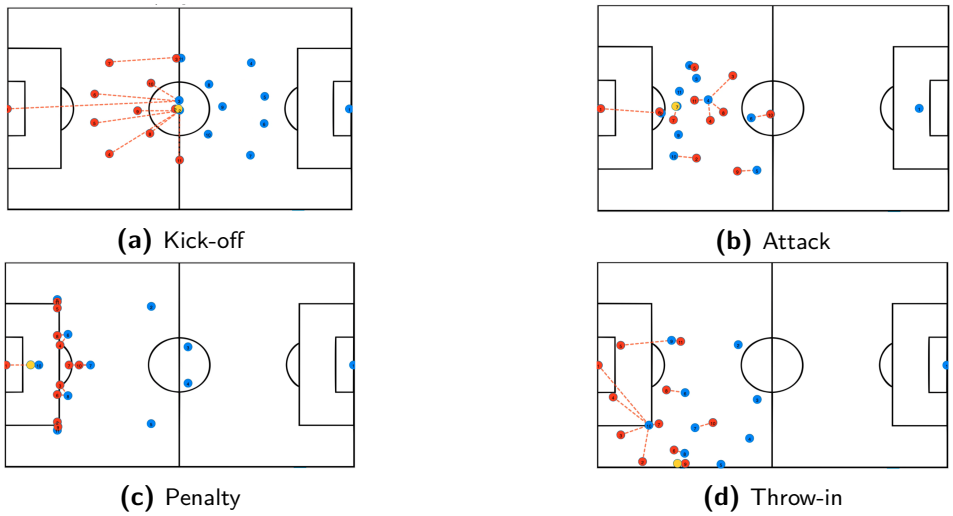
Figure B-1: Figure B-1a shows a histogram of the total amount of goals scored in each of the 58 games simulated. Figure B-1b shows a pie chart of the winning distribution of both teams.

Using the tracking data of all players and the ball over time, it was possible to recreate the match as an animation. The Bokeh library was used to visualize the data in Python. Different snapshots of a match can be seen in Figure B-2 and a video of the match was posted on Vimeo [3]. The animation

[2] `https://www.skysports.com/premier-league-table`. Date accessed: 20-05-2021
[3] `https://vimeo.com/manage/videos/594632118`

portrays the match accurately and shows all players and ball move smoothly across the pitch. A visual inspection shows no visible errors within the data, indicating the tracking data of all players and ball is accurate. Furthermore, the scores of the match also seem to be accurate as the score increases each time a goal is scored.

Other insights were also obtained from visual inspection of the data. First of all, the fixed 1-4-3-3 formations of both teams can be observed in Figure B-2a. Figure B-2b shows a snapshot of an attack by the blue team. By letting the animation play, it becomes obvious defenders are not acting in a smart way and attackers have trouble finding the right pass and shooting on target. Set pieces, like free kicks, penalties and throw-ins, occur after stoppages and result in a reset of players to fixed positions, as can be observed in Figure B-2c and Figure B-2d. Last, it can be observed that only single agents within a team are behaving in a more complex and realistic manner. The player who owns the ball will generally dribble while other players on its team are not actively participating. On the defensive side, the player closest to the ball will generally be the only player actively trying to regain possession of the ball.



(a) Kick-off **(b)** Attack **(c)** Penalty **(d)** Throw-in

Figure B-2: Different game scenario's in a match played by the red team on the left and the blue team on the right. The ball is colored in yellow.

Oddly enough, not all data was found to be accurate. The possession state variable, z_o, does not conclude who is in possession of the ball at all time instances. z_o can take on values from the set $\{-1, 0, 1\}$, where -1 indicates ball ownership of the left team, 1 indicates ball ownership of the right team, and 0 is undefined. A visual representation of the ownership of the ball over time shows the possession of the ball over time. Possession switches ownership rather rapidly and the ball is not in ownership of a team a significant proportion of the time. However, real life football usually only considers possession as a binary value; either the left team or the right team has possession. Therefore, an assumption was made that whenever ball ownership is undefined, it will be assumed the previous team in ownership of the ball will still be in ownership of the ball. This means that the set of possible values of ball ownership is decreased to $\{-1, 1\}$. This assumption is visualized in When compared to the video animation, the possession data does not appear to be accurate. It is clearly visible that the possession data does not correspond to the team possession within the video recording. Sometimes, possession is lost for an extended period of time, which is then not captured within the possession data. Therefore, it was chosen to discard z_o from the data accuracy

due to poor accuracy. Instead, a separate algorithm will be written to infer who is in possession of the ball.

Deduction of the player in possession

A couple of assumptions were made to deduce which player is in possession. Although these assumptions will not lead to perfect accuracy, reasonable results were obtained when compared to the animation.

Assumption 1. *Player A is in ownership of the ball when he is within 1.5 metres of the ball for at least 3 seconds.*

Assumption 2. *Player A will keep ownership of the ball until another player takes ownership of the ball.*

Assumption 3. *Player B will take ownership from player A whenever player A is not within 1.5 metres of the ball and player B is within 1.5 metres of the ball for at least 3 seconds.*

Following the above assumptions, an algorithm was written to infer possession.

B-2 The eFootball Dataset

This section will explain how the threshold was set for the template matching algorithm in Figure B-3 and present the detection percentages of the data extraction algorithm over all games in subsection B-2-2.

B-2-1 Setting suitable threshold for the template matching functions

Whenever the templates of players move over one another, the template behind cannot be detected anymore by the algorithm. In this case, the algorithm should output that the template has an unknown location. As a solution, a threshold was set for the accuracy of a match. Whenever this accuracy threshold is reached, the match will be accepted. An accuracy of 1.0 indicates a perfect match and 0.0 indicates no similarities. After a visual comparison of the matches of all templates within numerous image frames, a threshold was set at 0.97 (see Figure B-3).

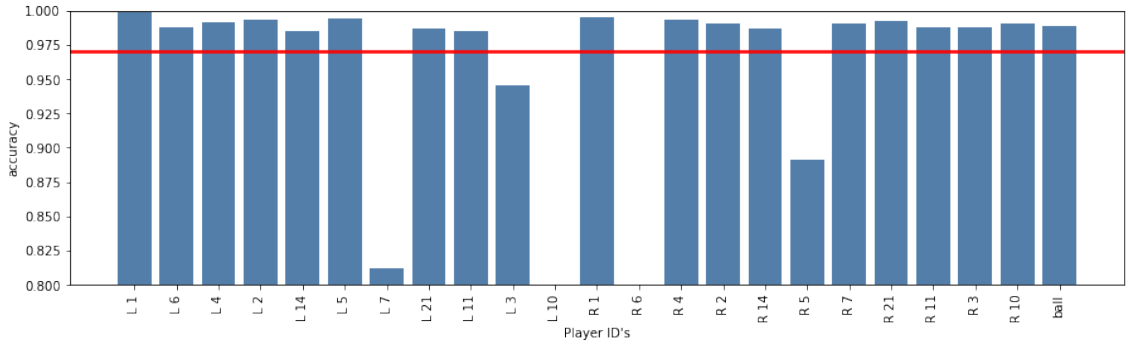


Figure B-3: Accuracy threshold for all templates in a single image. Only the matches with an accuracy above the threshold were accepted as matches.

B-2-2 Detection percentages of detection algorithms

The detection percentages of the detection algorithms of all games can be observed in Table B-2. Games where percentages reached below values of 75% were discarded.

Game no.	Players location	Ball location	Possession team
1	82.3 %	80.9 %	82.6 %
2	85.6 %	87.0 %	84.2 %
3	86.1 %	90.8 %	74.3 %
4	88.1 %	88.1 %	84.7 %
5	84.4 %	87.4 %	80.2 %
6	83.4 %	85.7 %	84.9 %
7	75.2 %	90.4 %	89.6 %
8	84.5 %	89.7 %	82.7 %
9	75.1 %	90.1 %	87.8 %
10	75.6 %	90.9 %	88.8 %
11	79.4 %	87.4 %	84.6 %
12	51.7 %	87.4 %	86.6 %
13	54.2 %	89.3 %	86.6 %
14	81.0 %	92.3 %	90.8 %
15	53.7 %	91.3 %	91.0 %
16	56.3 %	91.1 %	90.3 %
17	48.8 %	88.4 %	87.3 %
18	68.5 %	92.3 %	92.3 %
19	78.2 %	86.5 %	86.5 %
20	79.2 %	90.0 %	90.0 %
21	69.3 %	87.9 %	87.9 %
22	75.8 %	87.7 %	87.7 %
23	76.5 %	86.0 %	86.0 %
24	75.1 %	91.4 %	91.4 %
25	67.8 %	94.4 %	94.4 %

Table B-2: Number of times the locations of players and ball as well as the possession team were identified, given as a percentage of all cases per match.

Appendix C

Decision-Making Models

This section will propose two decision-making models as extensions of the prediction model as defined in chapter 6. Both decision-making models will be based on the Markov Decision Process (MDP) model as defined in section 4-2. However, the MDP model will now include multiple actions in the action space and will subsequently optimize over actions. Ultimately, the Multilayer Perceptron (MLP) will approximate the action-value function by redefining the Expected Danger (ED). The ED will then be able to give a long-term value to each action $a \in A$ when in any given state $\tilde{s} \in \tilde{S}$. Then, by testing all actions $a \in A$ and comparing the output values of this function, it should be possible to determine the best action to take in every given scenario. An updated function approximator $ED^h : \tilde{S} \times A \rightarrow \mathbb{R}$ for the state-action function can now be defined as follows:

$$ED^h(\tilde{s}, a) = \frac{1}{\min(h, T-t)} \quad \mathbb{E}\left[\sum_{k=1}^{\min(h,T-t)} \theta_{t+k} \mid \tilde{x}_t = \tilde{s}, a_t = a\right]. \tag{C-1}$$

Both models will try to find a method to approximate the updated version of the ED. In short, their approximation approach can be described as follows.

- **Model proposal 1:** The first decision-making model adds an extra *input* to the existing MLP prediction model.
- **Model proposal 2:** The second decision-making model adds an extra *output* to the existing MLP prediction model.

Both models have their advantages and drawbacks. Furthermore, it should be noted that both models are mere theoretical models and have not been practically implemented yet due to time limitations. Both models will be proposed in subsequent parts. Model 1 will be proposed in subsection C-0-1 and model 2 in subsection C-0-2.

C-0-1 Model proposal 1

The predictor function can be extended such that can make predictions about the state trajectory when different actions are taken. As a result, it will become possible to assign a value to an action $a \in A$, taken in state $\tilde{s} \in \tilde{S}$, and consequently, optimize over the actions to find an optimal or sub-optimal policy. In order to do so, an extra input entry will be added, namely: action a_t taken at time t. The input layer will consist of $n_s + 1$ scalar input values. The updated and trained MLP is schematically depicted in Figure C-1.

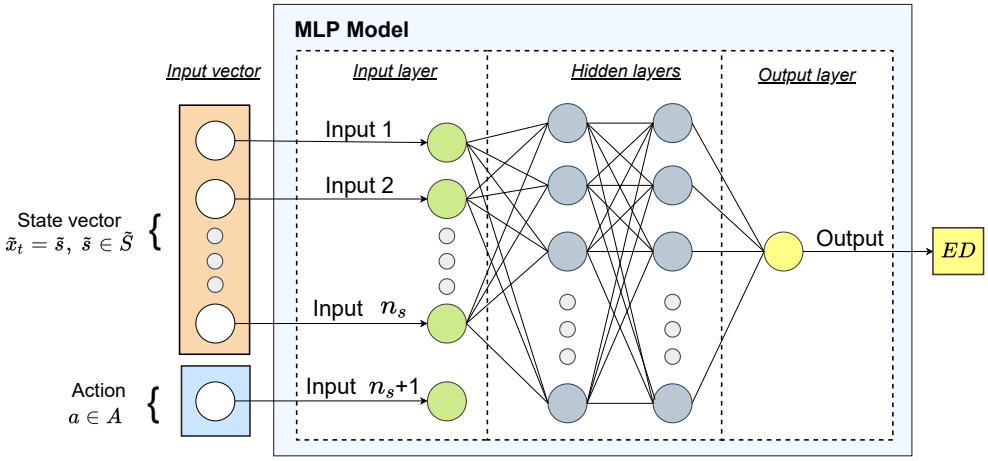


Figure C-1: Solution proposal 1: extra input to the MLP model.

In practice, within-game actions (i.e., formation changes) cannot occur as frequently as an optimal policy might want it to. It takes time for all players to get the same signal (action delay), and then they need time to adjust to their new formation. Therefore, it makes sense to set a minimum time boundary between any two actions. A change of actions can then only take place after a predefined time period. This minimum time boundary will be set to equal to horizon h, as this is how far will be predicted into the future and it will be practically convenient to let each time interval $[t, t+h]$ finish before a new action can be chosen. The action sequence $\bar{a}$ for an entire episode can be formulated as follows,

$$\bar{a} = \langle a_0,\ a_1,\ \ldots,\ a_{h-1},\ a_h,\ \ldots, a_{2h-1}, a_{2h}, \ldots, a_{T-1} \rangle, \qquad a_i \in A,\ \forall i \tag{C-2}$$

The action-taker's choice of actions is restricted by a set of action constraints, active only at certain moments in time. Let $C \subset A$ be a set of constraints prohibiting the action-taker to change its actions at certain times. The action-taker can freely choose any action at the following points in time, where the actions space is unconstrained:

$$\langle t_0, t_{h-1}, t_{2h-1}, t_{3h-1}, \ldots t_{kh-1} \rangle, \tag{C-3}$$

which occur periodically until time t_{kh-1}, with $k \in \mathbb{Z}^+$. At all other time instances, the action constraints C will be active. At those times, the action-taker is not allowed to change action, thereby fixing its policy until the fixed horizon is reached. The set of available actions at each time instance is limited by the constraints C and can be defined as the constrained action space $\bar{A}$. With respect to the action sequence in Equation C-2, this means that $a_0 = a_1 = \ldots = a_{h-1}$, and also

$a_h = a_{h+1} = \ldots = a_{2h-1}$, etc. A schematic overview of this process is given in Figure C-2, which shows the possible state trajectories and action possibilities starting from state $\tilde{x}_t$ and ending in state $\tilde{x}_{t+h+1}$.

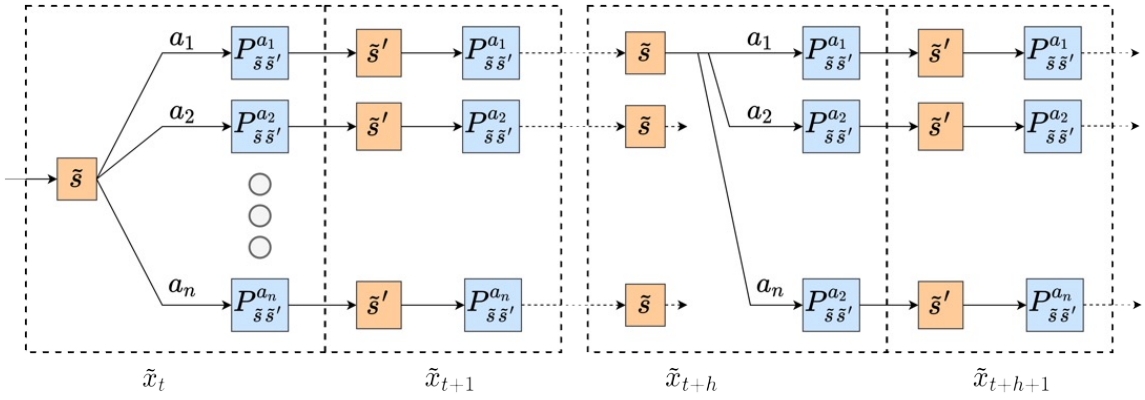


Figure C-2: Schematic overview of all state trajectory possibilities, each with its own probability. Actions $a \in \bar{A}$ can only be taken at certain time instances due to the constrained action space.

An optimal policy π^* when in any state $\tilde{s} \in \tilde{S}$ can now be defined as follows,

$$\pi^*(\tilde{s}) = \arg\max_a ED^h(\tilde{s}, a) \qquad \text{subject to} \quad a \in \bar{A} \tag{C-4}$$

At all unconstrained points in time, an optimal policy can be computed until the fixed horizon is reached. Finding the optimal policy in the MDP environment under the constrained action space will be the goal of this model.

Advantages and drawbacks

The main advantage of the model is that it can be created with relatively few adjustments to the original prediction model. The main disadvantage is that the creation of this model will be rather time-consuming due to the need to generate a very large amount of training examples. Each training example is created from a time interval $[kt, kt+h]$, with h the prediction horizon and $k = \{0, 1, 2, \ldots\}$. A single action is then taken at the beginning of each time interval, which will be held for its entire duration. The idea of each training example is to observe the effects of the action taken in a given state. In contrast to the prediction model, training examples can therefore not have any overlap in their time intervals. As a result, a lot of additional matches need to be simulated to generate enough training examples to correctly train a new Artificial Neural Network (ANN).

Practical implementation

The model can be practically implemented by following a series of steps. These steps are summarized in the following:

1. Choose and fix a horizon h.

2. Simulate and screen record new matches using the eFootball simulator, and randomly choose actions $a \in \bar{A}$ during the matches at the action times defined in Equation C-3. Annotate which action was chosen at which moment in time.

3. Convert the screen recordings to a data set using the digital image processing algorithm.

4. Adjust the data preprocessing algorithm such that it creates labelled training examples which include the action as input. Then use this algorithm to create a training set.

5. Update the MLP by increasing the size of the input layer and train the network. A similar network architecture and hyperparameters can be used as the prediction model because the problems are alike.

6. Validate and test the model on a separate, unbiased dataset to check the prediction performance of the model.

7. If results are satisfactory, a decision-making model has now been made. When using the model online, the current state can be used as the first n_s input values to the model. Additionally, each possible action $a \in A$ can be used as the $n_s + 1$ input variable to the model. Different actions lead to different output values of the ED^h. The action $a \in A$ with the highest output value for the ED can then be seen as the most favorable action to take.

C-0-2 Model Proposal 2

Again, the MLP model can be extended such that can make predictions about the state trajectory when different actions are taken. As a result, it will hopefully become possible to assign a value to an action $a \in A$, taken in state $\tilde{s} \in \tilde{S}$, and consequently, optimize over the actions to find an optimal or suboptimal policy. In order to do so, a new MLP will be designed such that it can fit two output values.

Training procedure

The model will be trained using a different training technique than applied in this work that is based on the work of Mnih et al. (2015) [93] about human-level control through Reinforcement Learning (RL), who were able to train an agent using RL to make it successfully execute challenging tasks within 2600 Atari games. The training method consists of grouping successive labelled examples into chunks, where the examples in each chunk consist of only a single action taken, denoted as c_i^a. A chunk of examples where only the action $a = 433$ is taken might look as follows: $c_1^{a=433} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_{20}, y_{20})\}$. The MLP model will then be trained for each individual chunk, however, only the connections in the MLP that are connected to the output value of the action a of that particular chunk are kept intact. The architecture of the model during training will therefore be the same as the prediction model, and thus only have a single output value. The difference is that this output gate switches based on the action of each chunk of training data. This method resembles transfer learning techniques, however, instead of transferring the hyperparameters, the model now transfers both the hyperparameters and the parameters after training a model on each chunk.
After training, the weights of all output gates will be restored such that the amount of outputs is

equal to the size of the action space. The updated predictor function $ED : \tilde{S} \rightarrow \mathbb{R}^{n_a}$ should now be able to compute different ED values corresponding to different actions taken at time t. The coach is then encouraged to choose the action corresponding to the highest ED. A schematic depiction of this process is given in subsection C-0-2.

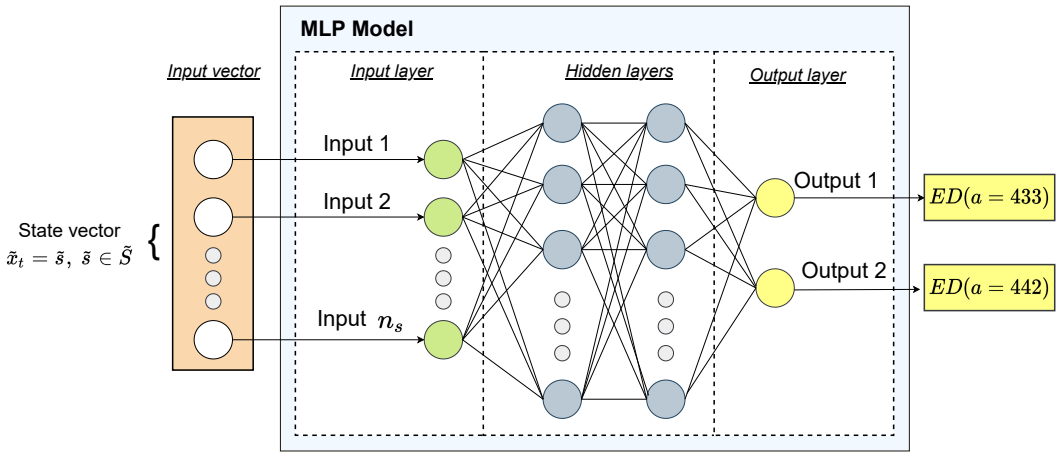


Figure C-3: Solution proposal 2: extra output to MLP model.

Advantages and drawbacks

The disadvantage of this model is that it involves more complexity and will therefore be harder to design than the model of proposal 1. The main advantage is that the model is not dependent on the creation of an extremely large amount of labelled examples, and has the potential to make accurate predictions over different actions using only a relatively small amount of data. Solution proposal 2 is therefore preferred by the author.

Practical implementation

The model can be practically implemented by following a series of steps. The implementation in the following parts only considers an action space of two actions, so $|A| = 2$. However, this was done for explanatory reasons. If desired, the action space can be extended to an arbitrary size. The steps are summarized in the following:

1. Create two datasets, where the user team in each dataset plays according to a different action. For example, the first dataset only contains data of a 4-3-3 formation, while the other consists of games played in a 4-4-2 formation.

2. Convert the data in the datasets to labelled examples using the preprocessing techniques as described in this work. Each dataset of labelled examples will correspond to a formation used.

3. Divide each dataset into chunks of labelled examples, where each chunk has the same size (i.e. amount of examples) as the mini-batch size of the MLP. Both datasets will now be split into smaller chunks.

4. Order the chunks such that one chunk of one dataset follows a chunk of the other dataset in 1-by-1 pattern. The ordered set of chunks of both datasets will now serve as the training set of the MLP.

5. Randomly initialize the parameters of the MLP to non-zero values.

6. Create an architecture where the connections to one of the outputs will be removed after each mini-batch of training, while the connections to the other output will be replaced. This means that the connections between the last two layers will look like a flashing light during training. Furthermore, recall that each output represents a single action and the chunks of data have the same size as the mini-batch. Consequently, each mini-batch of training will update the weights and biases based on a single output and thus a single action.

7. Change optimization algorithm from Adam's optimizer to gradient descent, as Adam's optimizer uses information of the parameters of previous mini-batch iterations.

8. Train the MLP on the entire dataset. Apart from the connections between the last two layers, the same weights and biases will be used for the entire network.

9. After training, performance can be tested. Any state $\tilde{s} \in \tilde{S}$ can be used as input at time t, and the ED outputs then correspond to a 4-3-3 action and a 4-4-2 action. The action which has the higher ED output value then represents the superior action at time t.