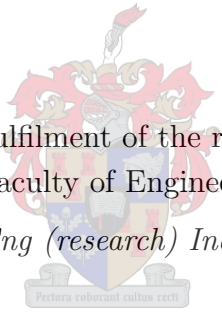
Developing basic soccer skills using reinforcement learning for the RoboCup Small Size League

Moonyoung Yoon

Department of Industrial Engineering

University of Stellenbosch



Thesis presented in partial fulfilment of the requirements for the degree of Master of Engineering in the Faculty of Engineering at Stellenbosch University

MSc.Eng (research) Industrial

Study leaders: James Bekker, Steve Kroon

March 2015

Dedicated to my mother, who is the best mom in the world.

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe on any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date:. . . . . .

Acknowledgements

I would like to acknowledge that I am grateful to the following people:

- Prof. James Bekker and Prof. Steve Kroon, my supervisors, for their supports in technical, emotional and financial ways.
- Prof. YS Kim, for guiding me to a new chapter of my life.
- Marlene Rose, for proofreading the thesis.
- My friends here, who shared the office, frustration and hope.
- My family, for encouraging me and enduring my absence.
- God, my Creator and Savior, my love and strength, for being there all the time watching me.

Abstract

This study has started as part of a research project at Stellenbosch University (SU) that aims at building a team of soccer-playing robots for the RoboCup Small Size League (SSL). In the RoboCup SSL the Decision-Making Module (DMM) plays an important role for it makes all decisions for the robots in the team. This research focuses on the development of some parts of the DMM for the team at SU.

A literature study showed that the DMM is typically developed in a hierarchical structure where basic soccer skills form the fundamental building blocks and high-level team behaviours are implemented using these basic soccer skills. The literature study also revealed that strategies in the DMM are usually developed using a hand-coded approach in the RoboCup SSL domain, i.e., a specific and fixed strategy is coded, while in other leagues a Machine Learning (ML) approach, Reinforcement Learning (RL) in particular, is widely used. This led to the following research objective of this thesis, namely to develop basic soccer skills using RL for the RoboCup Small Size League. A second objective of this research is to develop a simulation environment to facilitate the development of the DMM. A high-level simulator was developed and validated as a result.

The temporal-difference value iteration algorithm with state-value functions was used for RL, along with a Multi-Layer Perceptron (MLP) as a function approximator. Two types of important soccer skills, namely shooting skills and passing skills were developed using the RL and MLP combination. Nine experiments were conducted to develop and evaluate these skills in various playing situations. The results showed that the learning was very effective, as the learning agent executed the shooting and passing tasks satisfactorily, and further refinement is thus possible.

In conclusion, RL combined with MLP was successfully applied in this research to develop two important basic soccer skills for robots in the RoboCup SSL. These form a solid foundation for the development of a complete DMM along with the simulation environment established in this research.

Opsomming

Hierdie studie het ontstaan as deel van 'n navorsingsprojek by Stellenbosch Universiteit wat daarop gemik was om 'n span sokkerrobotte vir die RoboCup Small Size League (SSL) te ontwikkel. Die besluitnemingsmodule (BM) speel 'n belangrike rol in die RoboCup SSL, aangesien dit besluite vir die robotte in die span maak. Hierdie navorsing fokus op ontwikkeling van enkele komponente van die BM vir die span by SU.

'n Literatuurstudie het getoon dat die BM tipies ontwikkel word volgens 'n hiërargiese struktuur waarin basiese sokkervaardighede die fundamentele boublokke vorm en hoëvlak spangedrag word dan gerealiseer deur hierdie basiese vaardighede te gebruik. Die literatuur het ook getoon dat strategieë in die BM van die RoboCup SSL domein gewoonlik ontwikkel word deur 'n hand-gekodeerde benadering, dit wil sê, 'n baie spesifieke en vaste strategie word gekodeer, terwyl masjienleer (ML) en versterkingsleer (VL) wyd in ander ligas gebruik word. Dit het gelei tot die navorsingsdoelwit in hierdie tesis, naamlik om basiese sokkervaardighede vir robotte in die RoboCup SSL te ontwikkel. 'n Tweede doelwit was om 'n simulasie-omgewing te ontwikkel wat weer die ontwikkeling van die BM sou fasiliteer. Hierdie simulator is suksesvol ontwikkel en gevalideer.

Die tydwaarde-verskil iterariewe algoritme met toestandwaarde-funksies is gebruik vir VL saam met 'n multi-laag perseptron (MLP) vir funksiebenaderings. Twee belangrike sokkervaardighede, naamlik doelskop- en aangeevaardighede is met hierdie kombinasie van VL en MLP ontwikkel. Nege eksperimente is uitgevoer om hierdie vaardighede in verskillende speelsituasies te ontwikkel en te evalueer. Volgens die resultate was die leerproses baie effektief, aangesien die leer-agent die doelskiet- en aangeetake bevredigend uitgevoer het, en verdere verfyning is dus moontlik.

Die gevolgtrekking is dat VL gekombineer met MLP suksesvol toegepas is in hierdie navorsingswerk om twee belangrike, basiese sokkervaardighede vir robotte in die RoboCup SSL te ontwikkel. Dit vorm 'n sterk fondament vir die ontwikkeling van 'n volledige BM tesame met die simulasie-omgewing wat in hierdie werk daargestel is.

Contents

List of Figures

List of Tables

List of Algorithms

Nomenclature

Abbreviations and Acronyms

AI	Artificial Intelligence
ANN	Artificial Neural Network
API	Application Programming Interface
BP	Back-Propagation
BPA	Back-Propagation Algorithm
CBR	Case-Based Reasoning
DMM	Decision-Making Module
EA	Evolutionary Algorithm
FIRA	The Federation of International Robot-soccer Association
FM	Frequency Modulation
FNN	Fuzzy Neural Network
GA	Genetic Algorithm
GUI	Graphic User Interface
HARL	Heuristically Accelerated Reinforcement Learning
LVQ	Learning Vector Quantisation
MDP	Markov Decision Process

MFC	Microsoft Foundation Class
MiroSot	The Micro-Robot World Cup Soccer Tournament
ML	Machine Learning
MLP	Multi-Layer Perceptron
MSE	Mean Squared Error
MSL	Middle Size League
ODE	Open Dynamics Engine
PR	Pattern Recognition
RL	Reinforcement Learning
SL	Supervised Learning
SOM	Self-Organising Map
SSL	Small Size League
STP	Skills, Tactics and Plays architecture
SU	Stellenbosch University
TCP/IP	Transmission Control Protocol/Internet Protocol
TD	Temporal Difference
TL	Transfer Learning

Greek Symbols

α	The angle between the orientation of the learning agent and the line connecting the ball and the target, one of the state variables in RL experiments
β	The angle between the line connecting the learning agent and the ball, and the line connecting the ball and the target, one of the state variables in RL experiments

δ	Parameter used for a step-size in gradient-descent methods
ϵ	A small positive value to define the exploration rate for ϵ-greedy search
η	Learning rate in BP algorithms and in RL
η_n	Learning rate after the n^{th} learning episode in BP algorithms
γ	Discount rate for future rewards in RL
Ω	A set of possible angular velocities of the learning agent in RL experiments
ω	Angular velocity of the learning agent, one of the action variables in RL experiments
π	Policy in RL
$\pi(s)$	Action chosen by the policy π in state s
$\pi(s, a)$	Probability of taking action a in state s under the policy π
π^*	Optimal policy in RL
φ	The angle between the moving direction of the ball and the line connecting the ball and the target, one of the state variables in RL experiments
Θ	A set of possible moving directions of the learning agent in RL experiments
θ	Moving direction of the learning agent, one of the action variables in RL experiments

Roman Symbols

A	A set of actions in RL
a	Current action in RL
a_t	Action chosen at time-step t in RL

b	Bias in a neuron
c	Shape parameter of a sigmoid function
D	Distance between the ball and the learning agent, one of the state variables in RL experiments
D_{Max}	Maximum distance allowed between the ball and the learning agent in RL experiments
$\mathbb{E}$	Expected value
$\mathbb{E}_{\pi}$	Expected value given that the agent follows policy π
F	Function approximator
h	Net input of a neuron
l	Number of neurons in the hidden layer in a MLP
m	Number of neurons in the input layer in a MLP
n	Number of neurons in the output layer in a MLP
N_{Max}	Maximum number of learning episodes in an RL experiment
n_{Max}	Maximum number of steps in an episode in RL experiments
N	Number of samples in a training data set
P	Probability
$\mathcal{P}^{a}_{ss'}$	Probability of a possible next state s' given a state s and an action a in RL
Q	Action-value function in RL
$Q(s,a)$	The value of taking an action a at a state s in RL
Q_k	The estimation of the action-value function Q at k^{th} iteration in RL
$\mathfrak{q}_t$	TD target at time-step t for action-value functions in RL

$Q^{\pi}(s, a)$	The value of taking an action a at a state s and following a policy π thereafter in RL
Q^{*}	Optimal action-value function in RL
$\mathbb{R}$	The set of real numbers
r_a	Reward given to the agent as a result of its action a
R_t	Sum of the rewards given to the agent after time-step t in RL
r_{t+1}	Reward given to the agent as a result of its action at time-step t in RL
$\mathcal{R}^{a}_{ss'}$	Expected value of the reward given any current state and action, s and a, along with any next state s' in RL
S	A set of states in RL
s	Current state in RL
s_t	State of the environment at time-step t in RL
S'	A set of states visited by the learning agent in RL
s'	Next state in RL
s'_a	Next state given the action a in RL
S^{+}	A set of terminal states with success in RL
S^{-}	A set of terminal states with failure in RL
T	Final time-step in an episode in RL
$\mathbb{T}$	Training data set
t	Index for time-step in RL
$t(s)$	Target value given a state s in RL
t_i	Correct output of the i^{th} input in a training data set

V	State-value function in RL
$\mathbb{V}$	A set of possible moving speeds of the learning agent in RL
v	Moving speed of the learning agent, one of the action variables in RL experiments
$\mathfrak{v}$	TD target for state-value functions in RL
$V(s)$	The value of a state s in RL
V_k	The estimation of the state-value function V at the k^{th} iteration in RL
v_{bx}	Velocity of the ball in x-direction, one of the state variables in RL experiments
v_{by}	Velocity of the ball in y-direction, one of the state variables in RL experiments
v_{ij}	The weight connecting the i^{th} input and the j^{th} neuron in the hidden layer in Algorithm 6 and Algorithm 7
$\mathfrak{v}_t$	TD target at time-step t for state-value functions in RL
v_x	Velocity of the learning agent in x-direction, one of the state variables in RL experiments
v_y	Velocity of the learning agent in y-direction, one of the state variables in RL experiments
V^π	State-value function under the policy π in RL
V_k^π	The estimation of V^π at the k^{th} iteration in RL
$V^\pi(s)$	The value of a state s under a policy π in RL
$V_k^\pi(s)$	The estimation of $V^\pi(s)$ at the k^{th} iteration in RL
V^*	Optimal state-value function in RL
w_i	The weight for the i^{th} input of a neuron

w_{jk}	The weight connecting the j^{th} neuron in the hidden layer and the k^{th} neuron in the output layer in Algorithm 6 and Algorithm 7
x_i	The i^{th} input of a neuron
y	Output of a neuron
y_i	Mapped output of the i^{th} input in a training data set
$\vec{z}$	A parameter vector used in the function approximator F

Chapter 1

Introduction

1.1 Background

1.1.1 Artificial intelligence

Artificial Intelligence (AI) is probably one of the most interesting fields in science and engineering. Born only in the mid-20th century, it is still young and growing, providing researchers with good opportunities to lay a solid foundation for research in the field. Aiming to develop intelligent agents (Poole *et al.*, 1998), AI encompasses a wide range of subfields. Optimisation, evolutionary algorithms, decision theory, machine learning and neural networks can be seen as tools or methods of AI, and its application includes speech recognition, image processing, machine translation, game playing, automation, medical diagnosis, robotics and many more.

The victory of a computer, Deep Blue, over the human chess world champion in 1997 was probably the most outstanding achievement in AI's history until then. It was not only a great breakthrough but also became a turning point of mainstream AI research. The focus then shifted to more complicated problems, that is, developing intelligent agents working in dynamic, uncertain environments.

1.1.2 RoboCup

RoboCup (RoboCup webpage, 2014), an annual international robot soccer competition, is one such attempt to promote AI by performing a common task: soccer (Kitano *et al.*, 1997). It offers an integrated test-bed to develop a team of fully autonomous

soccer-playing robots. In order to achieve this, various technologies must be incorporated including autonomous agent design principles, multi-agent collaboration, strategy acquisition, real-time reasoning, robotics and sensor-fusion (Visser & Burkhard, 2007). Kitano & Asada (1998) stated the ultimate goal of the RoboCup initiative as follows:

> *"By the mid-21st century, a team of fully autonomous humanoid robot soccer players shall win a soccer game, in compliance with the official rules of the FIFA, against the winner of the most recent World Cup."*

RoboCup Soccer is divided into five leagues: the Small Size League (SSL), the Middle Size League (MSL), the Simulation League, the Standard Platform League and the Humanoid League. Each league has its own challenges. This research forms part of a project at Stellenbosch University (SU) that aims to build a team of soccer-playing robots conforming to the rules of the RoboCup SSL. Therefore, it is focused on the RoboCup SSL, which is concerned with the problem of intelligent multi-agent cooperation and control in a highly dynamic environment with a hybrid centralised/distributed system (RoboCup SSL webpage, 2014).

1.1.3 RoboCup Small Size League

In the RoboCup SSL, teams consisting of maximum six robots play soccer games using an orange golf ball on a pitch of 6.05 m $\times$ 4.05 m. The robot shape and size are confined to a cylinder with a diameter of 180 mm and a height of 150 mm. Activities on the field are captured by two cameras mounted above the field and the corresponding information, such as the positions of robots and the ball, is processed by open-source software called SSL-Vision on an off-field computer. Using this information, an independent computer program, usually called an AI module in the literature, produces team strategies for the robot actions and sends commands to the robots via a wireless radio frequency. Changes on the field caused by movements of the robots and the ball are again captured by the cameras and all the processes described above are repeated throughout the game. This control loop iterates approximately 60 times per second. Figure 1.1 shows the RoboCup SSL system.

The hardware design of the robots developed at SU is shown in Figure 1.2(a). The robot has four omnidirectional wheels (Figure 1.2(b)) and a kicker unit. The omnidirectional wheels have numerous small wheels all around the circumference. These small

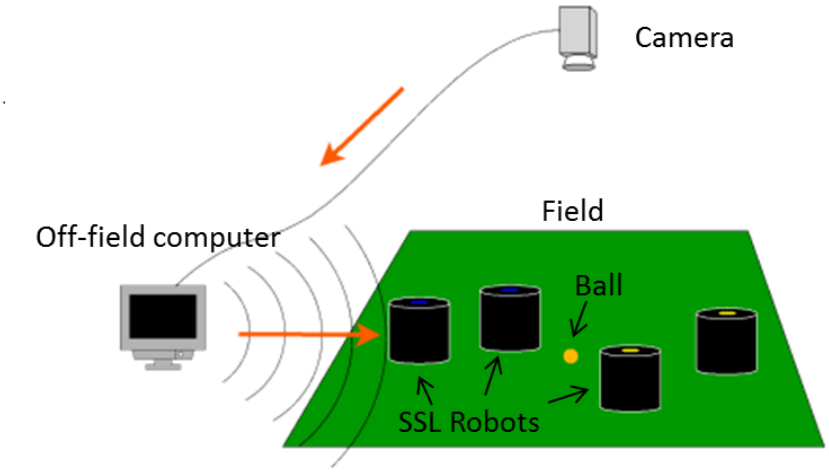
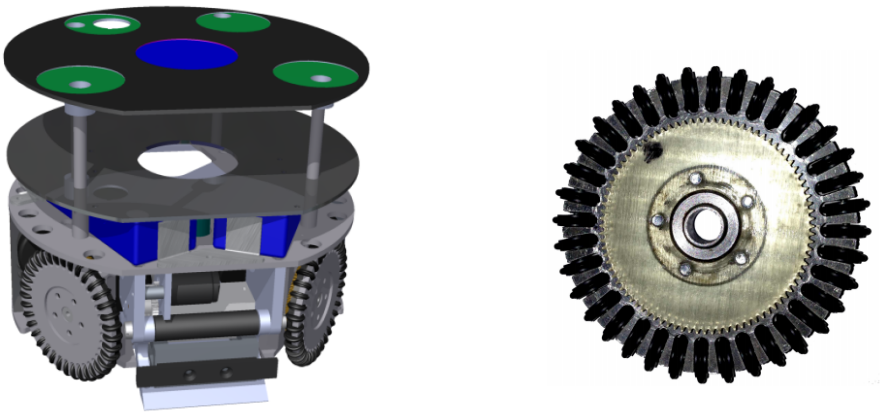


Figure 1.1: The RoboCup SSL system (RoboCup SSL webpage, 2014).



(a) CAD design of the SSL robot

(b) Omnidirectional wheel

Figure 1.2: Hardware design of an SSL robot (Smit, 2014).

wheels rotate freely, enabling the robot to move in any direction without having to turn. This feature has a significant effect on the design of the experiments to be performed in this research, which will be discussed later in Chapter 5 and Chapter 6.

1.2 Motivation

As one can see from the above description of the RoboCup SSL system, global perception is possible and control is centralised. Each robot does not have an independent control unit, nor does it make decisions on its own. The robots move only according to the instructions from the omniscient brain of their team, i.e., the Decision-Making Module[1] (DMM). This is called the *centralised* control system.

Another way to control the system, which is more complicated but realistic, is through a *distributed* control system. In a distributed control system, each robot has on-board sensors to capture the information about the field and makes decisions independently based on this information. All the other RoboCup leagues require distributed control, while both centralised and distributed control are allowed in the RoboCup SSL. However, the use of distributed control systems is hardly found in the context of the RoboCup SSL. The RoboCup SSL is used to rather explore good strategies in a centralised control system, where the DMM plays a key role.

Relatively speaking, the DMM has drawn less attention for its importance in the SSL domain. Because the mechanics is still a challenging issue, teams in the RoboCup SSL have been concentrating on robot hardware and control problems rather than strategies. This research, however, focused purely on the development of an intelligent DMM for the robot soccer team at SU.

1.3 Objectives

Although the DMM forms the focal point of this research, it is not the objective of this research to develop a complete DMM, which would be able to control an entire soccer match. Developing a DMM would involve a huge amount of programming work and thus require far more human resources than those available for Master's research.

[1]The DMM corresponds to the *AI module* described in the previous section. However, in this context, the term *decision-making module* is preferred as it indicates more precisely the function of the module in question.

However, developing a complete DMM remains a long-term goal of the SU project team's AI research.

The objective of this research is therefore to lay a foundation for the development of a DMM by establishing basic building blocks of the DMM. The aim is also to provide a simulation environment, which is absolutely required to test and validate the function of the DMM.

1.4 Methodology

The methodology used in this research can be summarised as follows:

1. Develop a simulation environment.
2. Define basic building blocks of the DMM: individual soccer skills.
3. Select some soccer skills to be developed in this research.
4. Implement the selected skills using machine learning, focusing on reinforcement learning techniques.

Each step will briefly be described in the following sections.

1.4.1 A simulation environment

In order to develop a DMM, a simulation environment must be constructed first. A simulation environment in this context means a system in which the function of the DMM can be tested and validated with no involvement of hardware such as real robots and cameras. The simulation environment, therefore, should include the DMM and a simulator that is able to simulate the control processes described in Section 1.1.3, excluding the processes done by the DMM. It would be impossible to develop the DMM in a time- and cost-effective way without the simulation environment. Developing a simulation environment is therefore a prerequisite to the development of the DMM.

1.4.2 Basic building blocks of the DMM

Once the simulation environment has been implemented, the next step would be to design the DMM. One of the important approaches towards the design of the DMM is to build it in a layered architecture with different levels of abstraction.

In a layered architecture, individual soccer skills such as **Shoot**, **Pass**, **Intercept**, etc., are developed first, then they are used to implement high-level team behaviours such as **AgressiveAttack** or **Defence**. Individual soccer skills, in this context, indicate actions that can be performed by a single robot. In some teams individual soccer skills are further divided into lower-level skills, such as **GoToAPoint**, **AimAtTheGoal**, **Kick**, etc. These lower-level skills are used in the implementation of upper-level individual skills. In other teams, individual skills are developed as a single layer.

Although the number of layers and the level of abstraction in each layer vary depending on the design, the most important feature that is found in the design of the layered architecture in almost all teams in the RoboCup SSL is that individual soccer skills are developed as complete, independent modules. Furthermore, high-level team behaviours are implemented by using these modularised individual skills.

This layered approach makes it relatively easy to develop and test various strategies. Once a collection of individual skills is implemented, strategies can be developed simply by writing different team behaviours using those individual skills. They can be changed without causing many alterations to other parts of the program. Also, new individual skills can easily be added to the existing skills. In this sense, the individual soccer skills form essential building blocks to build a DMM efficiently.

1.4.3 Important individual soccer skills

In Section 1.3 it was mentioned that one of the important goals of this research is to lay a foundation for the development of the DMM by establishing basic building blocks. This can be achieved by means of two steps: defining the building blocks and actually developing them. The layered architecture was discussed in the previous section to define individual soccer skills as the basic building blocks. In this section, six basic soccer skills are introduced as the building blocks that will be developed in this research. They are:

- shooting a stationary ball
- shooting a moving ball
- shooting a stationary ball against a goalkeeper
- shooting a moving ball against a goalkeeper

- passing a stationary ball
- passing a moving ball.

These skills were chosen because shooting and passing skills are the most fundamental soccer skills. At the same time, it is not as simple to implement these skills as it seems. Shooting a stationary ball, for example, includes approaching the ball, aiming at the goal or the target and kicking the ball. The robot should approach the ball from the opposite direction of the goal while maintaining the right angle to keep the ball in front of its kicker. It should also be able to control its velocity so that the ball gets close enough to kick, but the robot must not get so close to the ball that it pushes it instead of kicking.

1.4.4 Machine learning

Although the layered architecture can significantly reduce the effort required to develop the DMM, it is still exceptionally hard work to implement strategies. Usually, strategies are hand-coded and fine-tuned manually. Hand-coded solutions may work well with relatively simple tasks, but developing them becomes harder and more time consuming as the complexity of the task increases. Also, there is a strong possibility that the work could end up with a bad solution because the hand-coding approach is highly dependent on human logic.

Machine Learning (ML), a subfield of AI that concerns the construction and study of systems that can learn from data, offers a good alternative to hand-coding approaches. Human brain power can be replaced with computer power by using ML (Riedmiller *et al.*, 2009), and the learning result is less error prone (Meriçli *et al.*, 2011).

ML is often used for the strategy development in other RoboCup leagues, but it is hardly found in the SSL context. Possible reasons are discussed in Section 3.1.3. In this research, it was decided to explore the possibility of using ML for strategy development. Therefore, although the individual skills to be developed in this research are relatively simple, making the hand-coding approach a possibility, the strategies were developed using ML, specifically, reinforcement learning. The idea here is, instead of straightforward programming, to let the robot try various actions, observe the results, and learn from the experience. ML and the rationale for the use of reinforcement learning will be dealt with in detail in Chapter 2.

There are a few cases in the literature, all based on other leagues, where similar tasks have been developed using reinforcement learning. However, the differences in hardware requirements, control methods or problem designs make the use of these existing learning skills inappropriate for the problems investigated in this research, if not impossible. These issues will be discussed further in the literature review in Section 3.2.1, when each case is introduced and explained.

1.5 Structure of the thesis

The rest of the thesis is structured as follows: Chapter 2 provides the theoretical background to ML. Two main topics of ML will be presented in particular: reinforcement learning and the multi-layer perceptron. A brief overview of the strategy development process as well as examples of machine learning applications in robot soccer literature will be introduced in Chapter 3. Chapter 4 discusses the simulation environment established in this research. This is followed by Chapters 5 and 6, which present ML experiments performed to implement the shooting and passing skills mentioned in Section 1.4.3. Finally, Chapter 7 concludes the thesis with a short summary and recommendations on future work.

Chapter 2

Machine learning

This chapter introduces two important topics in machine learning, namely Reinforcement Learning (RL) and the Multi-Layer Perceptron (MLP). These two concepts are central to the main methodology of the research. Machine learning is briefly explained first, followed by a discussion of RL and MLP. These sections will provide the theoretical basis for the machine learning experiments described in Chapters 5 and 6.

2.1 Introduction to machine learning

Thanks to the rapid development of technology, many complex problems can be solved with the help of computer programs. These computer programs use various algorithms to solve problems. However, there are problems that have no specific algorithms. For example, an algebra problem can be solved by a computer program using an algorithm, but problems such as which customers on the client list would be interested in a newly launched car, are tricky. There is no explicit algorithm for this problem. The clue to solving these kinds of problems is *data*. Information such as a customer's age, gender, monthly income, purchase history, etc., is certainly connected to the customer's purchasing behaviour. Data plays an important role to finding the solution to the problem. Although an explicit algorithm cannot be written for this problem, a computer program can extract an algorithm automatically from the data (Alpaydin, 2010). This is called *Machine Learning* (ML).

ML is literally about studying machines, or computer programs, to be specific, that can learn. Arthur Samuel defined ML as a field of study that gives computers the ability

to learn without being explicitly programmed (Simon, 2013). It is widely quoted that "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E"(Mitchell, 1997).

Marsland (2011) classified ML into the following four categories according to different methods by which the algorithms find answers:

- **Supervised learning**: A training set of examples with correct responses (targets) is provided and, based on this training set, the algorithm generalises to also respond correctly to inputs not included in the training data.
- **Unsupervised learning**: Correct responses are not provided. Instead, the algorithm tries to identify similarities between the inputs so that inputs that have something in common are categorised together.
- **Reinforcement learning**: This is somewhere between supervised learning and unsupervised learning. The algorithm gets told when the answer is wrong, but does not get told how to correct it. It has to explore and try out different possibilities until it works out how to get the answer right.
- **Evolutionary learning**: The algorithm uses biological evolution concepts to represent current solutions, to assess their fitness, and to produce new (better) solutions.

Of these categories, reinforcement learning has particularly drawn attention in this research because it provides an environment that suits the problems of implementing the shooting and passing skills mentioned in Section 1.4.3. This is discussed in detail in the following section where reinforcement learning is explained. The concept of multilayer perceptron, one of the typical supervised learning methods, is also dealt with in this research (Section 2.3) to accommodate problems with continuous variables.

2.2 Reinforcement learning

Reinforcement Learning (RL) is a branch of Machine Learning (ML) where the learning agent tries to learn to perform a specific task in a specific environment. A situation in the environment is defined by *states*, and the learning agent is given a set of *actions* it

can choose from at a given state. The state $s_t \in S$ defines the environment's situation at time t, where S is the set of possible states. At each time-step, the agent chooses an action $a_t \in A$, where A is the set of possible actions. Performing an action transitions the agent to a new state s_{t+1} at the next time-step, and the agent receives a *reward* of $r_{t+1} \in \mathbb{R}$ as a result of the action.

The agent's goal is to maximise the overall rewards it receives in the long run, rather than maximising its immediate reward. By changing its policy about how to choose an action at a given state as it interacts with the environment, the agent learns the optimal policy to achieve its goal of maximising the overall rewards. To summarise, in RL, the learning agent explores through the state space, interacting with the environment (receiving a reward) and adjusting its behaviours (choosing a different action at a given state) according to the experiences, in order to achieve a goal. Therefore RL problems are formulated by three factors: states, actions and a rewarding scheme.

One of the advantages of RL in implementing the individual skills aimed at in this research is the simplicity in formulation. The three factors mentioned above can easily be defined for these problems. It is not simple, however, for other ML methods. Solving the shooting problems with evolutionary learning, for example, would require the researcher to define decision variables first, and to explicitly formulate the fitness (or performance) function of the decision variables. To relate the performance and the decision variables would be very complicated, if not impossible. Especially in the problems at hand, the good result is often delayed, which makes the formulation even harder.

More importantly, RL does not require training data like supervised learning methods. As mentioned by Sutton & Barto (1998), it is often impractical to obtain examples of desired behaviour that are both correct and representative of all situations, which is also true for the problems to be handled in this research. In RL, the agent learns from its own experience rather than from training examples, and therefore it does not require such data.

2.2.1 Basic concepts of RL

The goal of RL is to maximise the overall rewards, more precisely, the expected *return*. It uses the notion of *value functions* to achieve this goal and the *Markov property* plays

an important role in the procedure. This section explains these important concepts in detail.

2.2.1.1 Returns

The return R_t is the sum of the rewards the learning agent receives after time-step t. If r_{t+1} is the reward given to the learning agent as a consequence of its action at time-step t, the return R_t is defined as

$$R_t = r_{t+1} + r_{t+2} + r_{t+3} + \cdots + r_T, \tag{2.1}$$

where T is a final time-step in *episodic* cases in which the interactions between the agent and the environment come to a natural end. In *continuing* tasks, on the other hand, the return R_t can be defined as

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}, \tag{2.2}$$

where γ is a *discount rate* ($0 \leq \gamma \leq 1$). The discount rate is used to convert future values to present values. To simplify, (2.3) is used henceforth to denote the return R_t for both episodic and continuing cases:

$$R_t = \sum_{k=0}^{T-t-1} \gamma^k r_{t+k+1}. \tag{2.3}$$

It is assumed that $\gamma = 1$ for episodic cases, and $T = \infty$ for continuing cases.

2.2.1.2 Markov property

An environment is said to hold the Markov property if the probability distribution of the state and response of the environment at time $t+1$ depends on the state and action at time t (s_t and a_t), regardless of the states that the agent passed through to reach the state at time t. That is, the environment has the Markov property if the dynamics of the environment satisfy

$$P(s_{t+1} = s, r_{t+1} = r \,|\, s_t, a_t) \tag{2.4}$$

$$= P(s_{t+1} = s, r_{t+1} = r \,|\, s_t, a_t, r_t, s_{t-1}, a_{t-1}, r_{t-1}, \ldots, r_0, s_0, a_0) \tag{2.5}$$

for all s, r, s_t and a_t.

Most of the RL tasks can be modelled as a *Markov Decision Process*, or MDP, where the environment satisfies the Markov property. If the states and actions are finite, the RL problem is called a finite MDP. In a finite MDP, the dynamics of the system can be described with two sets of values: 1) $\mathcal{P}^a_{ss'}$: the probability of a possible next state s' given a state s and an action a, and 2) $\mathcal{R}^a_{ss'}$: the expected value of the reward given any current state and action, s and a, along with any next state s'. They are defined as follows:

$$\mathcal{P}^a_{ss'} = P(s_{t+1} = s' \,|\, s_t = s,\, a_t = a), \tag{2.6}$$

$$\mathcal{R}^a_{ss'} = \mathbb{E}(r_{t+1} \,|\, s_t = s,\, a_t = a,\, s_{t+1} = s'). \tag{2.7}$$

It is said that the dynamics of the environment are completely known if $\mathcal{P}^a_{ss'}$ can be obtained for all $s \in S$ and $a \in A$. Henceforth, it is assumed that the RL problems are finite MDPs and the dynamics of the environment are completely known unless otherwise noted.

It is important for an RL problem to be modelled as an MDP because then the decisions and values can be determined only by the current state s, which is a solid basis of the RL algorithms described in following sections.

2.2.1.3 Value functions

To find an optimal policy, RL algorithms use the notion of *value functions* (Sutton & Barto, 1998). Value functions are functions of state (or functions of state-action pairs) that represent "how good the given state is" (or "how good it is to perform a given action in a given state), based on the rewards the agent can expect to receive in the future starting from that state (and choosing the given action) and following a policy π thereafter. Formally, the *state-value function* $V^\pi(s)$, the value of a state s under a policy π is defined as

$$V^\pi(s) = \mathbb{E}_\pi\{R_t \,|\, s_t = s\} \tag{2.8}$$

$$= \mathbb{E}_\pi\left\{\sum_{k=0}^{T-t-1} \gamma^k r_{t+k+1} \,|\, s_t = s\right\}, \tag{2.9}$$

where $\mathbb{E}_\pi\{\}$ denotes the expected value given that the agent follows policy π. Similarly, the *action-value function* $Q^\pi(s, a)$, the value of taking action a in state s, and thereafter

following a policy π, is defined as

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\{R_t \,|\, s_t = s, a_t = a\} \tag{2.10}$$
$$= \mathbb{E}_{\pi}\left\{\sum_{k=0}^{T-t-1} \gamma^k r_{t+k+1} \,|\, s_t = s, a_t = a\right\}. \tag{2.11}$$

An important property of value functions is that they satisfy the following recursive relationship known as *the Bellman equation* (Sutton & Barto, 1998):

$$V^{\pi}(s) = \mathbb{E}_{\pi}\{R_t \,|\, s_t = s\} \tag{2.12}$$
$$= \mathbb{E}_{\pi}\left\{\sum_{k=0}^{T-t-1} \gamma^k r_{t+k+1} \,|\, s_t = s\right\} \tag{2.13}$$
$$= \mathbb{E}_{\pi}\left\{r_{t+1} + \gamma \sum_{k=0}^{T-t-2} \gamma^k r_{t+k+2} \,|\, s_t = s\right\} \tag{2.14}$$
$$= \mathbb{E}_{\pi}\left\{r_{t+1} + \gamma V^{\pi}(s_{t+1}) \,|\, s_t = s\right\} \tag{2.15}$$
$$= \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma \mathbb{E}_{\pi}\left\{\sum_{k=0}^{T-t-2} \gamma^k r_{t+k+2} \,|\, s_{t+1} = s'\right\}\right] \tag{2.16}$$
$$= \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V^{\pi}(s')]. \tag{2.17}$$

Here $\pi(s,a)$ denotes the probability of choosing action a in state s under the policy π. The Bellman equation shows the relationship between the value of a state s and the values of all possible following states of s.

2.2.2 Algorithms to solve RL problems

Solving an RL problem is to find the best action for each possible state in which the learning agent might find itself in. In other words, the aim is to find an optimal policy. This section presents several RL algorithms to find an optimal policy, some of which are used in Chapter 6.

2.2.2.1 Policy iteration: policy evaluation and policy improvement

If $V^{\pi}(s)$ and $Q^{\pi}(s,a)$ are known for all $s \in S$ and $a \in A$ for a given deterministic policy π, then it is possible to find a new policy π', which is as good as, or better than π, by choosing an action

$$a' = \pi'(s) = \arg\max_a Q^{\pi}(s,a) \neq \pi(s) = a \tag{2.18}$$

for each state s. That is, for a given policy π, a better, or equally good policy π' can always be obtained by making it *greedy* based on $V^\pi(s)$, i.e., by choosing the best action a at each state s based on $V^\pi(s)$, unless the policy π is already an optimal policy (π^*). This is called *policy improvement*.

If the dynamics of the environment are known, (2.17) is a system of $|S|$ linear equations and $|S|$ unknowns (the $V^\pi(s)$, $s \in S$). The unique solution of this system is the true value $V^\pi(s)$ for all $s \in S$. Computing $V^\pi(s)$ directly, however, is often impractical, especially in problems with large state spaces. In RL, value functions are *estimated* using iterative methods. The value $V^\pi_{k+1}(s)$, the estimation of $V^\pi(s)$ at the $(k+1)^{th}$ iteration, is defined as

$$V^\pi_{k+1}(s) = \mathbb{E}_\pi\{r_{t+1} + \gamma V^\pi_k(s_{t+1}) \mid s_t = s\} \tag{2.19}$$

$$= \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V^\pi_k(s')], \tag{2.20}$$

with an arbitrarily chosen initial estimation of V^π_0. This update rule is from the Bellman equation (see (2.15) and (2.17)). It is known that V^π_k converges to V^π as $k \to \infty$ under the condition that either $\gamma < 1$ or the events are episodic (Sutton & Barto, 1998). This way of estimating value functions, i.e., the repeated application of (2.20) to convergence, is called *policy evaluation*. Sutton & Barto (1998) also said an optimal policy can be obtained then by alternating policy evaluation and policy improvement:

$$\pi_0 \xrightarrow{E} V^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V^{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \cdots \xrightarrow{I} \pi^* \xrightarrow{E} V^*, \tag{2.21}$$

where $\xrightarrow{E}$ and $\xrightarrow{I}$ denote policy evaluation and policy improvement respectively. This algorithm is called *policy iteration*. Algorithm 1 shows a pseudo-code for policy iteration. Note that the outer sum in (2.20) is not needed in the policy evaluation section in Algorithm 1 as the algorithm deals with a deterministic policy π.

2.2.2.2 Value iteration

Value iteration is a more efficient way of finding an optimal policy. The policy iteration process can be protracted because each policy evaluation is an iterative calculation that may take long and it should be done repetitively for each improved policy. Value

Algorithm 1 The policy iteration algorithm (Sutton & Barto, 1998).

1: **#Initialisation**
2: $V(s) \in \mathbb{R}$ and $\pi(s) \in A$ arbitrarily, for all $s \in S$
3:
4: **#Policy evaluation**
5: **repeat**
6: $\quad \Delta \leftarrow 0$
7: $\quad$ **for** each $s \in S$ **do**
8: $\quad\quad \mathfrak{v} \leftarrow V(s)$
9: $\quad\quad V(s) \leftarrow \sum_{s'} \mathcal{P}_{ss'}^{\pi(s)}[\mathcal{R}_{ss'}^{\pi(s)} + \gamma V(s')]$
10: $\quad\quad \Delta \leftarrow \max(\Delta, |\mathfrak{v} - V(s)|)$
11: $\quad$ **end for**
12: **until** $\Delta < \delta$ (a small positive number)
13:
14: **#Policy improvement**
15: *policy_stable* $\leftarrow$ *true*
16: **for** each $s \in S$ **do**
17: $\quad b \leftarrow \pi(s)$
18: $\quad \pi(s) \leftarrow \arg\max_a \sum_{s'} \mathcal{P}_{ss'}^{a}[\mathcal{R}_{ss'}^{a} + \gamma V(s')]$
19: $\quad$ **if** $b \neq \pi(s)$ *policy_stable* $\leftarrow$ *false*
20: **end for**
21:
22: **if** *policy_stable* stop
23: **else** go to line 4

iteration combines these two processes in one:

$$V_{k+1}(s) = \max_a \mathbb{E}\{r_{t+1} + \gamma V_k(s_{t+1}) \mid s_t = s, a_t = a\} \tag{2.22}$$

$$= \max_a \sum_{s'} \mathcal{P}_{ss'}^{a}[\mathcal{R}_{ss'}^{a} + \gamma V_k(s')]. \tag{2.23}$$

Algorithm 2 shows the value iteration algorithm. In value iteration, policy evaluation does not continue until convergence. It is stopped after one sweep of evaluation of each state, and policy improvement occurs immediately. According to Sutton & Barto (1998), the sequence V_k can be shown to converge to V^* under the condition that either $\gamma < 1$ or the events are episodic.

Algorithm 2 The value iteration algorithm (Sutton & Barto, 1998).

Initialise $V(s)$ arbitrarily, for all $s \in S$
repeat
    $\Delta \leftarrow 0$
    for each $s \in S$ **do**
        $\mathfrak{v} \leftarrow V(s)$
        $V(s) \leftarrow \max_a \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V(s')]$
        $\Delta \leftarrow \max(\Delta, |\mathfrak{v} - V(s)|)$
    end for
until $\Delta < \delta$ (a small positive number)

Output a deterministic policy π such that
    $\pi(s) = \arg\max_a \sum_{s'} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V(s')]$

2.2.2.3 TD algorithms for model-free problems

Thus far, we have discussed methods of solving RL problems when the dynamics of the environment are known. However, a more realistic application of RL is found when the dynamics of the environment are not known due to the stochastic nature of the environment. These kinds of problems are often called *model-free* problems.

In model-free problems, the agent should explore the environment to collect information about the dynamics of the environment. Given a state s, the agent takes an action a, and observes the reward r and the next state s'. It uses the observed reward r and $V(s')$, the estimated value of the next state s', to estimate the value of the current state s. The update rule is

$$V_{k+1}(s_t) = \eta\left[r_{t+1} + \gamma V_k(s_{t+1})\right] + (1-\eta)V_k(s_t) \tag{2.24}$$

$$= V_k(s_t) + \eta[r_{t+1} + \gamma V_k(s_{t+1}) - V_k(s_t)], \tag{2.25}$$

where η is known as the *learning rate* $(0 \leq \eta \leq 1)$.

The expected return over all possible next states is not used to estimate the value of the current state $V_{k+1}(s_t)$ as done in (2.19) and (2.20), because it cannot be obtained in model-free problems. Instead, the value $r_{t+1} + \gamma V_k(s_{t+1})$ is given from the experience the agent just went through. This is only an instance of many possibilities. The agent might have been moved to a different next state because the environment is stochastic. For this reason, instead of fully assigning $r_{t+1} + \gamma V_k(s_{t+1})$ to $V_{k+1}(s_t)$, the algorithm

Algorithm 3 The TD algorithm for estimating V^π (Sutton & Barto, 1998).

Initialise $V(s)$ arbitrarily, π to the policy to be evaluated
for all episodes **do**
    Initialise s
    repeat(for each step of episode)
        $a \leftarrow$ given by π for s
        Take action a, observe reward r, and the next state s'
        $V(s) \leftarrow V(s) + \eta[r + \gamma V(s') - V(s)]$
        $s \leftarrow s'$
    until s is terminal
end for

adds a part of what was just learned ($\eta[r_{t+1} + \gamma V_k(s_{t+1})]$) and throws away a part of what it currently has ($(1-\eta)V_k(s_t)$) (see (2.24)). Or it can be said that the value is updated towards the target ($r_{t+1} + \gamma V_k(s_{t+1})$) from where it is ($V_k(s_t)$) by adding a portion of the difference between the target and the current value ($\eta[r_{t+1} + \gamma V_k(s_{t+1}) - V_k(s_t)]$) (refer to (2.25)). This is called *Temporal-Difference* (TD) learning because it uses the difference between the target value and the current value of the state s_t, but the difference is effective only in that iteration, that is, in the k^{th} iteration. The target value $r_{t+1} + \gamma V_k(s_{t+1})$ is called the TD target or the backup value of s_t. Algorithm 3 shows the TD algorithm for evaluating $V(s)$ for a given policy π.

2.2.2.4 Q-learning

Algorithm 3 is used to evaluate the state-value function $V(s)$ for a given policy π, i.e., it is a policy evaluation algorithm. To find an optimal policy, either the policy iteration algorithm or the value iteration algorithm should be used, as shown in Algorithms 1 and 2, respectively. For model-free problems, Q-learning (Watkins, 1989), a value iteration algorithm using action-value functions $Q(s, a)$, is often used. The update rule is

$$Q_{k+1}(s_t, a_t) = Q_k(s_t, a_t) + \eta[r_{t+1} + \gamma \max_a Q_k(s_{t+1}, a) - Q_k(s_t, a_t)]. \tag{2.26}$$

Algorithm 4 shows the Q-learning algorithm. Watkins & Dayan (1992) proved that this algorithm converges to the optimal Q^* values as long as all state-action pairs are visited and updated infinitely. Also, it is known that the Q-learning algorithm approximates the optimal action-value function Q^*, regardless of the policy being followed

Algorithm 4 The Q-learning algorithm (Sutton & Barto, 1998).

1: Initialise $Q(s, a)$ arbitrarily,
2: **for all** episodes **do**
3: Initialise s
4: **repeat**(for each step of episode)
5: Choose a from s using policy derived from Q (e.g. ϵ-greedy)
6: Take action a, observe reward r, and the next state s'
7: $Q(s, a) \leftarrow Q(s, a) + \eta[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
8: $s \leftarrow s'$
9: **until** s is terminal
10: **end for**

when selecting an action a, given a state s, i.e., the policy used at line 5 in Algorithm 4. The ϵ-greedy method is often used for this purpose, where the best action is chosen based on the current action-value function $Q(s, a)$ with a probability of $1 - \epsilon$, and an action is selected randomly for a proportion of ϵ.

2.2.2.5 TD value iteration algorithm with state-value functions

In essence, Q-learning is a value iteration algorithm with action-value functions $Q(s, a)$ for model-free problems. Using action-value functions $Q(s, a)$ instead of state-value functions $V(s)$ requires more resources. Function values should be defined for all state-action pairs instead of for all states only. Obviously it will take more time to estimate optimal values $Q^*(s, a)$ for all state-action pairs than $V^*(s)$ for all states. Using a value iteration algorithm with state-value functions $V(s)$ may be an alternative, and this algorithm can be considered a *TD value iteration algorithm with state-value functions*. The update rule would look as follows:

$$V_{k+1}(s_t) = V_k(s_t) + \eta[\max_a \{r_{(t+1),a} + \gamma V_k(s_{(t+1),a})\} - V_k(s_t)], \tag{2.27}$$

where $r_{(t+1),a}$ represents the reward given to the agent as a result of action a at time-step t, and $s_{(t+1),a}$ denotes the next state (the state at time-step $(t + 1)$) when the action chosen at time-step t was a. For the sake of simplicity, $r_{(t+1),a}$ and $s_{(t+1),a}$ are henceforward denoted by r_a and s'_a, respectively.

The update rule (2.27) uses the maximum TD target $\mathfrak{v} = \max_a [r_a + \gamma V(s'_a)]$ over possible actions. Ideally the maximum value should be obtained after considering all possible next states s' given a state s and an action a. In model-free problems, however,

Algorithm 5 The TD value iteration algorithm with state-value functions.

Initialise $V(s)$ arbitrarily
for all episodes **do**
    Initialise s
    repeat(for each step of episode)
        for all possible action a **do**
            Take action a, observe reward r_a, and the next state s'_a
        end for
        $\mathfrak{v} \leftarrow \max_a [r_a + \gamma V(s'_a)]$
        $a \leftarrow \arg\max_a [r_a + \gamma V(s'_a)]$ with ϵ-greedy
        Take action a and observe the next state s'
        $V(s) \leftarrow V(s) + \eta[\mathfrak{v} - V(s)]$
        $s \leftarrow s'$
    until s is terminal
end for

the reward r_a and the next state s'_a are *observed* by actually taking an action a given a state s. That is, the next state s'_a (and the reward r_a accordingly) could be a different one given the same state s and the same action a due to the stochastic feature of the environment. Therefore, it is not guaranteed that the empirical maximum TD target $\mathfrak{v} = \max_a [r_a + \gamma V(s'_a)]$ over possible actions, which is observed from an experience, represents the actual maximum value over all possible next states s' and over all actions a given a state s.

For the reasons discussed in the previous paragraph, the TD value iteration algorithm using state-value functions $V(s)$ (Algorithm 5) is seldom feasible for model-free problems. However, it plays an important role in this research. The rationale for the use of this algorithm is discussed in Chapter 5.

2.2.3 Function approximation

In the RL algorithms described in the previous sections, state-value functions $V(s)$ and action-value functions $Q(s, a)$ are presented as a lookup table storing a value for each state,s or for each state-action pair. This could be a problem if the number of states is large, or even worse, if the state variables are continuous. The size of memory to store large quantities of data, as well as the time to convergence, restricts the application of these algorithms. The value functions need to be represented *approximately* when the

state space is large or continuous. This is called *function approximation*.

2.2.3.1 Generalisation

In function approximation, the value functions are presented not as a table but by a set of parameters. In other words, the value of a state $V(s)$ and the value of a state-action pair $Q(s,a)$ are represented by a function approximator F with a parameter vector $\vec{z}$, as shown below:

$$V(s) = [F(\vec{z})](s), \tag{2.28}$$

$$Q(s,a) = [F(\vec{z})](s,a). \tag{2.29}$$

Now the problem of estimating $V(s)$ for each state s, or $Q(s,a)$ for each state-action pair, has been changed to the problem of searching for the parameter vector $\vec{z}$ that represents $V(s)$, or $Q(s,a)$, as precisely as possible. This is called *generalisation*, which is concerned with generalising the value of states, or the value of state-action pairs as a function of $\vec{z}$. This directly involves *Supervised Learning* (SL) (see Section 2.1), a primary topic studied in machine learning, in the domain of the problem.

In supervised learning, a set of input-output pairs is provided as a training data set or samples. The task is to learn a mapping from the input to the output so that the output can be predicted correctly for any input that is not included in the samples. It is said that the training data set is given by a supervisor, meaning that the outputs in the samples are suitable output values corresponding to the input values. This is why it is called supervised learning. Regression can be seen as an example of supervised learning (Alpaydin, 2010). In Figure 2.1, for example, blue dots represent the samples in a data set given by a supervisor, and the algorithm tries to learn a mapping (the red line) that best reflects the relations between the inputs and the outputs. In the RL problem of estimating $V(s)$ or $Q(s,a)$, the mapping is from the state variable s to the value of the state $V(s)$, or from the state-action pair (s,a) to the value of the state-action pair $Q(s,a)$. A major concern here is how to use the samples in the training data set to obtain a good approximation over the entire input set.

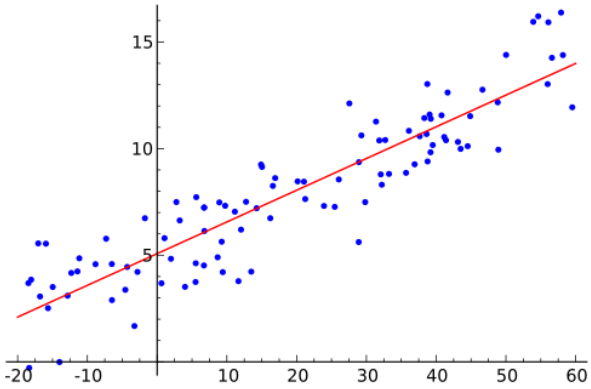


Figure 2.1: An illustration of linear regression on a data set. Blue dots represent samples in a data set given by a supervisor, and the red line shows the learned mapping from the data set.

2.2.3.2 Gradient-descent methods

Most supervised learning algorithms seek the parameters $\vec{z}$ such that the Mean Squared Error (MSE) is minimised. The MSE is defined by

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(t_i - y_i)^2, \tag{2.30}$$

where N is the number of samples in the training data set, t_i is the correct output of the i^{th} sample and y_i is the corresponding mapped output. As $t_i \in \mathbb{R}$ and y_i is a function of $\vec{z}$, MSE in (2.30) is also a function of $\vec{z}$. Therefore, searching $\vec{z}$ such that the MSE is minimised, is an optimisation (minimisation) problem of which the objective function is the MSE and the variables are the elements of the parameter vector $\vec{z}$. Because the mappings are not necessarily linear with respect to the parameters $\vec{z}$, solving this minimisation problem directly is often very hard. An iterative method is used alternatively to solve the problem, namely the gradient-descent method (Sutton & Barto, 1998). Gradient-descent methods are particularly well suited to RL algorithms in the sense that both are iterative methods.

In a gradient-descent method, each sample in the training data set is given to be evaluated, and the parameters $\vec{z}$ are adjusted by a small amount after each iteration in the direction that most decreases the error E observed in that iteration. Assuming that $V^{\pi}(s_t) \in \mathbb{R}$, the true value of state s_t, is given in the training data set for some

states $s \in S' \subset S$, the update rule for the problem of estimating $V^\pi(s)$ for all $s \in S$ is as follows:

$$\begin{aligned} \vec{z}_{t+1} &= \vec{z}_t - \frac{1}{2}\,\delta\,\nabla_{\vec{z}_t} E && (2.31)\\ &= \vec{z}_t - \frac{1}{2}\,\delta\,\nabla_{\vec{z}_t}[V^\pi(s_t) - V_t(s_t)]^2 && (2.32)\\ &= \vec{z}_t + \delta\,[V^\pi(s_t) - V_t(s_t)]\nabla_{\vec{z}_t} V_t(s_t), && (2.33) \end{aligned}$$

where $0 < \delta < 1$ is a step-size parameter, and $V_t(s_t)$ is the current mapping at time t. The error $E = [V^\pi(s_t) - V_t(s_t)]^2$ is given by the square of the difference between the true value of state s_t and the current estimation. $V_t(s_t)$ is a function of $\vec{z}_t$, and $\nabla_{\vec{z}_t} F(\vec{z}_t)$ indicates the gradient of function $F(\vec{z}_t)$ with respect to $\vec{z}_t$. If $\vec{z}_t$ is a n-dimensional vector $\vec{z}_t = (z_t(1), z_t(2), \ldots, z_t(n))$, then the gradient

$$\nabla_{\vec{z}_t} F(\vec{z}_t) = \left(\frac{\partial F(\vec{z}_t)}{\partial z_t(1)}, \frac{\partial F(\vec{z}_t)}{\partial z_t(2)}, \cdots, \frac{\partial F(\vec{z}_t)}{\partial z_t(n)} \right) \tag{2.34}$$

points to the direction in which the value of $F(\vec{z}_t)$ increases at the greatest rate. Therefore the negative gradient of the error $(-\nabla_{\vec{z}_t} E)$ in (2.31) shows the direction in which the error decreases most quickly.

2.2.3.3 Function approximation: RL combined with SL

The update rule discussed in the previous section ((2.31), (2.32) and (2.33)) cannot be applied in the problem of estimating $V^\pi(s)$ for all $s \in S$ because $V^\pi(s_t)$, the true value of state s_t, is not known. It is actually what the RL algorithms are searching for. Instead, the value is estimated by using the TD target of $V^\pi(s_t)$: $\mathfrak{v}_t = r_{t+1} + \gamma V(s_{t+1})$. This yields the TD gradient-descent function approximation for the state-value function:

$$\begin{aligned} \vec{z}_{t+1} &= \vec{z}_t + \delta[\mathfrak{v}_t - V_t(s_t)]\nabla_{\vec{z}_t} V_t(s_t) && (2.35)\\ &= \vec{z}_t + \delta[r_{t+1} + \gamma V(s_{t+1}) - V_t(s_t)]\nabla_{\vec{z}_t} V_t(s_t). && (2.36) \end{aligned}$$

To approximate the action-value functions, the following equation

$$\begin{aligned} \vec{z}_{t+1} &= \vec{z}_t + \delta[q_t - Q_t(s_t, a_t)]\nabla_{\vec{z}_t} Q_t(s_t, a_t) && (2.37)\\ &= \vec{z}_t + \delta[r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q_t(s_t, a_t)]\nabla_{\vec{z}_t} Q_t(s_t, a_t), && (2.38) \end{aligned}$$

where $q_t = r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a')$ is the TD target of $Q(s_t, a_t)$, can be used.

This is an exquisite combination of RL and SL algorithms. It is an RL algorithm because it tries to find $V^{\pi}(s)$ using the TD target as shown in (2.25), but it is different from the original TD algorithm in that $V^{\pi}(s)$ is represented as a function of parameters $\vec{z}$ and these parameters, not $V(s_t)$, are updated in each iteration. It is also an SL algorithm because it attempts to seek the parameters $\vec{z}$ to best approximate the input-output pairs in the training samples. It is distinguished, however, from the normal SL algorithms because the correct output values are not given by a supervisor but are calculated using TD targets as the agent explores the environment, which is the typical procedure of RL.

2.3 Multi-layer perceptron learning

In the previous section we have discussed a combination of RL methods and SL methods as a way to deal with problems that have large numbers of states or continuous state variables. Although, in theory, any supervised learning method could be used for handling these problems (Sutton & Barto, 1998), Artificial Neural Networks (ANNs) are a typical example of parametrised function approximators that can deal with nonlinear functions of the inputs (Busoniu *et al.*, 2010).

2.3.1 Artificial neural networks

An ANN is a machine that is designed to model the way in which the human brain performs a particular task or function of interest (Haykin, 2009). It is known that the human brain consists of a huge number of nerve cells, or neurons, which connect to each other to form neural networks. Each neuron can be seen as an information-processing unit which makes a simple decision: whether to fire or not. This is known as the "all-or-none" character of nervous activity. When it fires, an electrochemical pulse is generated and spreads to thousands of neurons that are connected to the firing neuron. Each neuron that accepts this electrochemical signal in turn makes its own decision about firing, based on the signal it received from the aforementioned neuron as well as signals from thousands of other neurons it is also connected to. McCulloch & Pitts (1943) tried to model the functioning of neural networks mathematically, which led to the development of ANNs. They presented a mathematical model of a neuron that has three basic elements (Marsland, 2011):

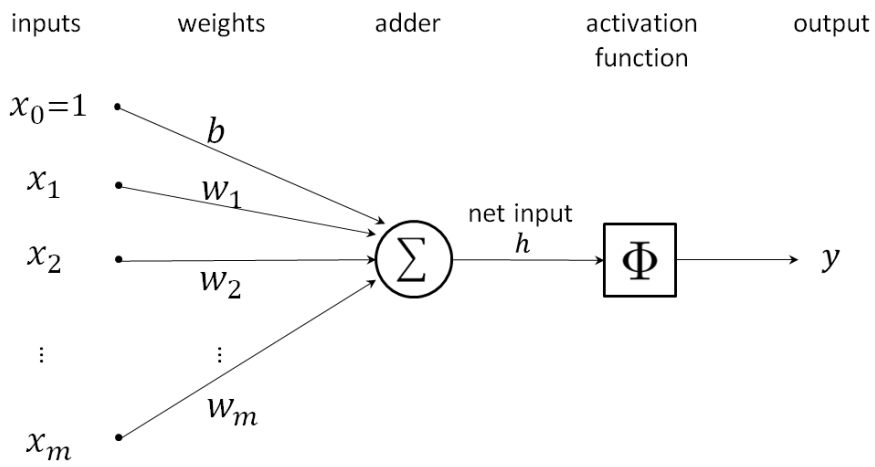
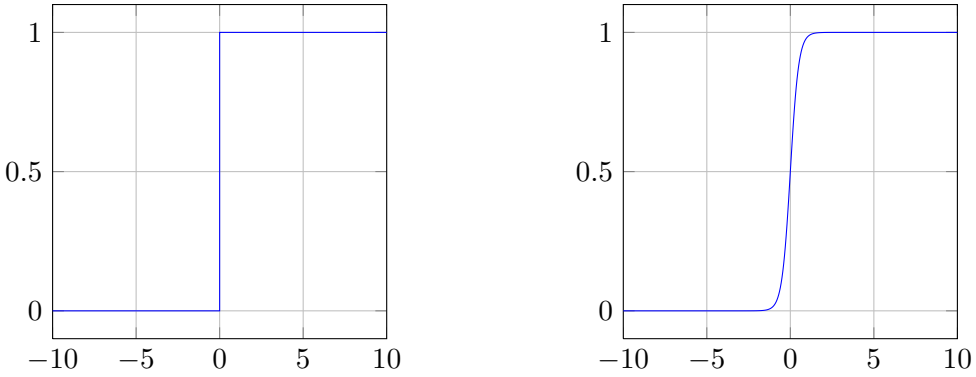


Figure 2.2: A neuron model. The neuron has m inputs $(x_1, x_2, \ldots, x_m)$, a bias input $x_0 = 1$, and an output (y). The adder calculates the sum of weighted inputs to form the net input $h = \sum_{i=1}^{m} w_i x_i + b$. The activation function accepts this value as its input and determines the output $y = \Phi(h)$.

- **a set of weights** denoted by w_i for the i^{th} input of the neuron
- **an adder** to sum the input signals
- **an activation function** that determines whether the neuron fires for the current inputs.

Figure 2.2 illustrates this mathematical model of a neuron. The neuron has m inputs $(x_1, x_2, \ldots, x_m)$, a bias input x_0 (which alway has a value of 1), and an output (y). The adder calculates the sum of weighted inputs to form the net input $h = \sum_{i=1}^{m} w_i x_i + b$. The activation function accepts this value as its input and determines the output $y = \Phi(h)$. A typical activation function that best represents the neuron's all-or-none character is shown in Figure 2.3(a). If the sum of weighted inputs is greater than or equal to zero, it fires. Otherwise it does not fire. This is called *the threshold activation function*. The mathematical form of the threshold activation function is

$$\Phi(h) = \begin{cases} 1, & \text{if } h \geq 0; \\ 0, & \text{if } h < 0. \end{cases} \tag{2.39}$$



(a) Threshold activation function.

(b) Sigmoid activation function.

Figure 2.3: Types of activation function.

Another form of activation function, called *the sigmoid function*, is preferred in our discussion for reasons that will become apparent in Section 2.3.4. Figure 2.3(b) shows an example of a sigmoid function, of which the graph is S-shaped. It looks reasonably similar to the graph of the threshold activation function, but increases smoothly. A common example of the sigmoid function is the logistic function. The mathematical form is

$$\Phi(h) = \frac{1}{1 + \exp(-ch)}, \tag{2.40}$$

where c is a positive parameter to indicate how quickly the function transitions from low values to high values. The bigger the parameter, the more the shape of the sigmoid function resembles that of the threshold function (Figure 2.3(a)).

2.3.2 Perceptron

The previous section described the general idea of ANNs. How can ANNs be used to solve problems or to learn? In order to answer this question, a more concrete notion of ANNs is discussed in this section: *perceptrons*. Technically, perceptrons are nothing more than a collection of McCullough-Pitts neurons (Marsland, 2011) connected in layers, but the importance is that it was proposed (by Rosenblatt, 1958, 1962) as the first model of learning with a teacher, i.e., supervised learning (Haykin, 2009). Figure 2.4 shows a model of a simple perceptron with $m + 1$ inputs (including the bias

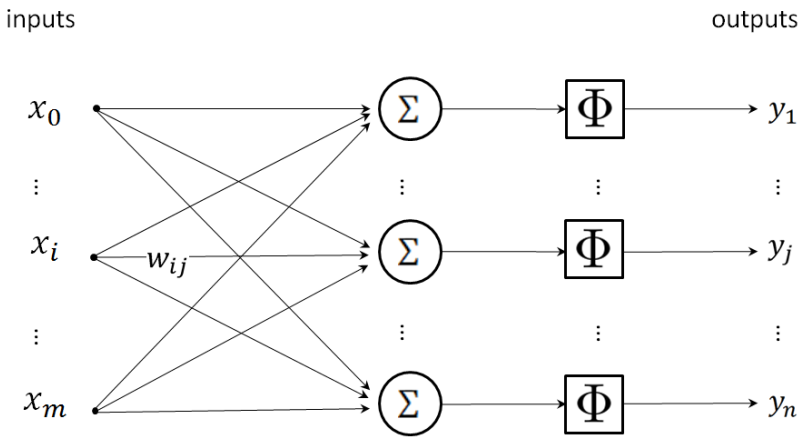


Figure 2.4: A single-layer perceptron model. It is assumed that the perceptron has $m+1$ inputs (including the bias input) and n outputs. The weight connecting input i and output j is denoted as w_{ij}.

input) and n outputs.[1] The inputs and outputs are connected by weights, and each output neuron has its own activation function. In Figure 2.4 the weight connecting input i and output j is denoted as w_{ij}. The model in Figure 2.4 is also called a single-layer perceptron to distinguish it from multi-layer perceptrons. Multi-layer perceptrons will be discussed in Section 2.3.3.

As mentioned in Section 2.2.3.1, in supervised learning a set of correct input-output pairs are provided as training samples and the problem is to learn a mapping that best describes the relationship between inputs and outputs to find the underlying distribution. In perceptron learning, this is done by *updating the weights* in the perceptron. Let x_i, y_j, t_j be the i^{th} input, the output of the j^{th} neuron, and the correct (desired) output (provided in the training samples) of the j^{th} neuron, respectively. In practice, t_j is often called the target. The weight connecting the i^{th} input x_i and the j^{th} neuron,

[1] There seems to have been some confusion regarding the concept of "perceptron". Perceptrons are introduced in most of the literature as the simplest form, that is, with one output neuron, insinuating that perceptrons are essentially the same as the McCullough-Pitts neuron. In this view, the ANN shown in Figure 2.4 would be a collection of n perceptrons rather than a perceptron with n output neurons. In a general sense, however, it would be more precise to say that a perceptron has one or more output neurons, which work together in the perceptron learning process.

w_{ij}, is updated as

$$w_{ij} \leftarrow w_{ij} + \eta(t_j - y_j) \cdot x_i, \tag{2.41}$$

where η $(0 < \eta \leq 1)$ is a learning factor, which is gradually decreased in time for convergence (Alpaydin, 2010). This update rule adjusts the weight so that the difference between the target and the output decreases. The reader is referred to Alpaydin (2010, p. 242) for an in-depth analysis of this matter.

Although the single-layer perceptron, which uses threshold activation functions in most cases, can be used to solve classification problems (Haykin, 2009), it applies only to those problems that are linearly separable due to the nature of the threshold activation function. It is possible for a single-layer perceptron with sigmoid activation functions to learn nonlinear mappings from the inputs to the outputs, but the extent of nonlinearity is significantly limited. Multi-layer perceptron learning attempts to address this issue.

2.3.3 Multi-layer perceptron

A Multi-Layer Perceptron (MLP) is an ANN that has one or more *hidden* layers between its input layer and output layer. Figure 2.5 shows a model of an MLP with two hidden layers. Each neuron is represented by a solid circle that includes the adder and the activation function. Neurons in the first hidden layer take the input of the network and produce outputs according to their activation functions. These outputs are fed to the neurons in the second hidden layer as their inputs, and the results are again taken by the neurons in the output layer. Lastly, the outputs of the neurons in the output layer are the final outputs of the network.

It is important to mention that each neuron in the MLP has a *sigmoid* activation function, not a threshold activation function. It is these nonlinear activation functions along with the existence of hidden layers that enables an MLP to solve complex tasks such as nonlinear regression (Haykin, 2009).

Learning occurs in the same way as the single-layer perceptron learning: by updating weights. The difference is that, due to the interdependence of weights, it is much more complicated to update the weights in order to decrease the difference between the target and the actual output of the network as done in (2.41) in the single-layer

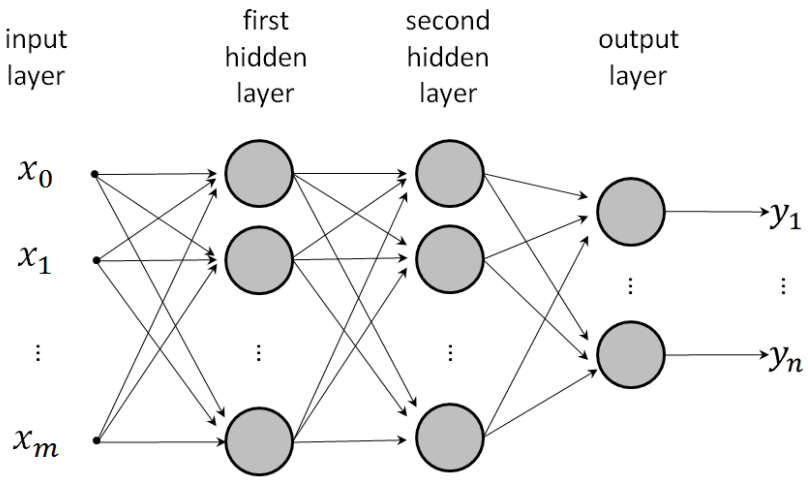


Figure 2.5: A multi-layer perceptron model. It is assumed that the multi-layer perceptron has two hidden layers with $m+1$ inputs (including the bias input) and n outputs. The number of neurons in the hidden layers are not specified in this figure.

perceptron learning. Rumelhart *et al.* (1986) has introduced an algorithm called *back-propagation* as a computationally efficient method to train MLPs (Haykin, 2009). The following section discusses this in detail.

2.3.4 Back-propagation algorithm

The Back-Propagation (BP) algorithm consists of two phases: a forward phase and a backward phase. In the forward phase, an input vector is fed to the network. This input signal goes through the network, layer by layer, until the output is produced. An error is calculated by comparing the target value of the input and the actual response of the network for the given input. This error signal is then sent backward through the network, again layer by layer, to be used in the updates of the weights. It is as if the error propagates from the output back to the inputs of the network, hence the name *back-propagation* (Alpaydin, 2010).

The BP algorithm uses the error-correction learning rule which is essentially the same as the ideas seen in Section 2.2.3.2. It tries to minimise the sum of squared errors by means of the gradient-descent method. In an MLP that has n output neurons, the

sum of squared errors E, with a scaling factor of $\frac{1}{2}$, is defined as

$$E = \frac{1}{2}\sum_{k=1}^{n}(t_k - y_k)^2, \tag{2.42}$$

where t_k denotes the target (desired) output of the k^{th} neuron and y_k is the actual output of the k^{th} neuron. We need to compute the gradient of the error function in (2.42) with respect to the weights so that we can update the weights in the direction that most decreases the error. For a given weight w_{ij}, which connects neuron i in a layer to neuron j in the next layer, the update rule is

$$w_{ij} \leftarrow w_{ij} - \eta\frac{\partial E}{\partial w_{ij}}, \tag{2.43}$$

where η is a learning factor ($0 \leq \eta \leq 1$).

There are two problems, however, in this approach. First, the threshold activation function is not differentiable because it is discontinuous. Secondly, it is not that simple to compute the gradient of the error function with respect to the weights when the network has one or more hidden layers (thus the network has more than one layer of weights). The use of the sigmoid activation function resolves the first problem. It looks similar to the threshold activation function (thus it acts like a neuron), but it is differentiable. For the second problem, the BP algorithm uses the chain rule of differentiation. The reader is referred to Haykin (2009, p.161–173) for a detailed explanation of how the algorithm uses the chain rule to update weights as well as for the derivation of the algorithm. In this thesis it is briefly illustrated by presenting a pseudo-code of the algorithm (Algorithm 6).

The code presented in Algorithm 6 is for the logistic activation function specifically, not for general sigmoid functions. Also, it is for an MLP with one hidden layer. It could easily be expanded for MLPs with more hidden layers, but in this discussion we keep it with one hidden layer for the sake of simplicity. It is assumed that the MLP has m input nodes, l neurons in the hidden layer and n output neurons. The notation used in this algorithm is summarised and illustrated in Table 2.1 and in Figure 2.6, respectively.

In Algorithm 6, the weights are updated after each training example is presented to the network. This is an application of the BP algorithm with *the sequential mode*. In the sequential mode of the algorithm, each training example is presented to the

Algorithm 6 The back-propagation algorithm (sequential mode) (Marsland, 2011).

#Initialise
$v_{ij} \leftarrow$ small random values for all j and k
$w_{jk} \leftarrow$ small random values for all j and k

repeat(for each training example)
    #Forward phase
        for $j = 1, \ldots, l$ **do**
            $h_j = \sum_{i=0}^{m} x_i v_{ij}$
            $a_j = g(h_j) = \frac{1}{1+\exp(-ch_j)}$
        end for
        for $k = 1, \ldots, n$ **do**
            $h_k = \sum_{j=1}^{l} a_j w_{jk}$
            $y_k = g(h_k) = \frac{1}{1+\exp(-ch_k)}$
        end for

    #Backward phase
        for $k = 1, \ldots, n$ **do**
            $\delta_k = c(t_k - y_k)y_k(1 - y_k)$
        end for
        for $j = 1, \ldots, l$ **do**
            $\delta_j = ca_j(1 - a_j)\sum_{k=1}^{n} \delta_k w_{jk}$
        end for

    #Update weights
        $w_{jk} \leftarrow w_{jk} + \eta\delta_k a_j$ for all j and k
        $v_{ij} \leftarrow v_{ij} + \eta\delta_j x_i$ for all i and j
until convergence

network, the error is calculated, and the weights are updated before the next training example is given to the network.

There is another way of applying the algorithm, called *the batch mode*, where the error is accumulated and averaged while a set of training samples is presented, and the weight update is performed once all the examples in the training set have been presented to the network. In the batch mode of the BP algorithm with N examples in a training sample set, for a MLP with n output neurons, the averaged sum of squared

Table 2.1: Notations used in the back-propagation algorithms. The notations in this table are for both sequential mode (Algorithm 6) and batch mode (Algorithm 7). Symbols with d in parentheses are used in the batch mode BP algorithm. They represent the same as their corresponding symbols in the sequential mode BP algorithm, except that the value is when the d^{th} example in the training data set is given to the network.

Notations	Meanings
v_{ij}	the weight connecting the i^{th} input node and the j^{th} neuron in the hidden layer
w_{jk}	the weight connecting the j^{th} neuron in the hidden layer and the k^{th} neuron in the output layer
x_i or $x_i(d)$	the input fed to the i^{th} input node
h_j or $h_j(d)$	the weighted sum of inputs for the j^{th} neuron in the hidden layer
a_j or $a_j(d)$	the activation of the j^{th} neuron in the hidden layer
h_k or $h_k(d)$	the weighted sum of inputs for the k^{th} neuron in the output layer
y_k or $y_k(d)$	the activation of the k^{th} neuron in the output layer
t_k or $t_k(d)$	the target output for neuron k in the output layer

errors over the training examples is defined as

$$E_{av} = \frac{1}{N}\sum_{d=1}^{N}\frac{1}{2}\sum_{k=1}^{n}(t_k(d)-y_k(d))^2 \tag{2.44}$$

$$= \frac{1}{2N}\sum_{d=1}^{N}\sum_{k=1}^{n}(t_k(d)-y_k(d))^2, \tag{2.45}$$

where $t_k(d)$ denotes the target value of the k^{th} output neuron in the d^{th} training example, and $y_k(d)$ is the actual output of the network for the output neuron k in the training example d. The batch mode BP algorithm uses (2.45) in its calculation of the gradient-descent of the error function. For the weights v_{ij} and w_{jk} as defined in Table 2.1, the update rules are as follows:

$$v_{ij} \leftarrow v_{ij} - \eta\frac{\partial E_{av}}{\partial v_{ij}}, \tag{2.46}$$

$$w_{jk} \leftarrow w_{jk} - \eta\frac{\partial E_{av}}{\partial w_{jk}}. \tag{2.47}$$

Algorithm 7 shows the pseudo-code of the batch mode BP algorithm for an MLP with the same structure shown in Figure 2.6. A new index d was added on top of the

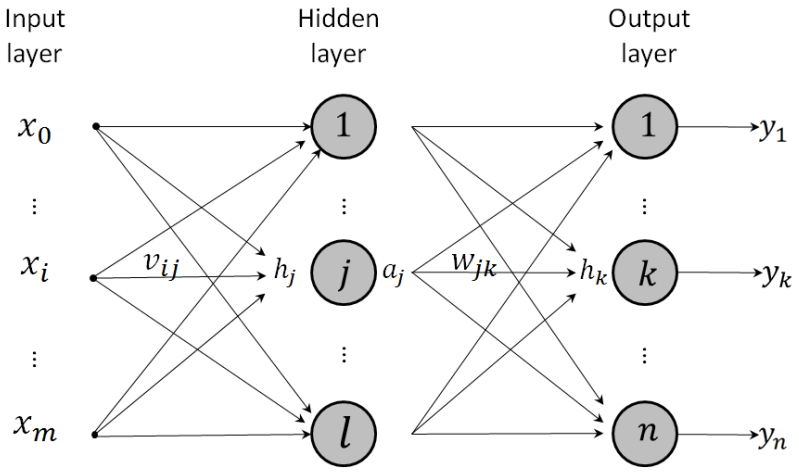


Figure 2.6: A multi-layer perceptron model used in the back-propagation algorithms. The multi-layer perceptron has $m+1$ inputs (including the bias input), a hidden layer with l neurons, and n output neurons. The symbol v_{ij} denotes the weight connecting the i^{th} input and the j^{th} neuron in the hidden layer, and w_{jk} represents the weight connecting the j^{th} neuron in the hidden layer and the k^{th} neuron in the output layer. The input of the j^{th} neuron in the hidden layer is defined by the weighted sum of the inputs $h_j = \sum_{i=0}^{m} v_{ij}x_i$, whereas a_j denotes the output of the j^{th} neuron in the hidden layer. Similarly, $h_k = \sum_{j=1}^{l} w_{jk}a_j$ is the input of the k^{th} output neuron in the output layer, and the activation of the output neuron is denoted by y_k.

notations used for the sequential mode of the algorithm, to represent each example in a training set of N examples. The notation used in this algorithm is shown in Table 2.1. The algorithm runs on a set-by-set basis until the weights converge. (Convergence conditions will be discussed later in chapters that deal with the experiments.) The batch mode BP algorithm has been used throughout the experiments performed in this research to develop the individual soccer skills mentioned in Section 1.4.3. These experiments will be dealt with in detail in Chapter 5 and Chapter 6.

Algorithm 7 The back-propagation algorithm (batch mode).

#Initialise
$v_{ij} \leftarrow$ small random values for all i and j
$w_{jk} \leftarrow$ small random values for all j and k
repeat(for each set of training examples)
  #Forward phase
    for $d = 1, \ldots, N$ **do**
      for $j = 1, \ldots, l$ **do**
        $h_j(d) = \sum_{i=0}^{m} x_i(d) v_{ij}$
        $a_j(d) = g(h_j(d)) = \frac{1}{1+\exp(-ch_j(d))}$
      end for
      for $k = 1, \ldots, n$ **do**
        $h_k(d) = \sum_{j=1}^{l} a_j(d) w_{jk}$
        $y_k(d) = g(h_k(d)) = \frac{1}{1+\exp(-ch_k(d))}$
      end for
    end for
  #Backward phase
    for $d = 1, \ldots, N$ **do**
      for $k = 1, \ldots, n$ **do**
        $\delta_k(d) = c(t_k(d) - y_k(d)) y_k(d)(1 - y_k(d))$
      end for
      for $j = 1, \ldots, l$ **do**
        $\delta_j(d) = c a_j(d)(1 - a_j(d)) \sum_{k=1}^{n} \delta_k(d) w_{jk}$
      end for
    end for
  #Update weights
    $w_{jk} \leftarrow w_{jk} + \eta \sum_{d=1}^{N} \delta_k(d) a_j(d)$ for all j and k
    $v_{ij} \leftarrow v_{ij} + \eta \sum_{d=1}^{N} \delta_j(d) x_i(d)$ for all i and j
until convergence

2.4 Conclusion: Machine learning

This chapter introduced the concept of Machine Learning (ML) focused on Reinforcement Learning (RL) and Multi-Layer Perceptron (MLP).

Several RL algorithms were investigated, of which the objectives are to find V^*, the state-value function for the optimal policy π^*, or Q^*, the action-value function for the optimal policy π^*. It was mentioned that it is often impractical to calculate V^* or Q^* directly in most problems due to the large number of states, therefore the optimal value functions V^* or Q^* are *estimated* in the RL algorithms.

MLP was discussed as a function approximator for the RL problems with continuous state variables. With a function approximator, the optimal value functions (V^* or Q^*) are represented as a function of a set of parameters $\vec{z}$. The purpose of function approximation is to find a set of parameters $\vec{z}$ that approximate V^* or Q^* as accurately as possible. For this purpose, MLP learning uses the back-propagation algorithm, which adopts an iterative method to *estimate* the parameters.

Thus, two types of estimations occurred in the RL with function approximation (one for the estimation of the optimal value functions V^* or Q^*, the other for the estimation of the parameters $\vec{z}$ to represent these value functions as accurately as possible). These two estimated values are influenced by each other and are updated as the iteration goes on until convergence.

Chapter 3

Related work

This chapter presents an overview of the literature regarding the development of strategies in robot soccer. The purpose is to provide the reader with current knowledge of the field, giving the context of robot soccer strategy development. The overview is divided into two sections. The first section describes how strategies have been developed in robot soccer, focusing on the RoboCup Small Size League (SSL), while the second explores the application of Machine Learning (ML) techniques to various soccer strategies in the robot soccer domain.

There are two major robot soccer organisations, namely the Federation of International Robot-soccer Association (FIRA) and the RoboCup. Although there are some differences in terms of mechanical requirements of the robots and the rules, both organisations have been working actively since they started in the late 1990s to encourage research in the field of artificial intelligence (AI) and robotics. The overview in this chapter is not necessarily limited to studies in the RoboCup SSL. It tries to encompass research based on both robot soccer domains, where applicable. In particular, the second section of the chapter (Section 3.2) includes many examples from the other RoboCup leagues that were mentioned in Section 1.1.2, and from FIRA, as the ML application is hardly ever found in the RoboCup SSL context.

3.1 The development of strategies in robot soccer

This section deals with the development of strategies in robot soccer by focusing on the processes and methods rather than on individual strategies. The procedures generally

taken to implement strategies are introduced, followed by the layered architecture commonly used in the structure of Decision-Making Modules (DMMs). The hand-coded approach is also discussed as the most prominent method in the implementation of strategies, along with the use of ML approaches that complement the former.

3.1.1 The general decision-making procedure

Early research in the robot soccer strategies tends to explain individual strategies in a given situation. Illustrations of how decisions were made on different levels of abstraction occupy most of the literature in the field. However, from the literature study of this research, it can be said that the general decision-making process in robot soccer follows the procedures below.

Step 1: Formation selection

In soccer, *formation* is a term used to indicate the arrangement of players on the field. For example, the 4-4-2 formation shows a deployment of a team with four defenders, four mid-fielders and two attackers. In the general decision-making process an appropriate formation is selected first for the team based on the current game state. This is to guide the team with overall strategies such as **AgressiveAttack**, **NaiveAttack**, **Defence**, etc. A formation defines roles, or positions, for the players in the team, and each position usually has its own region within which the player is supposed to remain.

Step 2: Role assignment

Once a formation is determined, the next step is to assign roles in the formation to each robot in the team based on the position of the ball and the robots.

Step 3: Positioning of each robot

Although each robot's responsible zone is decided in earlier steps, sometimes the exact position of a player is required. For example, when the team performs a pass, the position of the pass-receiving robot should be defined accurately.

Step 4: Action selection for each robot

Lastly, actions for each robot should be selected. One of the typical decision-making problems of this category is the decision of the player in possession of the ball, that is, whether to shoot, pass or dribble.

The decision-making in each step is typically performed according to a set of rules. That is, if the current game situation satisfies certain conditions, then corresponding actions are taken and carried out. For example, in step 4 of the general decision-making procedures above, rules for the robot in possession of the ball might be as follows:

```
If there is no opponent between me and the goal,
    then try to shoot.
Else check for all teammates from nearest to furthest
    if there is a clear path between me and the teammate,
        then pass to the teammate.
Otherwise
    dribble.
```

Strategies are defined by how these rules are set up and applied in each step. Various functions, indices and methods have been developed to produce different strategies. In this chapter, however, individual strategies are not discussed in further detail as the focus is to present the big picture of strategy development. The interested reader is referred to Table 3.1, which summarises individual strategies shown in the literature of the RoboCup SSL, and to the references in the table.

3.1.2 The hierarchical structure

The most distinguishing feature of strategy development in robot soccer is its hierarchical structure. The general decision-making procedures discussed in the previous section already show, albeit implicitly, the layered approach in the structure. The first and second steps determine high-level team strategies, while in the later steps, decisions are made for individual robots.

This is consolidated in the Skills, Tactics and Plays (STP) architecture, proposed by Browning *et al.* (2005), where the high-level team strategies are implemented as *plays* and the individual robot actions are further divided into two layers, namely *tactics* and *skills*. The STP architecture has provided a foundation for the hierarchical design of the DMMs in the RoboCup SSL domain. Before the appearance of the STP architecture, teams had implicitly been using the layered structure by following the general decision-making procedures. After the introduction of this architecture, most teams explicitly stated that their DMM is implemented in a hierarchical way inspired by the STP architecture (Chuengsatiansup *et al.*, 2009; Ishikawa *et al.*, 2011; Niknejad *et al.*, 2011; Panyapiang *et al.*, 2013; Poudeh *et al.*, 2013; Wu *et al.*, 2013b).

Table 3.1: Individual strategies used in the RoboCup SSL teams.

Steps	**Strategies shown in the literature**
Formation	– *play* selected using weight during the game for adaptation (CMDragons, Bowling *et al.*, 2004a) – *play* selected using expert system during the game for adaptation (CMDragons, Bowling *et al.*, 2004b) – *play* updated after the game to keep the score of each *play* of their own team (plasma-Z, Chuengsatiansup *et al.*, 2009)
Role assignment	– cost function (Cornell Big Red, D'Andrea, 2005) – manually decided at the beginning of the game (plasma-Z, Chuengsatiansup *et al.*, 2009) – potential field (RoboRoos, Ball & Wyeth, 2004; Tews & Wyeth, 2000)
Positioning	– SPAR algorithm for pass-receiving position (CMUnited, Veloso *et al.*, 1998) – pass position evaluation function for pass-receiving position (CMDragons, Zickler *et al.*, 2010) – safety region for the position of defence players (RoboDragons, Inagaki *et al.*, 2012; Maeno *et al.*, 2010) – potential field (RoboRoos, Ball & Wyeth, 2004; Tews & Wyeth, 2000)
Action selection	– obstacle-free-index to choose between shoot and pass (CMUnited, Veloso & Stone, 1998) – decision theoretic action selection to choose between shoot and pass (CMUnited, Veloso *et al.*, 1998) – local cost function (Cornell Big Red, D'Andrea, 2005) – dominant region to choose between direct play and 1-2-3 play (RoboDragons, Nakanishi *et al.*, 2010; Nakanishi *et al.*, 2008) – simple *if* statement to decide defensive actions (CMUnited, Veloso *et al.*, 1998) – potential field (RoboRoos, Ball & Wyeth, 2004; Tews & Wyeth, 2000)

3.1.3 Hand-coded vs machine learning approaches

In a hierarchical structure, whole game plans are implemented by a series of decisions. It is mentioned in Section 3.1.1 that these decisions are often made according to a set of rules. That is, the behaviours of the agents in each decision-making procedure are modelled by a set of action rules. In this regard, the DMM in robot soccer can be said to have a rule-based system. There are two basic approaches towards the implementation of these rules, namely the hand-coded approach and the machine learning approach.

In most cases, rules are established using human expertise. Strategies implemented through human knowledge is often called "hand-coded" in the literature. It is natural and inevitable to count on human knowledge to implement strategies in robot soccer, but it entails the unavoidable process of refinement. Numerous trials and errors are required to account for all potential game situations or missed possibilities. Also, the hand-coded, human-driven solutions are not guaranteed to be optimal as humans, even experts, are likely to be biased.

Machine Learning (ML) approaches have emerged to reduce manual efforts in the implementation of strategies and to improve the existing hand-coded algorithms. These approaches are actively adopted especially by teams in the RoboCup Simulation League, where two teams of 11 autonomous software agents each play soccer in a common simulation environment called Soccer Server (Noda *et al.*, 1998). Because it is liberated from the burden of hardware design and control, the main challenge of the Simulation League is to pursue ways to implement smart individual agents whose decisions and actions contribute to the completion of a successful soccer team in a fully distributed, partially observed, real-time environment.

The ML application is also found quite often in the studies based on the other RoboCup leagues or FIRA domain, but it is hardly used in the context of the RoboCup SSL. Being the simplest among the five RoboCup leagues, the RoboCup SSL has been served as the entry league for new teams. Hardware design and control are the most urgent priorities of each new team, and the highly dynamic environment of the RoboCup SSL increases the necessity of strong and competitive hardware. A faster robot is often more helpful than elaborated strategies. Consequently teams in the RoboCup SSL have concentrated on hardware design and control rather than on strategy development.

3.2 Machine learning applications in robot soccer strategies

This section presents a brief overview of the Machine Learning (ML) techniques applied to various soccer strategies. Many ML techniques have been widely used in this field, including Reinforcement Learning (RL), Case-Based Reasoning (CBR), Pattern Recognition (PR), Evolutionary Algorithms (EA), Artificial Neural Networks (ANNs), etc. Of these, RL is the most dominant technique for the reasons discussed in Section 2.2.

Reinforcement learning applications are analysed first in Section 3.2.1 and other ML techniques are reviewed in Section 3.2.2. Wu *et al.* (2013a) provide an overview of literature on the applications of various ML techniques. Section 3.2.2 is based on their work.

The overview is limited to research on soccer strategies only, although the application of ML is also found in many other areas in the robot soccer domain such as low-level control tasks (Oubbati *et al.*, 2005), computer vision (Budden *et al.*, 2013; Kaufmann *et al.*, 2005; Li *et al.*, 2003; Treptow & Zell, 2004), and to learn walking or kicking patterns for humanoid robots (Budden, 2012; Hausknecht & Stone, 2011; Ogino *et al.*, 2004).

3.2.1 Reinforcement learning applications

Rabiee & Ghasem-Aghaee (2004) and Farahnakian & Mozayani (2009) have applied Q-learning to implement a scoring policy for their Simulation League team, *UvA Trilearn*. Their main concern was how to make a decision for a robotic soccer agent, namely whether it should attempt to score or not in a given situation. Consequently the action space is quite small, containing only two actions—**ToShootTowardTheGoal** and **NotToShootTowardTheGoal**—but they used, in their state spaces, not only normal parameters such as the distance between the ball and the goalkeeper but also the probability of scoring obtained from the previous research of the team (Kok *et al.*, 2003).

Riedmiller and other researchers have presented, in a series of articles, their experiences of applying RL techniques for their Simulation League team, *Brainstormers*. They succeeded in applying RL techniques to some individual soccer skills such as kicking (Riedmiller & Gabel, 2007; Riedmiller *et al.*, 2001), intercepting (Gabel &

Riedmiller, 2006; Riedmiller & Gabel, 2007), and aggressive defence behaviour (Gabel *et al.*, 2009), as well as to a higher level of team cooperation to model the attacking strategy (Merke & Riedmiller, 2002; Riedmiller & Gabel, 2007; Riedmiller *et al.*, 2001). They used a value iteration algorithm in most cases, except for the high-level team strategy, where a variation of the Q-learning algorithm, called distributed Q-learning (Lauer & Riedmiller, 2000), was used.

Note that the development of kicking skills may seem closely related to the work conducted (and shown in Section 6.1) in this research. In fact, they are two different tasks. The work in Riedmiller *et al.* (2001) and Riedmiller & Gabel (2007) is to learn a good kicking routine when the ball is already in the kickable area of the learning agent. As the Soccer Server requires that harder kicks must be composed of a number of elementary kick commands for the learning agent to be able to kick hard enough for long passes or shootings, a number of kick commands should be applied along with turn commands to aim in the target direction. Therefore, their work is to learn a low-level kicking skill with parametrised kick and turn commands as the action scheme. The task is considered to be achieved if the ball leaves the kickable area with the specified velocity in the specified direction. On the other hand, in the work of this research, the task involves finding a good path towards the ball to achieve the object of the task, i.e., scoring a goal. The low-level kicking skill is simply implemented in this work by using a kick command when the ball is in the kickable area.

Riedmiller and other researchers used parametrised turns and dashes as the action scheme for other individual skills, while for the team-level attacking task, each agent is allowed to choose from 10 different actions, that is, to move in eight different directions, to go back to its home position, or to try to intercept the ball. In all cases, the RL-based skills or team behaviours showed some improvements compared to the corresponding hand-coded routines.

Other team behaviours often found in the literature of the Simulation League are **NaiveAttack** strategies. Stone *et al.* (2005) proposed a subtask domain of RoboCup soccer called *KeepAway* in this context. In the *KeepAway* domain, a team of *keepers* tries to maintain control of the ball in a certain designated area for as long as possible, while the opponents, called the *takers*, attempt to gain possession of the ball. These researchers focused on the learning of the keepers when in possession of the ball. The

keeper in possession of the ball can choose an action from **HoldBall**, **PassToTeammate*1***, ..., **PassToTeammate*n***. The strategies of the keepers without the ball, as well as those of the takers, are fixed and pre-specified. A number of experiments were done with varying numbers of players and different sizes of the field, and under random, hand-coded and learned policies. The results showed that the performance of the learned policy was in most cases better than the others except when the field is large (hence the task of keeping the ball for as long as possible is considerably easier).

Kalyanakrishnan *et al.* (2007) extended *KeepAway* soccer to a more complex task, *Half Field Offence*. With the field enlarged to half the size of a soccer field, the state space is larger and the action set is richer in this task. The purpose is to outperform a defence team to score a goal. As in *KeepAway*, the defence team follows a fixed strategy, as do the offence players without the ball, and the learning is only for the attackers when in possession of the ball. The actions are **PassToTeammate*1***, ..., **PassToTeammate*n***, **Dribble** and **Shoot**. On account of the nature of the task, **HoldBall** is replaced with **Dribble** and **Shoot** is added. Although the task is considerably harder than the *KeepAway* task, due to the larger state/action spaces and the sparse reward, the team successfully applied the $Sarsa(\lambda)$ algorithm—a variation of Q-learning—to the *Half Field Offence* task to deliver a satisfying result.

Neri *et al.* (2012) also applied RL techniques to the team-level attacking task, but at an even higher level—to the whole simulation game. Learning was conducted and continued until the game was completed. As in the *KeepAway* and *Half Field Offence* tasks, learning was applied only to the attacking player in possession of the ball. The researchers also confined learning to the cases when the learning agent is in the penalty area. For the other players or in the other cases, the team's hand-coded strategies were used throughout the game. It is worth mentioning that, in contrast to the states represented by measures such as distance to the ball, or angles toward the goal in the *KeepAway* and *Half Field Offence*, the state space in this task was compressed into several linguistic forms such as **AdversaryBehind** or **KickOpportunity**. The action space consists of seven actions, namely **Launch**, **SlowForward**, **FastForward**, **Pass**, **Dribble**, **HoldTheBall** and **Kick**. The researchers proved the superiority of the proposed approach by presenting the results of several matches played against different benchmarking teams in the Simulation League.

An application of RL to the full multi-agent system is found in the work of Duan *et al.* (2012). In contrast to the various cases mentioned above, in this work, learning is not limited only to the agent in possession of the ball, but each agent in the team is regarded as a learning individual to pursue a common goal. Learning is applied to the task of selecting an appropriate action from an action set consisting of six actions: **Shoot**, **Dribble**, **Pass**, **Intercept**, **Clear** and **Mark**. In order to achieve team cooperation and coordination, which are inevitable issues for problems in multi-agent systems, they introduced a method to predict the actions of other agents in the team. These predictions are considered by each learning agent when it chooses its own action.

This work is based on their previous work (Duan *et al.*, 2007), where layered learning is skilfully implemented in the building of their robot soccer team (*NewNeu*) for the Micro-Robot World Cup Soccer Tournament (MiroSot) in FIRA. They proposed a hierarchical learning system where the complex decision-making task is divided into multiple learning subtasks that include action implementation, role assignment and action selection. At the action implementation level, two fundamental actions were developed: shooting for the player in possession of the ball, and positioning for the other players. The development of shooting action is particularly relevant to the work performed in this research, which also investigated ways to develop a shooting policy with a stationary ball using RL. The difference is seen in the action schemes of the RL. The control variables in the work of Duan *et al.* (2007) are left/right wheel velocities because robots in the MiroSot have two wheels in design. Turning action is implemented by having different velocities between the left wheel and the right wheel. The ability of omnidirectional moving of the RoboCup SSL robots (refer to Section 1.1.3 and Figure 1.2(b)), however, allows the use of much richer action sets in this research, including turning while moving. In addition, different function approximators were used: Fuzzy Neural Network (FNN) in their work, multi-layer perceptron in this research.

In the multi-agent RL system where multiple agents learn simultaneously to achieve a common goal, the problem of assigning global reward to each agent inevitably arises. While Duan *et al.* (2012) used a fixed strategy for reward assignment, Liu *et al.* (2012) proposed a method to split the observed global reward among individual agents. In their approach, local reward for each agent was calculated based on the global reward

using a Kalman filter (Kalman, 1960), and then used to update the Q-value of each agent.

A few examples of RL applications are also found in the RoboCup Middle Size League (MSL) domain. Kleiner *et al.* (2003) proposed an approach that applies RL in a hierarchical way for their MSL team, *CS Freiburg*, in a simulation base. First, $Sarsa(\lambda)$ was applied to develop low-level skills such as **GoToBall**, **ShootGoal**, **DribbleBall**, etc., and then $Q(\lambda)$ was used for the selection of these skills in order to learn the task of **ShootingTowardsTheGoal**. Again, the skill **ShootGoal** in this work is about operating the kicker when the ball is in front of the robot, i.e., at a lower level than the shooting skills developed in this research. The task **ShootingTowardsTheGoal** is close to the work in this research, but the difference is again in the action scheme. The developed low-level skills were used in the work by Kleiner *et al.* (2003), while combinations of linear velocities and angular velocities were used in this research (see Section 5.3 for the action scheme used in this research). Kleiner *et al.* (2003) demonstrated that a good strategy of skill selection was learned after 500 episodes, although the performance did not exceed that of the hand-coded routine that was formerly used for the team. The comparable performance, however, being acquired by simulation, was achieved by far less time and effort for the design and parametrisation

The application of RL used directly on real robots is presented by Riedmiller *et al.* (2009) in the attempt of training their MSL robots to learn how to dribble. Learning on real robots is certainly expensive in terms of time and hardware wearing out, but as Riedmiller *et al.* (2009) pointed out, it has an additional advantage that the controller is directly tuned to the behaviour of the actual hardware, taking into account the real-world effects, which are usually extremely difficult to model.

3.2.2 Other machine learning applications

Case-Based Reasoning (CBR) (Aamodt & Plaza, 1994; Kolodner, 1992) is an approach of learning where new problems are solved based on the solutions of similar past problems, i.e., *cases*. An application of CBR is found in the rare literature of ML applications based on the RoboCup SSL. Marling *et al.* (2003) used CBR to help their RoboCup SSL team, *RoboCats*, in planning individual moves, team strategies and modelling the world. Three CBR prototypes were built in simulation mode where CBR was used

to position the goalkeeper, select team formations and recognize game states, respectively. They used the prototypes in predefined scenarios. The results were examined case by case by human experts. While the first prototype used for the position of the goalkeeper involved numerous problems and opportunities for improvements, the other two prototypes showed promising results.

Ros *et al.* (2009) shared their experience of applying CBR to develop cooperative action selection behaviours in the RoboCup Four-Legged League domain. In this league (currently replaced by the Standard Platform League), teams competed with identical (i.e., standard) robots, thus concentrating on software development only (RoboCup Standard Platform League webpage, 2014). They performed experiments, both in simulation and with real robots, in scenarios of two attackers versus two defenders. The focus was on the ability of the attacking robots to learn action selection behaviours to achieve success in the team's attacking situation. They evaluated the performance of the CBR-based approach compared to a benchmark method in terms of the number of goals scored by the attackers, the ball possession rate of the defenders, etc. The results showed that the CBR-based approach generally outperformed the benchmark one, both in simulation and with real robots.

Bianchi *et al.* (2009) used CBR combined with Heuristically Accelerated Reinforcement Learning (HARL) for similar tasks as Ros *et al.* (2009), also in the RoboCup Four-Legged League domain. HARL is an approach that uses heuristic functions for selecting appropriate actions to perform in order to guide exploration of the state-action space during the learning process (Bianchi *et al.*, 2008). By encoding some expertise in the heuristic functions, the learning process can be accelerated in HARL. In this work, CBR was used as the heuristic function. The experimental results indicated that the CBR-HARL approach is generally better than other approaches such as HARL (reported in their earlier publication (Bianchi *et al.*, 2008)), CBR alone, and traditional Q-learning.

The combined use of CBR and HARL has been extended to the *KeepAway* domain by Celiberto *et al.* (2012). They were particularly interested in ways to speed up RL by taking advantage of domain knowledge. In this regard, they adopted Transfer Learning (TL) techniques besides the use of CBR-HARL. The idea of TL is that experience gained in learning to perform one task can help improve learning in a related, but different, task (Taylor & Stone, 2009). They proposed an approach where CBR, HARL

and TL are combined in a way that the policy learned in one domain (source domain) is stored as a case base and then used in a new domain (target domain) as a heuristic. They modified the *Littman's Soccer* domain (Littman, 1994) in order to use it as the source domain where the soccer game, composed of two teams of two players each, was modelled as a Markov decision process in a very simple way. *KeepAway* soccer was used as their target domain. The experiments clearly indicated that the proposed approach reduced the convergence time by 90% compared to the traditional *Sarsa* algorithm. Also, it showed that the proposed approach achieved higher performance than the CBR-HARL approach without TL.

Another example of CBR combined with other ML techniques is seen in the work of Kuo *et al.* (2013). They developed a hybrid learning approach called Case-Based Reasoning with Genetic Algorithm (CBR-GA) for their Simulation League team, *WrightEagle*. In their work, learning was applied for the selection of high-level team strategies. Thus, the learning agent is the whole team, not individual players. CBR was used to choose from a set of team strategies which includes different formations such as 4-4-2 or 4-3-3. The GA was employed at the case retrieval stage, in which the CBR module tries to retrieve the most similar case from the stored past examples, to obtain more accurate cases. The experimental results showed that the proposed hybrid approach has better learning effect compared to other combined or single methods.

The Pattern Recognition (PR) technique is often used to interpret current situations of the game, or the behaviour of opponents in the robot soccer domains. It looks for patterns in the arrangement of the players on the field to draw meaningful conclusions on these subjects. Lattner *et al.* (2006) applied unsupervised symbolic learning to extract frequent patterns in dynamic scenes. Miene *et al.* (2004) proposed an approach to interpret spatio-temporal relations between two moving objects. Both studies used time-series to detect the changes of the dynamic scenes of the field, and represented the motions based on qualitative descriptions. Meanwhile Huang *et al.* (2003) proposed and implemented in the RoboCup Simulation League domain a mechanism to analyse the behaviours of opponent players. These researchers presented a plan representation scheme first, followed by techniques to automatically recognise and retrieve the plans of opponent players in the given plan representation.

The use of an Artificial Neural Network (ANN) is also found in the literature. Jolly *et al.* (2007) applied ANN to the 3-versus-3 MiroSot small league game. The purpose

is to decide which of the two non-goalkeeper players should go after the ball while the other player remains as a supporter. Their work is applicable only to the 3-versus-3 game structure due to the design of the ANN. Recently Chen *et al.* (2012) proposed an advanced neural network model that combines two special classes of ANN: Self-Organizing Map (SOM) and Learning Vector Quantization (LVQ). In the proposed model, SOM's ability to transform high dimensional input vectors into its neurons on a low dimensional topological structure without losing the core features of the input data, is used to initialise LVQ, which then carries out the supervised learning. Their work was applied to the task of strategy selection from the four categories of strategies, that is, **Attack**, **AttackPreparation**, **DefencePreparation** and **Defence**, and showed an effective learning ability in both off-line and on-line learning.

Some researchers adopted Evolutionary Algorithms (EA) to construct a proper rule selection scheme. As mentioned previously, in a rule-based system, rules describe which action the agent should take when it is in a certain state. Nakashima *et al.* (2004) used EA to develop such a rule set for acquiring team strategies in their RoboCup Simulation League team. The antecedent part of the rule set for each agent consists of 96 states based on the position of the agent and the relation to the nearest opponent. The consequent part of the rule set indicates the action the agent should take when the antecedent part of the action rule is satisfied. Ten different actions were predefined for the agent to choose from. The action rule for an agent is thus encoded by an integer string with 96 integers, each value in the interval $[1, 10]$ representing the action for the agent to take in a given state. They modelled the team strategy as a concatenated integer string with 960 integers, which serves as a chromosome in the EA, to encompass all 10 players in the team. For the operation of the EA, they used only mutations, i.e., no crossover operations were used. After a prespecified number of new integer strings have been generated by the mutation operation, the best integer strings are selected using the $(\mu + \lambda)$-strategy of evolution strategies (Back, 1996). Computer simulations were used to evaluate the performance of each integer string and the result showed that the attacking performance is improved during the execution of the EA.

While Nakashima *et al.* (2004) applied this technique for the team-level strategy, having actions such as **Dribble**, **Kick** or **Shoot**, Park *et al.* (2007) aimed to address a specific problem of robot posture (position and orientation) at the target position with emphasis on optimising the navigational path of the robot. This might be regarded as

a cornerstone to equip the agent with the shooting ability, because the final posture of the robot before shooting is of the utmost importance for the task. They used fuzzy logic for the rule selection scheme, and EA was employed to deal with the difficulty and tediousness of deriving fuzzy control rules. Their work was based on the MiroSot domain.

As seen in the previous paragraph, fuzzy theory is another approach often used in the rule-based system. Sng *et al.* (2002) used fuzzy logic to decide which robot should attack the ball. They considered four factors (distance to the ball, orientation, shooting angle and obstacles in the path) for each robot in the team. These factors are given to the fuzzy system, which then gives the priority order for the attacking role. Lee *et al.* (2005) also employed fuzzy logic for role assignment. In their work, the role assignment rule is predefined based on the positions of robots and the ball, and the fuzzy arbiter works to mediate two adjacent robots when their roles are in conflict. Wu & Lee (2004) presented a fuzzy mechanism, which selects an action for a robot from five categories: **Intercept**, **Shoot**, **Block**, **Sweep** and **StandBy**. They devised three factors (the defence factor, the competition factor and the angle factor) and used these as the fuzzy rules for action selection. All three works presented in this paragraph were implemented and tested in the FIRA domain, and the results showed that the presented fuzzy systems served their purposes.

3.3 Conclusion: Related work

This chapter contains the literature review of strategy development in robot soccer. The features of the strategy development process were presented in the first section. The decision-making procedures generally followed when the strategies are realised in the game were presented as well as the hierarchical structure of the DMM. It was also mentioned that the DMM is a rule-based system, where the decisions are made according to a set of action rules. Hand-coded approaches are typically used to implement these rules, but ML approaches are also used in smaller decision-making problems, where applicable, mostly to replace the existing hand-coded algorithms.

The use of ML approaches prevails in the RoboCup leagues or robot soccer domains (other than the RoboCup SSL) where strategy is of the utmost significance and advanced strategies are therefore inherently required. The second section of this chapter

reviewed these ML approaches that were applied to various decision-making problems. The review included not only the RL applications but also the applications of other ML techniques such as CBR, PR, ANN, EA and fuzzy theory.

It is worth mentioning that teams introduced in this chapter, such as *UvA Trilearn*, *Brainstormers* and *WrightEagle*, had achieved high rankings in recent competitions or at the time of the publication of their ML applications (RoboCup Simulation League Webpage, 2013). This confirms the effectiveness of the use of ML techniques in the development of robot soccer strategies.

Chapter 4

Simulation environment

In the last two chapters the theoretical background and literature study of this research were presented. This chapter introduces the simulation environment established in this study as the first practical stage towards the development of the Decision-Making Module (DMM). Grounds for the development of a well-designed simulator, and why existing simulators are not appropriate for this research, are discussed first. The focus is then moved to the real system that will be simulated in the simulation environment, followed by the architecture of the simulation environment, which includes the DMM and the simulator.

4.1 Why is a simulator necessary?

Nelson *et al.* (2001) defined simulation as the imitation of the operation of a real-world process or system over time. Sokolowski & Banks (2011) stated that simulation is engaged when the real system is inaccessible, dangerous or unacceptable, or when it simply does not exist. In this respect, it is obvious that a simulation is required to develop any kind of robot-related software, including the DMM. With simulation, the robot software development does not need to be delayed until real robots are available. Even if access to fully functional real robots is possible, the high risk of hardware damage during the software tests makes the use of real systems undesirable. Simulation also significantly decreases experimental time and cost.

A simulation always represents an abstracted view of the real system, and the level of abstraction depends on the nature of the work that needs to be done in the simulation.

In simulations with a high level of abstraction, a robot may be represented as a circle in a two-dimensional world, while simulations with a low level of abstraction attempt to simulate the operation of components of a robot such as sensors, actuators, motors, etc. The latter kind of simulations are often featured as *high-fidelity* in literature, and are typically used for the testing and development of low-level robot control software, which is run on the robot and controls the operation of the mechanical devices of the robot. In this research a high-level simulator is required to facilitate the development of the DMM.

Most existing robot simulators are designed for robot control software, having a low-level abstracted view of the world, and are therefore not appropriate for this study. Examples of these high-fidelity simulators are: SimRobot (Laue *et al.*, 2006), Player/Stage (Gerkey *et al.*, 2003), Gazebo (Koenig & Howard, 2004), Webots (Michel, 2004), USARSim (Carpin *et al.*, 2007) and UCHILSim (Zagal & Ruiz-del Solar, 2005). M-ROSE (Buck *et al.*, 2002) and Soccer Server (Noda *et al.*, 1998) can be classified as high-level simulators. However, being designed for the RoboCup Middle Size League (MSL) and the Simulation League respectively (see Section 1.1.2), they support only distributed control, which requires each robot in the team to make its own decisions individually. As centralised control is the control system aimed at in this study, the use of these simulators are not appropriate. In the context of the RoboCup Small Size League (SSL), two publications are found concerning simulators: Übersim (Browning & Tryzelaar, 2003) and grSim (Monajjemi *et al.*, 2012). Again, these are for low-level robot control to simulate the operation of the wheels, the kicker and other components, and are therefore not suitable.

As Beck *et al.* (2008) pointed out, the reuse of simulators developed by other teams is difficult, if not impossible, due to the diverse needs with regard to the level of abstraction. Their argument still seems true: nearly every team implements a simulation environment that is tailored to specific needs, like the hardware platform in use and the research focus of the particular team. This led to a decision that a high-level simulator is needed for the task aimed at in this research and the simulator cannot be adopted, but should be designed and developed by the team.

4.2 The control architecture of the RoboCup Small Size League

In order to build a simulation environment, the real system to be simulated should be discussed first. The SSL control architecture (the real system) is illustrated in Figure 4.1.

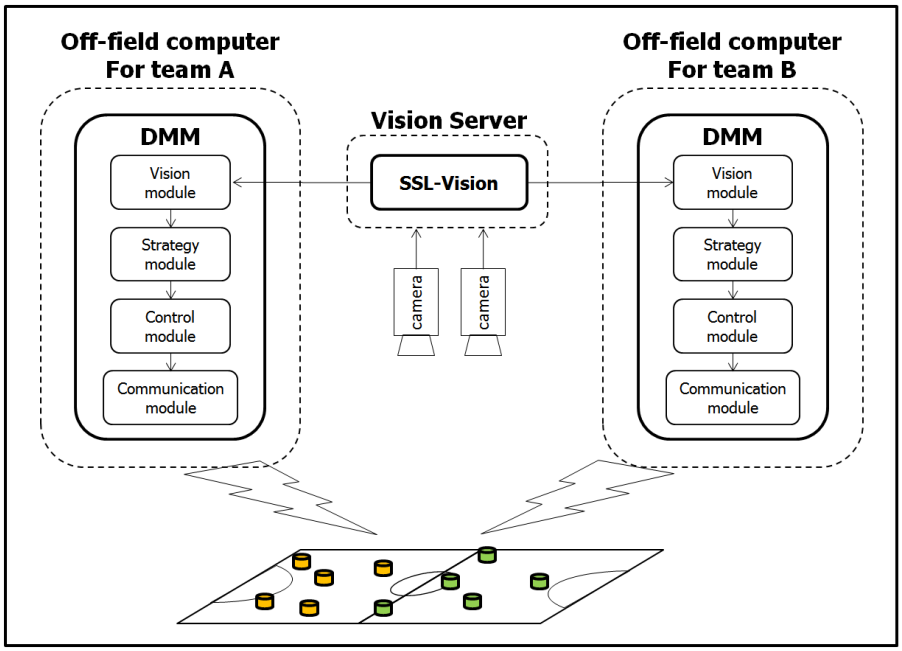


Figure 4.1: A typical SSL control architecture.

There are typically three main servers in the system: the vision server and two off-field computers. The vision server is a communal computer used for both teams and SSL-Vision, the image-processing program, is run on it. The off-field computers are for the teams participating in the game and each team's DMM is executed on each of the off-field computers. The tasks performed in the DMM can generally be categorised into four submodules, namely the vision module, the strategy module, the control module and the communication module.

The vision module receives information from SSL-Vision and performs calculations to predict the position of the robots and the ball for the current time-step. This is due to the latency of the image processing in SSL-Vision, which takes approximately 133 ms (Srisabye *et al.*, 2009). This could cause an error of up to 47 cm in the position

of the robots, assuming the fastest speed of 3.5 m/s usually reported in the literature. The error could be much more significant for a fast-moving ball.

The strategy module is where the game situation is checked, strategies are chosen and decisions are made. This module receives information from the vision module on the predicted position of the robots and the ball and calculates target destinations and actions for the robots based on its chosen strategies.

The control module deals with low-level control algorithms, such as path planning, obstacle avoidance and motion control. The communication module in the off-field computer sends the final commands of the DMM to the robots on the field in the form of translational and rotational velocities. The communication is wireless and typically uses dedicated commercial FM (Frequency Modulation) transmitter/receiver units (RoboCup SSL webpage, 2014).

4.3 The architecture of the simulation environment

As discussed in the previous section, the DMM is an independent computer program that works with the other components of the system shown in Figure 4.1. It continually communicates with the outside system, receiving inputs, processing them and giving outputs. Thus, to establish a simulation environment for the DMM, a counterpart of the DMM is required that represents the outside system, that is, the whole system except the DMM itself. The high-level simulator developed in this research plays the role of the counterpart.

The simulation environment in this research, therefore, consists of two subsystems, the simulator and the DMM, as represented in Figure 4.2. Details of the simulator are discussed in the next section (Section 4.4) as developing a simulator is one of the main objectives of this research. In this section, more general concepts of the two subsystems are presented under separate headings, focusing on the functions and roles of each in the simulation environment.

The simulation environment has been developed in C++ programming language with Microsoft Visual Studio as the development environment. Specifically, the Microsoft Foundation Class (MFC) was used as a base framework of the simulator and the DMM. MFC is an application framework for programming in Microsoft Windows. It provides portions of the Windows Application Programming Interface (API) in C++

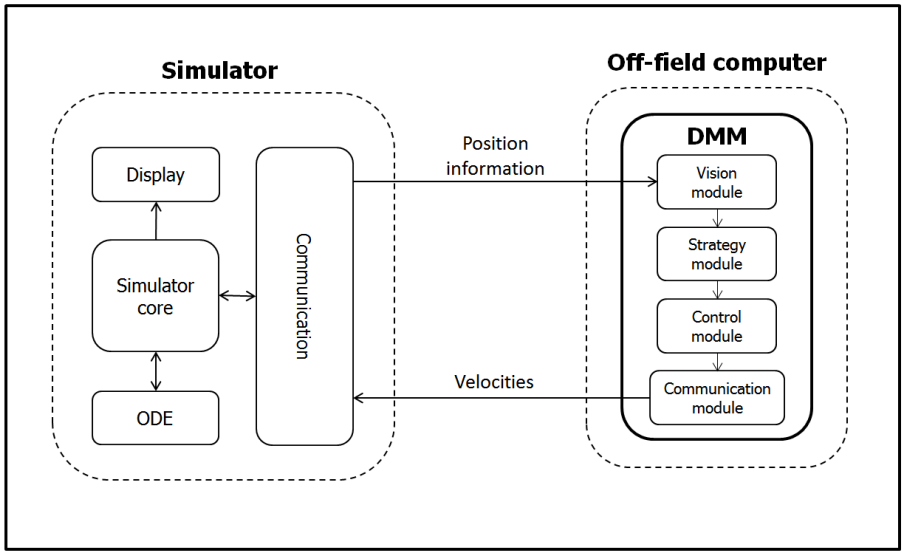


Figure 4.2: Overview of the simulation environment.

classes, thus helps developing applications in Windows in many ways. One of them is network programming. MFC provides Windows Sockets through which two or more applications, running on separate machines, can transmit data. The simulator and the DMM communicate with each other through this technology using TCP/IP (Transmission Control Protocol/Internet Protocol) as the network protocol.

The simulator

The simulator replaces the field and the objects on it, the cameras and SSL-Vision from the real world. It provides information on the position of the robots and the ball on behalf of SSL-Vision and receives the velocity commands from the DMM. More importantly, it accurately updates the position of the robots and the ball for the next time-step by moving the (virtual) robots according to the velocities delivered. The new information on the field is then passed back to the DMM. For convenient viewing and debugging, the simulator can also display the robots and the ball on the screen.

Extra care should be taken to ensure correct motion dynamics in cases where a robot hits or kicks the ball with a certain velocity or power. For this reason Open Dynamics Engine (ODE) (Smith, 2014a), a free industrial-quality library for simulating articulated rigid body dynamics (Smith, 2014b), was integrated into the simulator.

ODE enables not only accurate motion dynamics but also collision detection of two moving objects.

The decision-making module

The DMM in the simulation environment continuously communicates with the simulator. It receives the field information from the simulator and in return gives the desired velocities. Ideally these velocities should come from decisions carefully made by the strategy module. However, it should be mentioned that, at the time of writing, any submodule of the DMM does not exist (except the communication module). The development of the DMM, the strategy module in particular, is the long-term goal of this research. Consequently, issues that should be dealt with by other modules in the DMM, such as latency or obstacle avoidance, are not considered in the simulation environment. Some functions in the DMM, however, are still needed to establish the simulation environment. A framework of the DMM that receives the position information from the simulator and generates random velocity commands would be sufficient for this purpose. The communication module was implemented to perform these functions.

4.4 The simulator

In this section the simulator is discussed in detail. The simulator was developed with the possibility of being used by other teams. Therefore, almost all parameters and dimensions used in the simulator are modelled to allow customisation. Examples are the field size, the number of robots in each team,[1] the size and mass of the robots, the size and mass of the ball, the friction coefficient and the length of a time-step. The simulator can be used by other teams as far as TCP/IP is used for the network protocol between the simulator and their DMM, and the configuration of the communication packets (discussed in the last part of Section 4.4.2) is consistent.

[1]The number of robots in the simulator can be customised up to six, for the law (Laws of the RoboCup Small Size League, 2013) specifies a maximum of six robots in a team. The customisation of the number of robots is mainly for debugging and testing purposes.

4.4.1 Software design of the simulator

One of the most powerful functions of the MFC is the easy access to Graphic User Interface (GUI) programming. For this the MFC uses a Document/View architecture, where the data management is separated into two classes: the document class and the view class. The document class stores and manages the data, while the view class displays the data and manages user interaction with it (MFC Document/View Architecture, 2014).

The simulator is implemented based on this Document/View architecture. The document class and the view class were named SSL_SIMDoc and SSL_SIMView, respectively. The robots and the ball of which the position and velocity information form an important part of the data in the Document/View architecture, were modelled as classes, named CRobot and CBall respectively. These classes have the position and velocity information as their member variables. Another class called CWorld incorporates these two classes to make a complete representation of the world at a point in time.

When the simulator receives velocity commands from the DMM, the document class SSL_SIMDoc stores this information in the CWorld object for the current time-step, and calls an ODE function to obtain the position of the robots and the ball at the next time-step based on the velocity commands received. After storing the new position information in the CWorld object, SSL_SIMDoc sends a message to the view class SSL_SIMView for an update in the GUI. It then sends the new position information to the DMM through the Windows Socket object CClientSocket. This completes one cycle of procedures performed in SSL_SIMDoc. The next cycle starts with receiving velocity commands from the DMM after a predefined time-step, which can be customised, and set as 50 ms by default. Meanwhile, when it received a message from SSL_SIMDoc, the view class SSL_SIMView retrieves the CWorld object for the position of the robots and the ball, and shows it on the screen. Figure 4.3 shows the GUI of the simulator.

Figure 4.4 illustrates the software design of the simulator from a class point of view. Not all the classes in the simulator and their member variables and functions are indicated. Only those needed to show the main concept of the software design of the simulator are displayed. Part of the DMM was also included to show the communication between the simulator and the DMM.

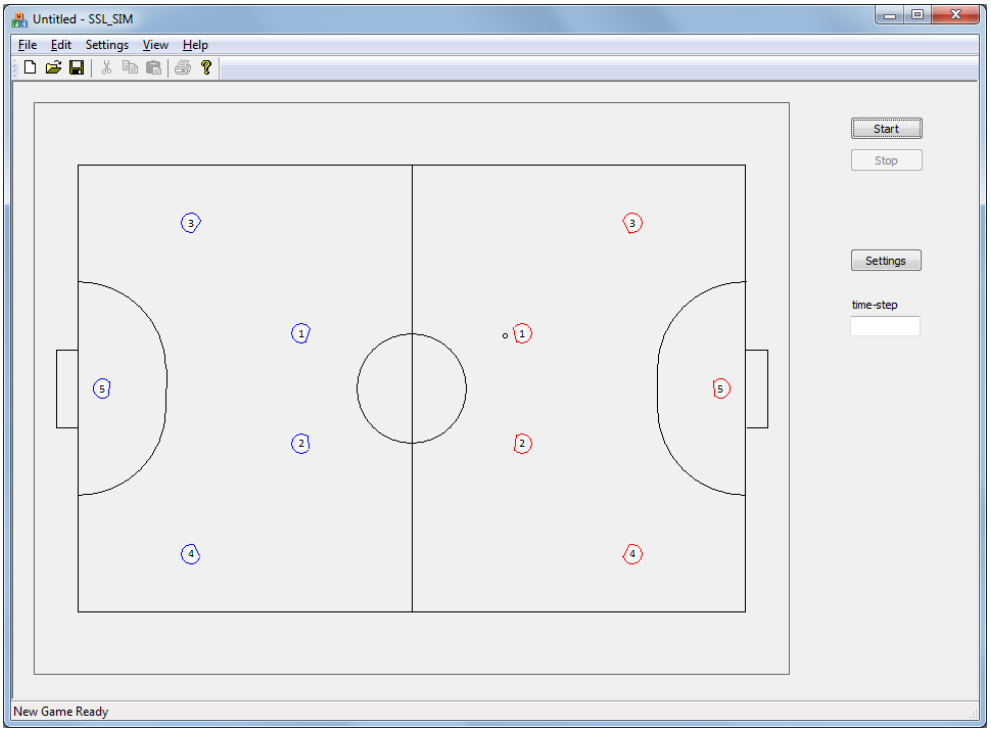


Figure 4.3: GUI of the simulator.

4.4.2 Implementation details

The simulator models the field, the robots and the ball. Details of the modelling of each object are presented next under separate headings. The dimensions and properties of the object are set by default in accordance with the laws of the RoboCup SSL (Laws of the RoboCup Small Size League, 2013). The dimensions and properties can be customised, as mentioned earlier, and thus can be modified in run time.

The field

The simulator provides a two-dimensional world where the field is represented in top view. Figure 4.5 shows the dimensions of the field designated by the laws of the RoboCup SSL. All dimensions are in millimetres. The field size is 6 050 mm × 4 050 mm, but the field surface is extended to 7 400 mm × 5 400 mm. As the laws require the field surface to be walled, the simulator also models a wall surrounding the field surface.

Figure 4.6 provides additional information about the field modelling in the simulator. It presents field coordinates and the names of some important areas of the field.

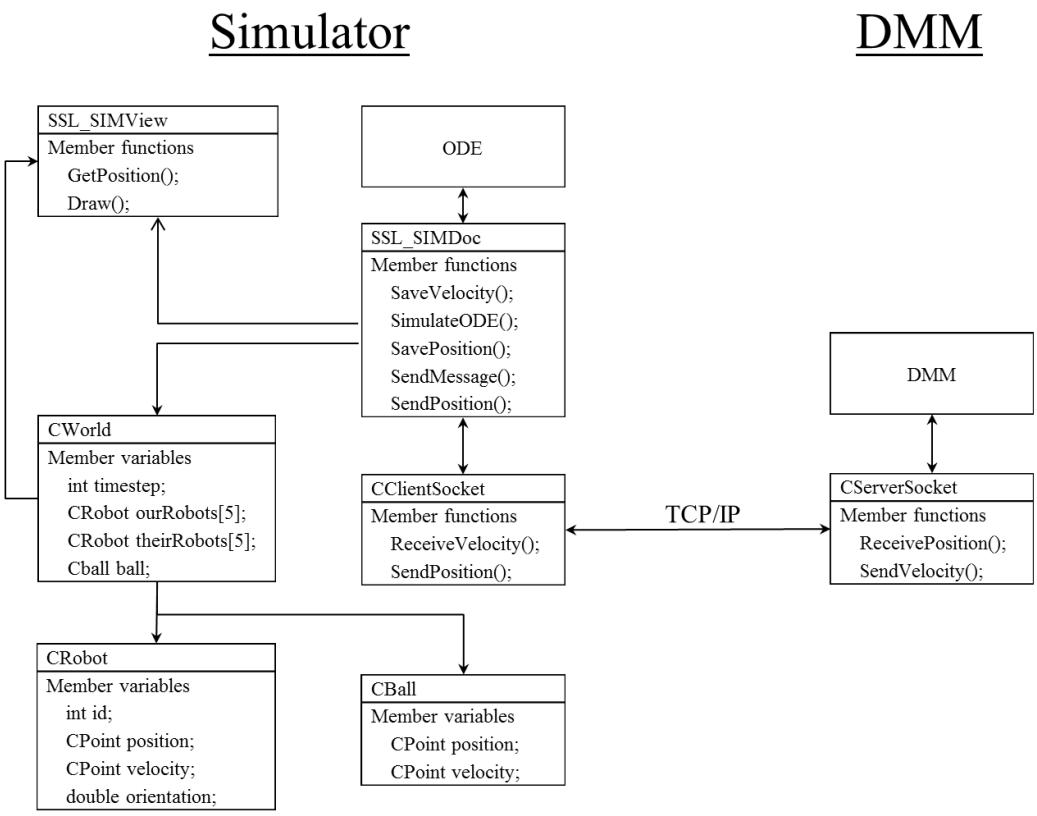


Figure 4.4: A schematic diagram of the simulation environment. Not all the classes and their members are shown in the figure. Only those important to the main concept of the software design are displayed. Part of the DMM was also included to show the communication between the simulator and the DMM.

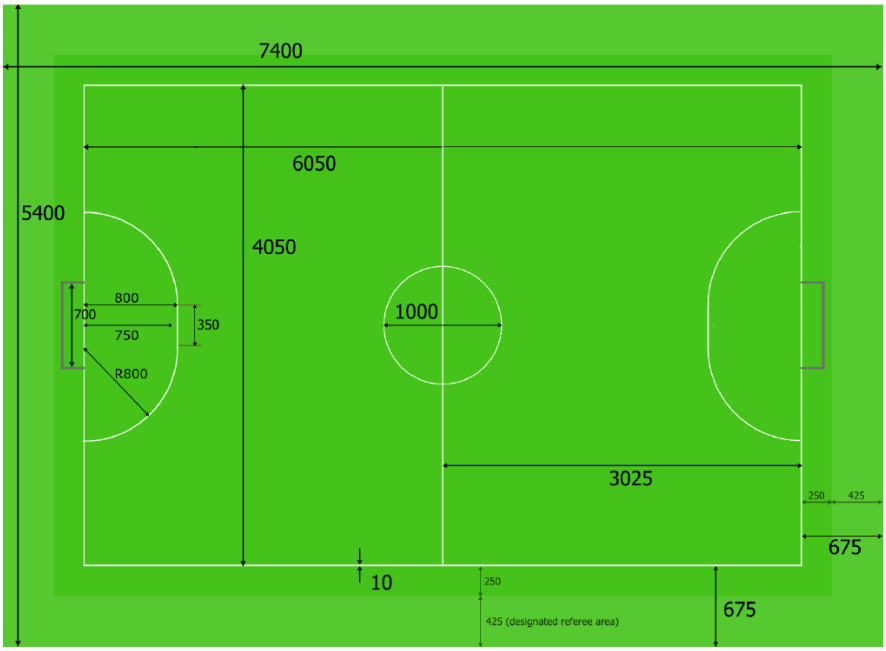


Figure 4.5: Field dimensions (Laws of the RoboCup Small Size League, 2013). All dimensions are in millimetres. The field size is 6 050 mm × 4 050 mm, and the field surface is extended to the size of 7 400 mm × 5 400 mm. The law requires the field surface to be walled.

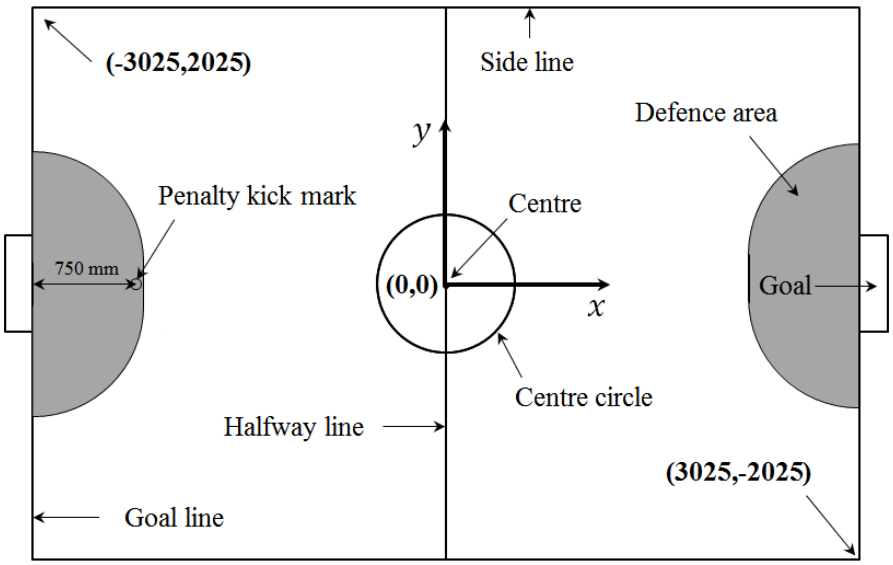


Figure 4.6: Field coordinates and important areas.

The Cartesian coordinate system was used in the field modelling with the origin at the centre of the field.

Robots

As the simulator being developed is a high-level one, a robot is modelled as a single object. It is implemented as a cylinder with a diameter of 180 mm and a height of 150 mm in the ODE context. The mass of the robots was assumed to be 2.4 kg as it ranges from 1.9 kg to 2.4 kg in the literature. Although the robot is modelled as a cylinder with a base of a whole circle, it is represented as an incomplete circle with a chord in the screen of the simulator, as seen in Figure 4.7. This is to show the robot's orientation clearly, which is important because the orientation of the robot is where the kicker is attached, and therefore determines the moving direction of the ball when kicked by the robot. The dark grey rectangle in Figure 4.7 shows the kicker unit attached to the front of the robot, and the light grey rectangle represents the kickable area, which is currently modelled as 60 mm × 90 mm. The size of the kickable area depends on the mechanical design of the kicker unit and they are adjustable for teams to modify according to their own designs.

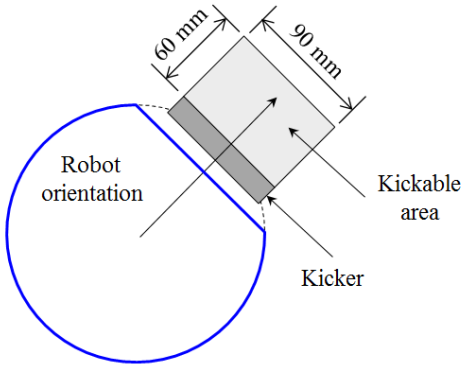


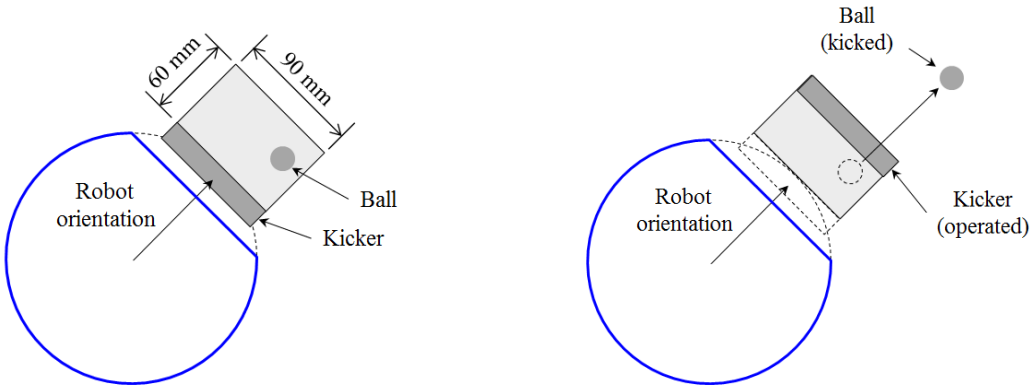
Figure 4.7: Robot orientation and the kickable area. The robot is represented as an incomplete circle with a chord (the blue object in the figure) in the screen of the simulator. This is only for a visualisation purpose. In all other functions of the simulator, including ODE, the robot is modelled as a cylinder with a base of a whole circle. The dark grey rectangle shows the kicker system attached in front of the robot and the light grey rectangle represents the kickable area.

The robot's acceleration is not modelled in the simulator. In reality, when receiving velocity commands from the DMM, the low-level robot control software that is run on the robot calculates and applies the acceleration to the device. Ideally the acceleration application mechanism should be analysed and incorporated in the simulator for a more realistic simulator. This is left for future work. It is assumed that the robots in the simulator are equipped with the commanded velocities at the next time-step.

The ball

The ball is modelled as a sphere with a diameter of 43 mm in the ODE context. The mass is set to 46 g as specified by the laws.

The force applied to the ball by the kicker is closely related to the mechanical design of the robot. In addition, different forces can be applied, depending on how hard the kick is. Therefore the number of forces available and the magnitude of the forces can be customised in the simulator. Currently it is modelled to have one level of force, which is 50N. Thus, if the robot operates the kicker and the ball is in the kickable area at that time, a force of 50N is exerted on the ball for 0.01 second in the direction of the robot's orientation. As a result, a stationary ball comes to move with a velocity of 10.869 m/s. Figure 4.8 shows the operation of the kicker and the ball's movement when kicked.



(a) The position of the ball and the kicker before the kicker is operated.

(b) The position of the ball and the kicker after the kicker is operated.

Figure 4.8: The operation of the kicker.

Friction

The Coulomb friction coefficient between the field surface and the objects on the field was set to 0.001, which means the friction in the simulator is almost negligible. Again, measuring the friction coefficient in an experiment with real robots and the ball and using this measured value would make the simulator more realistic. This is also left for future work. However, the friction coefficient can be customised in the simulator, thus it can be changed at any time.

Communication packets

This section describes the configuration of communication packets between the DMM and the simulator. The communication packets were designed at binary level, so that teams using other languages to develop their DMM can also use the simulator.

There are two types of communication packets: those the simulator receives from the DMM and those the simulator sends to the DMM. The communication packet the simulator receives from the DMM consists of a 2-byte header, a 48-byte body and a 2-byte footer. The body packet is divided into six 8-byte packets for the six robots in the team. The configuration of each 8-byte packet is described in Table 4.1.

Table 4.1: Communication packet for each robot.

	Description
1st byte	Robot ID
2nd–3rd bytes	Linear velocity in x-direction (in mm/s)
4th–5th bytes	Linear velocity in y-direction (in mm/s)
6th–7th bytes	Angular velocity (in degree/sec)
8th byte	Kick

The communication packet that is being delivered to the DMM from the simulator consists of a 2-byte header, an 88-byte body and a 2-byte footer. It carries the field information, in particular the position of the ball, and the position and orientation of the robots (including the opponent robots) in the field. Therefore the body packet consists of a 4-byte packet for the position of the ball, and twelve 7-byte packets for the information of the twelve robots in both teams. Table 4.2 shows the configuration of the 7-byte packet for the information of the robot.

Table 4.2: Communication packet for field information.

	Description
1st byte	Robot ID
2nd–3rd bytes	Position of the robot in x-coordinate (in mm)
4th–5th bytes	Position of the robot in y-coordinate (in mm)
6th–7th bytes	Robot orientation (in degrees)

For both types of communication packets, the body was configured with the assumption of six robots in a team. In cases that the number of robots in a team is customised in the simulator to have fewer than six robots in a team, the simulator considers as many bytes as needed after the first byte of the body packet. Also, in both types of communication packets, the header carries time-step information and the footer has a signal showing that the packet is completed.

4.4.3 Validation

This section provides validation of the developed simulator by assuming a few scenarios and performing a set of tests for each scenario. The scenarios are as follows:

- A robot moves with a certain velocity.
- A robot kicks a stationary ball.
- Two moving robots collide.
- The ball hits a wall and is reflected.

In the first two scenarios, five tests were performed with different initial positions, but the same velocities or forces were used for the sake of comparison. The distances the moving object travelled are measured at each time-step and compared with the values in theory. For the latter two scenarios, the trajectories of the moving objects are presented to check whether they are moving as expected.

Scenario 1: A robot moves with a certain velocity

In the first scenario, a robot is assumed to move by 1 m/s in the direction of $\frac{\pi}{6}$, measured anticlockwise from the direction of the positive x-axis (Figure 4.6).

Table 4.3: Position differences in Scenario 1.

		Theory	Test 1	Test 2	Test 3	Test 4	Test 5
Δx after	50 ms	43.3013	43.3013	43.3013	43.3013	43.3011	43.3013
	100 ms	86.6025	86.6027	86.6025	86.6025	86.6025	86.6025
	150 ms	129.9038	129.9038	129.9038	129.9038	129.9037	129.9038
	200 ms	173.2051	173.2052	173.2051	173.2051	173.2050	173.2051
	250 ms	216.5064	216.5063	216.5063	216.5063	216.5063	216.5063
	300 ms	259.8076	259.8076	259.8076	259.8076	259.8076	259.8076
	350 ms	303.1089	303.1089	303.1089	303.1089	303.1088	303.1089
	400 ms	346.4102	346.4102	346.4104	346.4102	346.4101	346.4102
	450 ms	389.7114	389.7115	389.7114	389.7114	389.7114	389.7114
	500 ms	433.0127	433.0128	433.0127	433.0127	433.0127	433.0127
Δy after	50 ms	25.0000	25.0000	25.0000	25.0000	25.0000	25.0000
	100 ms	50.0000	50.0000	50.0000	50.0000	50.0000	50.0000
	150 ms	75.0000	75.0001	75.0000	75.0000	75.0000	75.0000
	200 ms	100.0000	100.0000	100.0000	100.0000	99.9999	100.0000
	250 ms	125.0000	125.0000	125.0000	125.0000	125.0000	125.0000
	300 ms	150.0000	150.0001	150.0000	150.0000	150.0000	150.0000
	350 ms	175.0000	175.0000	175.0000	175.0000	175.0000	175.0000
	400 ms	200.0000	200.0000	200.0000	200.0000	200.0000	200.0000
	450 ms	225.0000	225.0000	225.0000	225.0000	225.0000	225.0000
	500 ms	250.0000	250.0000	250.0000	250.0000	250.0000	250.0000

Table 4.3 shows the difference from the initial position of the robot in x- and y-coordinates at each time-step. The shaded column shows the position differences calculated from the theory. The result shows the accuracy of the simulator in this task to the precision of the fourth decimal point.

Scenario 2: A robot kicks a stationary ball

In this scenario, a robot kicks a stationary ball in the same direction as in the first scenario and the force of 50N is applied to the ball. As mentioned earlier, the force causes the ball to move at approximately 10.869 m/s. The scenario starts with the ball placed in the kickable area of the robot.

Table 4.4 shows the distances the ball travelled in x- and y-direction after each time-step. The shaded column shows the theoretical travel distances. The result shows that the differences between the theoretical and the observed values are less than 3 mm in all cases. Also, no significant differences are found in the results of the five tests.

Table 4.4: Position differences in Scenario 2.

		Theory	**Test 1**	**Test 2**	**Test 3**	**Test 4**	**Test 5**
Δx after	50 ms	470.6660	470.6243	470.6242	470.6241	470.624	470.6243
	100 ms	941.3320	941.1941	941.194	941.194	941.1938	941.1941
	150 ms	1411.9979	1411.709	1411.709	1411.709	1411.709	1411.709
	200 ms	1882.6639	1882.171	1882.171	1882.171	1882.171	1882.171
	250 ms	2353.3299	2352.577	2352.577	2352.577	2352.578	2352.577
	300 ms	2823.9959	2822.93	2822.93	2822.93	2822.93	2822.93
	350 ms	3294.6619	3293.228	3293.228	3293.228	3293.228	3293.228
	400 ms	3765.3278	3763.472	3763.472	3763.472	3763.472	3763.472
	450 ms	4235.9938	4233.661	4233.661	4233.661	4233.661	4233.661
	500 ms	4706.6598	4703.796	4703.796	4703.796	4703.796	4703.796
Δy after	50 ms	271.7391	271.7012	271.7012	271.7012	271.7012	271.7012
	100 ms	543.4783	543.3481	543.3481	543.3481	543.3481	543.3481
	150 ms	815.2174	814.9406	814.9407	814.9407	814.9407	814.9406
	200 ms	1086.9565	1086.479	1086.479	1086.479	1086.479	1086.479
	250 ms	1358.6957	1357.963	1357.963	1357.963	1357.963	1357.963
	300 ms	1630.4348	1629.392	1629.392	1629.392	1629.392	1629.392
	350 ms	1902.1739	1900.767	1900.767	1900.767	1900.767	1900.767
	400 ms	2173.9130	2172.088	2172.088	2172.088	2172.088	2172.088
	450 ms	2445.6522	2443.354	2443.354	2443.354	2443.354	2443.354
	500 ms	2717.3913	2714.567	2714.566	2714.567	2714.566	2714.567

Scenario 3: Two moving robots collide

In the third scenario, the collision between two moving robots was tested. In the beginning of the scenario, two robots, a red and a blue one in Figure 4.9 are located at $(-200, 400)$ and at $(-200, -400)$, respectively. Both robots face towards the centre of the field and move forward with a velocity of 1 m/s.

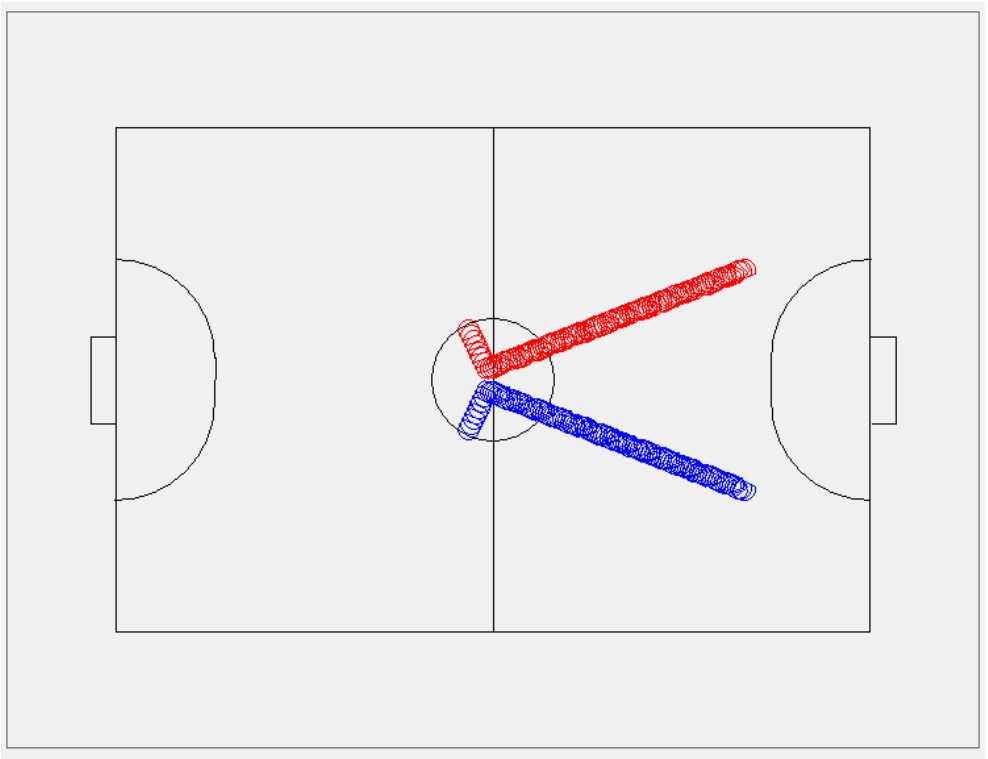


Figure 4.9: The collision of two moving robots. The red robot is initially located at $(-200, 400)$ and the blue robot at $(-200, -400)$. Both face the centre and starts moving forward at 1 m/s. After the collision the linear velocities of the robots are significantly reduced while they obtained angular velocities.

Figure 4.9 illustrates the trajectory of the robots before and after the collision. Note that the space between the positions at two consecutive time-steps is reduced after the collision, which means the velocities have been lessened due to the impact of the collision. Instead, the robots have acquired angular velocities, so the red robot has turned anticlockwise and the blue one clockwise.

Scenario 4: The ball hits a wall and is reflected

In the last scenario, a ball is located at $(-1\,000, 1\,700)$ in the beginning. It starts to move with a velocity of 3 m/s in the direction of $\frac{\pi}{4}$, as specified by the red arrow in Figure 4.10. The thick black lines around the field represent the walls. The ball continues to move until it hits a wall, which is located in the y-direction of the field, and then it bounces off the wall, as indicated by the blue arrow in Figure 4.10.

Figure 4.10 shows how the moving direction of the ball changes after the bounce-off. However, no significant changes were observed in the speed of the ball before and after the bounce-off.

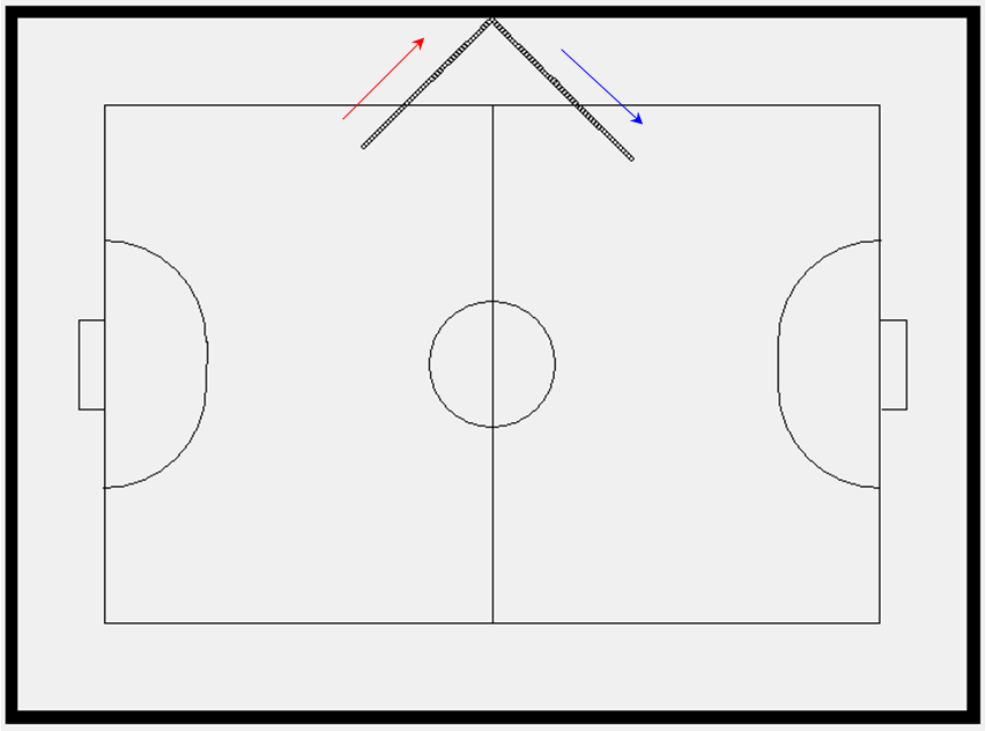


Figure 4.10: The trajectory of a moving ball when bounced off a wall. The ball is initially positioned at (-1000, 1700). It starts to move to the upper right direction (the direction pointed by the red arrow in the figure) with 3 m/s. The black thick lines around the field represent the walls. After the ball hits the upper wall, it changes the direction as pointed by the blue arrow in the figure. No significant changes in the speed are observed.

4.5 Learning simulator

In the previous section the simulator developed in this research was discussed in detail. Another kind of simulator, called the learning simulator, is needed to develop the individual soccer skills defined in Section 1.4.3 by using reinforcement learning. The reinforcement learning algorithms need to observe the next state s' when the learning agent takes an action a given a state s. The new state s' can be calculated without

difficulty when the learning agent is still approaching the ball, thus there is no collision between two moving objects in the environment. In cases that the action a causes a collision of any kind, the next state s' can be obtained from a simulator that uses ODE for such calculations. This is why a simulator is needed in the learning experiments, in addition to the visualisation purpose.

The learning simulator was developed separately from the simulator in the simulation environment. Many of the components of the simulator, however, were reused in the learning simulator, for example ODE and display-related functions. In addition, a new module was included to implement functions involved in the learning process.

4.6 Conclusion: Simulation environment

In this chapter the simulation environment established in this research in order to achieve the main objective of the study – the (partial) development of the DMM – has been discussed.

The necessity to build the simulation environment was addressed first, followed by the architectures of both the real system and the simulation environment. The simulation environment consists of the DMM and the simulator. The DMM forms the focal point of this research, aimed to be developed as a long-term goal, and the simulator is to help the development of the DMM. The simulator therefore has to be a high-level one in its abstracted view of the world, which was realised in the simulator built in this research. The simulator is discussed in detail in Section 4.4.

It was also mentioned that another simulator, called the learning simulator, was used additionally to develop the individual soccer skills mentioned in Section 1.4.3. Details of the developed soccer skills and the machine learning experiments performed to implement those skills are discussed in the next two chapters.

Chapter 5

Experimental design

The main object of this research is, as stated in Section 1.3, to lay a solid foundation for the Decision-Making Module (DMM) by establishing a simulation environment and implementing some of the basic building blocks of the DMM. The simulation environment was discussed in the previous chapter. This chapter and the following one will attend to the implementation of the basic building blocks of the DMM: individual soccer skills.

Shooting skills and passing skills were selected for the task, as mentioned in Section 1.4.3. These skills were developed using Machine Learning (ML) techniques, in particular, Reinforcement Learning (RL) with Multi-Layer Perceptron (MLP) as a function approximator. The rationale of the use of ML approaches as an alternative to the hand-coding approaches was mentioned in Section 3.1.3.

This chapter supplies the general information about the experiments and how the algorithms were applied in the experiments, while in the next chapter (Chapter 6), the experimental results are presented and discussed.

5.1 Experiments performed

The experiments performed in this research were aimed at implementing the six individual soccer skills mentioned in Section 1.4.3: 1) shooting a stationary ball, 2) shooting a moving ball, 3) shooting a stationary ball against a goalkeeper, 4) shooting a moving ball against a goalkeeper, 5) passing a stationary ball and 6) passing a moving ball. The third and fourth skills were further divided into several tasks depending on the behaviour of the goalkeeper. Table 5.1 summarises the skills and tasks to be learnt through

Table 5.1: Machine learning experiments performed to develop skills for different tasks.

Skill	Skill description	Task description	Task
Skill 1	Shooting a stationary ball	To learn to kick a ball to score a goal when the ball is placed in a random position	Task 1
Skill 2	Shooting a moving ball	To learn to kick a ball to score a goal when the ball is moving in a random direction	Task 2
Skill 3	Shooting a stationary ball against a goalkeeper	To learn to kick a ball to score a goal against a static goalkeeper when the ball is placed in a random position	Task 3
		To learn to kick a ball to score a goal against a dynamic goalkeeper when the ball is placed in a random position (the goalkeeper is only allowed to move along the goal line)	Task 4
		To learn to kick a ball to score a goal against a dynamic and smarter goalkeeper when the ball is placed in a random position (the goalkeeper is allowed to move off the goal line before the ball is moving)	Task 5
Skill 4	Shooting a moving ball against a goalkeeper	To learn to kick a ball to score a goal against a dynamic goalkeeper when the ball is moving in a random direction (the goalkeeper is only allowed to move along the goal line)	Task 6
		To learn to kick a ball to score a goal against a dynamic and smarter goalkeeper when the ball is moving in a random direction (the goalkeeper is allowed to move off the goal line before the ball is moving)	Task 7
Skill 5	Passing a stationary ball	To learn to kick a ball to pass to a target position when the ball is placed in a random position	Task 8
Skill 6	Passing a moving ball	To learn to kick a ball to pass to a target position when the ball is moving in a random direction	Task 9

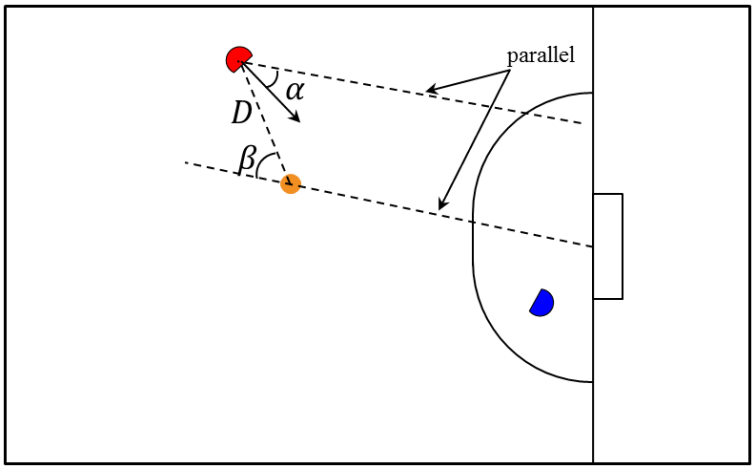


Figure 5.1: State variables. The symbol D denotes the distance between the ball and the robot, α represents the angle between robot orientation and the line connecting the ball and the target position, and β denotes the angle between the line connecting the robot and the ball, and the line connecting the ball and the target position. It is assumed that the target is the centre of the opponents' goal in this figure.

the experiments. Given the nine tasks, nine experiments were performed in this research. The shooting skills were developed by the seven RL experiments (Tasks 1 to 7). The passing skills (Tasks 8 and 9), however, were not developed by conducting RL experiments but by exploiting the results of the first two experiments (Tasks 1 and 2). This concept will be dealt with in detail in Section 6.8. It is assumed that in all tasks no opponent player exists except for the goalkeeper.

5.2 State space

The state space used for the experiments dealing with a stationary ball consists of three variables: 1) the distance D between the ball and the robot, 2) the angle α between robot orientation and the line connecting the ball and the target position, and 3) the angle β between the line connecting the robot and the ball, and the line connecting the ball and the target position. These variables are from Duan *et al.* (2007). The geometric relations of the objects in the field and the state variables are illustrated in Figure 5.1. It is assumed that the target is the centre of the opponents' goal in this figure.

Other experiments needed more state variables as the learning agent has to deal

with a moving ball. Four additional variables were used for this experiment: v_{bx} and v_{by}, the velocity of the ball in x- and y-direction respectively, and v_x and v_y, the velocity of the learning agent in x- and y-direction respectively.

Table 5.2 shows the state variables used for each experiment. Note that these state variables are continuous, therefore the problems are defined as Markov Decision Processes (MDPs) with infinite state spaces. Note also that the target position (the centre of the opponents' goal in this case) is not used directly as one of the state variables, but it is used indirectly in the algorithm in the calculation of two state variables (α and β). This plays an important role in the implementation of the passing skills without performing an experiment but using the results of the other experiments.

Table 5.2: State variables used for each experiment. Refer to Figure 5.1 for the definitions of D, α, and β. The velocity of the ball in x- and y-direction are denoted by v_{bx} and v_{by} respectively, and v_x and v_y stand for the velocity of the learning agent in x- and y-direction respectively.

Experiments	**State variables**
1, 3, 4, 5 and 8	D, α, β
2, 6, 7 and 9	D, α, β, v_{bx}, v_{by}, v_x, v_y

5.3 Action space

The action space consists of three continuous variables: moving direction θ (in rad), moving speed v (in m/s) and angular velocity ω (in rad/s). As RL algorithms require a finite number of actions, each action variable has a specific number of actions by making the continuous action space discrete, or by assigning appropriate values. This number of actions, as well as the values used for the action variables, is not necessarily the same in all experiments conducted in this research. Actually, there are considerable differences between the first two RL experiments and the following five RL experiments. All changes in the number of actions or in the values of actions are caused by the fact that in Experiments 3 to 7 the learning agent needs to aim at the goal more accurately with the goalkeeper in front of the goal, therefore more refined control of actions is required.

The action spaces used for Experiments 8 and 9 are the same as those for Experiments 1 and 2, respectively, as the later experiments use the results of the first two experiments. Therefore, in the following discussions in this section, only Experiments 1 to 7 are considered.

5.3.1 Action variables

Moving direction

The moving direction θ is defined as in a normal polar coordinate system, i.e., it is measured anticlockwise from the direction of a positive x-axis. Thirty-six different moving directions were used for Experiments 1 and 2, and the number of moving directions was extended to 72 in the remaining experiments. The moving direction θ is discretised such that the total number of values (36 or 72) are equally distributed between the upper and lower limits, which is given as

$$\theta \in [0, 2\pi).$$

Moving speed

The moving speed v is a scalar value at which the learning agent moves along the moving direction θ. The values of moving speed used in each experiment are given as follows: $v = 0.5$ or $v = 1$ for Experiment 1, and $v = 0.5$, $v = 1$ or $v = 2$ for the remaining experiments. In these experiments an additional value $v = 2$ was needed for the learning agent to deal with the moving ball or to work against a goalkeeper.

Angular velocity

The angular velocity (ω in rad/s) is the rotational speed of the learning agent with which it turns around to face a different direction. Positive values of the angular velocity indicate that it turns anticlockwise and negative values mean the opposite. Nine different angular velocities, including positive, zero and negative values, were used in all the experiments. The actual values used for the angular velocity ω is given as

$$\left\{-2\pi, -\pi, -\tfrac{1}{2}\pi, -\tfrac{1}{4}\pi, 0, \tfrac{1}{4}\pi, \tfrac{1}{2}\pi, \pi, 2\pi\right\}$$

for the first two experiments, and for the remaining experiments they are

$$\left\{-\pi, -\tfrac{1}{2}\pi, -\tfrac{1}{4}\pi, -\tfrac{1}{8}\pi, 0, \tfrac{1}{8}\pi, \tfrac{1}{4}\pi, \tfrac{1}{2}\pi, \pi\right\}.$$

Table 5.3: Number of values in action sets and the total number of possible actions for each experiment. The action sets Θ, $\mathbb{V}$ and Ω contain possible values of moving direction θ, moving speed v and angular velocity ω, respectively.

Experiments	Number of values in Θ	Number of values in $\mathbb{V}$	Number of values in Ω	Total number of actions
1 and 8	36	2	9	$657 = 36 \times 2 \times 9 + 9$
2 and 9		3		$981 = 36 \times 3 \times 9 + 9$
3 to 7	72			$1953 = 72 \times 3 \times 9 + 9$

5.3.2 The total number of actions

By virtue of its omnidirectional wheels, the learning agent is able to move from one position to another without having to turn to face the target position first. In addition, it is possible to rotate with a certain angular velocity while it is moving linearly. Therefore, an action is defined by the combination of the three action variables mentioned in the previous section.

Besides the combination of these three variables, there are additional cases that should be considered in particular. They are when the learning agent does not move but only turns at a certain angular velocity. This adds nine (the number of angular velocities used in the experiment) more cases to the action space, including when $\omega = 0$, in which case the learning agent neither moves nor rotates. This special case is considered as the *kick* action, i.e., the robot operates its kicker to kick the ball. Therefore, the action space is described by

$$A \;=\; \{\,(\theta, v, \omega) \mid \theta \in \Theta,\, v \in \mathbb{V},\, \omega \in \Omega\,\} \cup \{\,(v, \omega) \mid v = 0,\, \omega \in \Omega\,\},$$

where Θ, $\mathbb{V}$, and Ω are action sets containing possible values of θ (moving direction), v (moving speed) and ω (angular velocity), respectively. The number of possible values in each action set and the total number of possible actions for each experiment are summarised in Table 5.3.

5.4 Terminal states

In the nine experiments, the goal of the learning agent is either to score a goal, or to pass the ball to a certain target position. Therefore, the experiments are designed as

episodic problems where the interactions between the agent and the environment come to a natural end, be it success or failure. In the seven RL experiments for shooting skills, if the learning agent scores a goal as a result of an action given a specific state, the next state is a success terminal state. On the other hand, if the agent takes a shot and it misses the goal or is blocked by the goalkeeper, it is considered as a failure terminal state. In Experiments 8 and 9, it is considered as a success if the learning agent kicks the ball and the ball moves in the target direction. (A detailed success criterion will be presented in Section 6.8.) Otherwise it is regarded as a failure.

In addition, there are several more cases in which the episode ends with a failure terminal state. If the learning agent reaches a state that satisfies one of the following conditions, it is considered to be a failure. Consequently the episode is terminated immediately.

- The ball is out of field.
- The learning agent is out of field.
- The distance between the ball and the learning agent is longer than the maximum limit D_{Max}.
- The number of steps in the episode is more than the maximum number of steps n_{Max}.

The last two conditions are to accelerate the learning process by terminating the episode when the learning agent explores wrong states for too long. The values $D_{\text{Max}} = 3\,000$ mm and $n_{\text{Max}} = 100$ were used throughout the experiments.

5.5 Reward system

A cost-based method was used for the reward system, as was done in the work of Riedmiller *et al.* (2009). In a cost-based method, the learning agent is given a cost c_{t+1}, instead of a reward r_{t+1}, as a result of an action a_t at a state s_t, and the task is to minimise the overall cost in the long run. The cost function $c(s, a, s')$, the cost given to the agent when it chooses an action a at a state s and the following state is s', is

defined as

$$c(s,a,s') = \begin{cases} 0.00 & \text{if } s' \in S^+, \\ 1.00 & \text{if } s' \in S^-, \\ 0.01 & \text{else,} \end{cases} \tag{5.1}$$

where S^+ and S^- denote the set of terminal states with success and failure, respectively. Punishing the agent with a small constant cost 0.01 in all non-terminal states encourages the agent to achieve the goal as soon as possible. The costs for terminal states were determined as such because the output of the MLP is in the range of $[0, 1]$ due to the sigmoid function used in the MLP (refer to 2.3(b)).

5.6 The RL algorithm used: temporal-difference value iteration

As mentioned previously, the ML technique used for these experiments is RL combined with an MLP as a function approximator. Among various algorithms introduced in Section 2.2.2 to solve RL problems, the Temporal-Difference (TD) value iteration algorithm with state-value functions (Algorithm 5) was chosen for the experiments (refer to Section 2.2.2.5).

Model-based algorithms (such as policy iteration or value iteration) assume finite Markov Decision Processes (MDPs), therefore they are not appropriate for the problems in this research, which are defined as infinite MDPs. It is possible, though, to apply model-based algorithms to problems with infinite MDPs by discretising the continuous state space. In fact, this is a crude form of function approximation. However, for problems with infinite MDPs, which often have stochastic features in the dynamics of the environment, model-free algorithms are typically used. (Of course, a parametrised function approximator is needed in these cases, as discussed in Section 5.7.)

Q-learning was introduced in Section 2.2.2.4 as a typical model-free algorithm to search for the optimal action-value function Q^*. When the problem has stochastic features and thus the dynamics of the environment are unknown, which is often true for problems with infinite MDPs, Q-learning is a good option. However, in the problems being handled in this research, the dynamics of the environment in a given state are known. That is, given a state s and an action a, the environment transitions to a new state s' deterministically. This is obvious for Tasks 1 to 3, 8 and 9 where there is no

goalkeeper. It can also be applied for the remaining tasks, assuming the goalkeeper's behaviours are known and fixed. All events are deterministic and there are no stochastic factors in the environment.

This knowledge of the dynamics of the environment is exploited to the best advantage in this research by using a TD value iteration algorithm with state-value functions (Algorithm 5) instead of a Q-learning algorithm (Algorithm 4). With Algorithm 5, the learning task is simplified because it has the value function for states only, instead of the value function for state-action pairs.

5.7 The multi-layer perceptron model

The MLP structure

Because the state variables are continuous, the value function $V(s)$ in Algorithm 5 should be represented as a parametrised function $[F(\vec{z})](s)$. With an MLP as the function approximator, the parameter vector $\vec{z}$ would be composed of the weights of the MLP. The inputs of the MLP are the state variables discussed in Section 5.2, and the output of the MLP is the approximated value of the value function $V(s)$.

Therefore, the number of input neurons of the MLP, m, is equal to the number of state variables given in Table 5.2, and the number of output neurons of the MLP, n, is fixed to one. To simplify the problem, all MLP models used in the experiments have one hidden layer, which has proved to be enough for the purpose of approximating $V(s)$ accurately. The number of neurons in the hidden layer, denoted by l, was determined empirically. Table 5.4 shows the structure of the MLP used in each experiment in the form of m–l–n.

The input values to the MLP are normalised such that the values are in the range of $[0, 1]$. All neurons in the hidden layer and the output layer have the logistic function (2.40) with $c = 1$ for their activation function.

The training data set

In normal MLP learning, a set of training data is provided. In RL with MLP, the learning agent collects the training data by interacting with the environment and observing the reward and the next state. For the problem of evaluating value function $V(s)$, the

Table 5.4: The structure of the MLPs used. The structure of the network is presented in the form of m-l-n, where m denotes the number of input nodes, l is the number of neurons in the hidden layer, and n stands for the number of output neurons.

Experiments	MLP structure
$1, 3, 4, 5$ and 8	3–7–1
$2, 6, 7$ and 9	7–21–1

training data set $\mathbb{T}$ is given by

$$\mathbb{T} = \left\{(s, t(s)) \,|\, s \in S'\right\},$$

where $S' \subset S$ is a set of states visited by the learning agent and the target $t(s)$ is provided by means of a TD target (refer to Section 2.2.2.3). In the TD value iteration algorithm with state-value functions (Algorithm 5), the TD target $\mathfrak{v}$ is defined as

$$\mathfrak{v} = \max_a [r_a + \gamma\, V(s'_a)], \tag{5.2}$$

which was used for the target $t(s)$ in the experiments. Typically the discount rate $\gamma = 1$ was used, that is, no discount was considered for future rewards since the experiments are designed as episodic cases.

The batch mode algorithm

When an MLP is used as a function approximator for an RL problem, the batch mode Back-Propagation (BP) algorithm (Algorithm 7) is more efficient than the sequential mode BP (Algorithm 6). Because the MLP approximates the function in a global way, updating the parameters each time a training datum is collected, as in the case of the sequential mode BP, has an impact on the outcome of arbitrary other states. The batch mode BP algorithm turns out to be stabler in the face of these unintended changes (Riedmiller *et al.*, 2009).

The TD value iteration algorithm with an MLP

Since the batch mode BP algorithm is used in the experiments, the training data $(s, t(s))$ is accumulated for each state the learning agent visits until the episode ends. Then the parameters of the approximation function, i.e., the weights of the MLP, are

Algorithm 8 The TD value iteration with MLP (batch mode).

Initialise $\vec{z}$ arbitrarily
for all episodes **do**
    Initialise the training data set $\mathbb{T} = \varnothing$
    Initialise s
    repeat(for each step of episode)
        for all possible action a **do**
            Take action a, observe reward r_a, and the next state s'_a
        end for
        $t(s) = \max_a [r_a + \gamma V(s'_a)]$
        $\mathbb{T} \leftarrow \mathbb{T} \cup \{(s, t(s))\}$
        $a \leftarrow \arg\max_a [r_a + \gamma V(s'_a)]$ with ϵ-greedy
        Take action a and observe the next state s'
        $s \leftarrow s'$
    until s is terminal
    Update $\vec{z}$ using the batch mode BP algorithm (Algorithm 7) with training data set $\mathbb{T}$
end for

updated once for all data in the training data set $\mathbb{T}$ at the end of the episode. The algorithm for the TD value iteration with MLP as a function approximator is shown in Algorithm 8, which was used in this research to implement the experiments. The learning takes place at the end of each episode and continues until the predetermined maximum number of episodes (N_{Max}) is reached. The symbol N_{Max} is discussed later in Section 5.9.

5.8 Test episodes

As mentioned in the previous section, the learning takes place episode by episode in the experiments. During each episode the learning agent collects training examples, which is used in the learning process at the end of the episode. In this *learning episode*, the learning agent chooses its next action with ϵ-greedy policy to give the algorithm some degree of exploration.

Besides learning episodes, other types of episodes are needed to evaluate the performance of the MLP. These are called *test episodes*. In the test episodes, the learning agent chooses its next action with 100% greedy policy, i.e., it always selects its next action greedily based on the current estimation of the value function $V(s)$.

In this research it was observed that the MLP learns quite quickly in terms of the number of learning episodes, thus it was decided to run five test episodes after each learning to check the performance of the current network. Therefore, all experiments performed in this research follow the process illustrated in Figure 5.2.

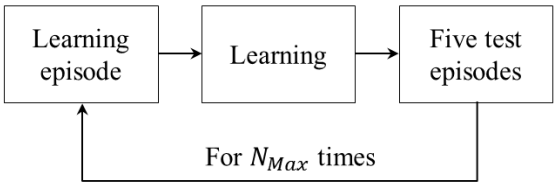


Figure 5.2: The learning process followed in the experiments.

5.9 Convergence

As mentioned in Section 2.3.2, the learning rate η, which is used in the back-propagation algorithms (Algorithm 6 and Algorithm 7), is gradually decreased in time for convergence. In all ML experiments conducted in this research, the learning rate starts with $\eta = 0.2$ and is decreased after each learning episode at a constant rate of ρ, i.e., η is updated as $\eta \leftarrow \rho \times \eta$. Therefore, η_n, the learning rate after the n^{th} learning episode, is calculated as

$$\eta_n = 0.2 \times \rho^{n-1}. \tag{5.3}$$

The learning rate η_n converges to zero as $n \to \infty$. A simple program, which is presented in Appendix A, indicates the number of learning episodes that makes the value of η_n effectively zero for a given value of ρ. After that the weights of the MLP are not changed. Therefore, N_{Max}, the maximum number of learning episodes in an experiment, is determined by the code presented in Appendix A. As the decreasing factor ρ is not necessarily the same for each experiment, N_{Max} is also different for each experiment. The values of N_{Max} used in each experiment, as well as other parameters, are discussed in the next section.

Table 5.5: Parameters used in the experiments.

Symbols	Definitions
m	the number of neurons in the input layer
l	the number of neurons in the hidden layer
n	the number of neurons in the output layer
c	shape parameter for the logistic function
η	learning rate in the batch mode BP algorithm
ρ	decreasing factor for η
γ	discount rate for future rewards
ϵ	a small positive value to define the exploration rate for ϵ-greedy search
D_{Max}	the maximum distance allowed between the ball and the learning agent
n_{Max}	the maximum number of steps in an episode
N_{Max}	the maximum number of episodes in an experiment

5.10 Parameter settings

Parameters used in the experiments are shown in Table 5.5. All these parameters were explained in previous discussions. Parameter settings have a strong influence on the performance of the ML experiments. The effect of changes to these parameters, however, is not discussed in this research. All parameters were kept constant for all the experiments, except for the learning rate η, which is decreased over time, as discussed earlier. All parameter settings are hand-tuned and therefore based on empirical studies. They are optimised after a number of experiments with different settings. The values used in this research are listed in Table 5.6.

5.11 Conclusion: Experimental design

In this chapter the experimental design of the RL experiments performed in this research was discussed. The RL experiments were conducted to develop the individual soccer skills mentioned in Section 1.4.3 to form the basic building blocks of the DMM. Nine experiments were defined in Table 5.1 based on various factors such as whether the skill is shooting or passing, whether the ball is stationary or moving and whether the goalkeeper is present or not.

Table 5.6: Parameter settings used in the experiments. Note that Experiments 8 and 9 are not RL experiments, thus learning-related parameters were indicated as Not Available (NA).

Experiments	**MLP**	c	η	ρ	γ	ϵ	D_{Max}	n_{Max}	N_{Max}
1	3– 7–1			0.95					699
2	7–21–1			0.75					125
3	3– 7–1			0.95					699
4	3– 7–1		0.2	0.97	1	0.2			1177
5	3– 7–1	1		0.97			3 000 mm	100	1177
6	7–21–1			0.75					125
7	7–21–1			0.90					340
8	3– 7–1		NA	NA	NA	NA			NA
9	7–21–1		NA	NA	NA	NA			NA

The common features of all nine experiments were discussed, with specific reference to state space, action space, terminal states and reward system. Also, the algorithm used in the experiments, the TD value iteration algorithm with an MLP as a function approximator, was presented in Algorithm 8. The chapter also explained test episodes to check the performance of the MLPs, stopping criteria and parameter settings.

In the next chapter, the results of individual experiments will be presented and analysed.

Chapter 6

Experimental results

The previous chapter provided the experimental design and the general information of the experiments. In this chapter the experiments will be explained in detail. Each section focuses on an individual experiment, presenting the experimental set-up and the results with discussions. Sections 6.1 to 6.7 deal with the Reinforcement Learning (RL) experiments performed to develop shooting skills and Section 6.8 focuses on the passing skills. As mentioned in Section 5.1, the passing skills were not developed using RL experiments but were implemented by exploiting the results of Experiments 1 and 2. The weights of the learned Multi-Layer Perceptrons (MLPs) in the experiments are presented in Appendix B.

6.1 Experiment 1: shooting a stationary ball

6.1.1 Experimental set-up

Experiment 1 is to learn how to shoot a stationary ball. No goalkeeper is assumed, therefore the learning agent and the ball are the only moving objects in the environment. At the beginning of each episode, the ball is located in a random position in the opponents' half of the field. Figure 6.1 shows the possible area in which the ball can be located initially. (In all experiments, it is assumed that the learning agent attacks the goal on the right side of the field.) The learning agent is randomly positioned at any place in the field, such that the distance to the ball is no more than 2 800 mm. This is to avoid an unnecessarily early termination of the episode by entering the terminal state $D > D_{\text{Max}} = 3\,000$ mm. The episode ends if the ball is kicked by the learning

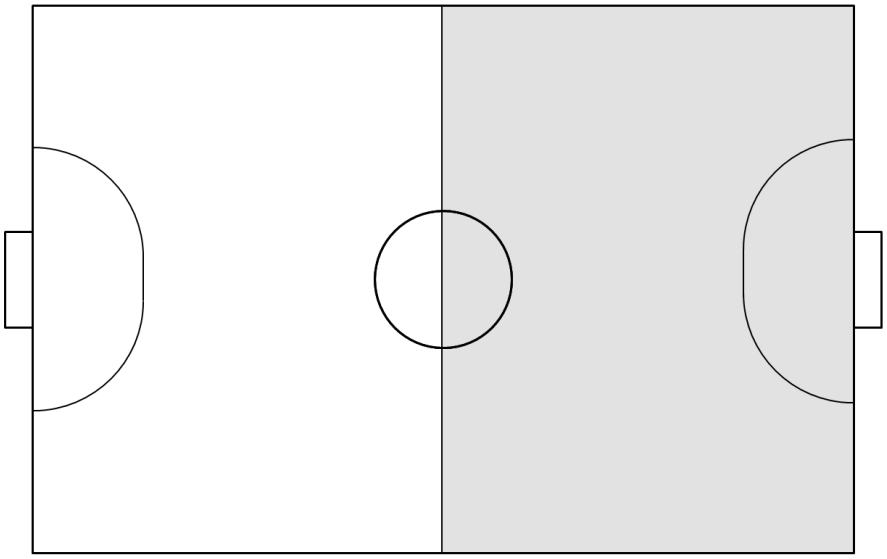


Figure 6.1: The possible initial position of the ball (shaded area).

agent (whether the result is a success or not), or if the state meets one of the terminal conditions presented in Section 5.4.

6.1.2 Results and discussions

Figure 6.2 shows the learning curve of Experiment 1. The graph represents the performance of the learned MLP at the end of each learning episode. The performance was measured from the results of the test episodes, but the graph was smoothed by using a moving average over the last 100 test episodes, which corresponds to the average performance of the last 20 learned MLPs. If the number of learning episodes was less than 20, the results from all test episodes performed until then were used to calculate the moving average. The same rule was applied to all the experimental results in this chapter.

As can be seen in Figure 6.2, the learning was very effective. After about 35 episodes, the learning agent consistently succeeded in scoring a goal with a success rate of approximately 99%. To investigate why the learning agent failed in 1% of the cases, another experiment was performed where 500 test episodes were run using the learned MLP. It showed a 99.4% success rate, with three failures. Figure 6.3 shows the initial position of the ball in the 500 test episodes, indicating the failure cases in red. In all the failure

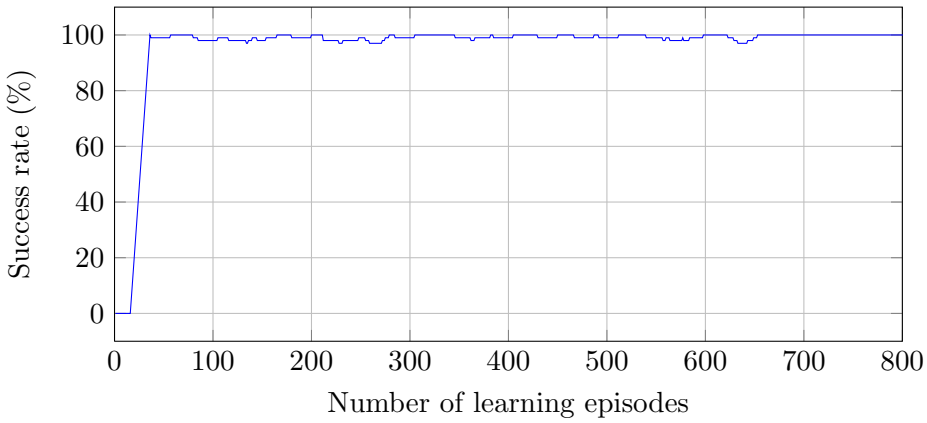


Figure 6.2: Result: shooting a stationary ball.

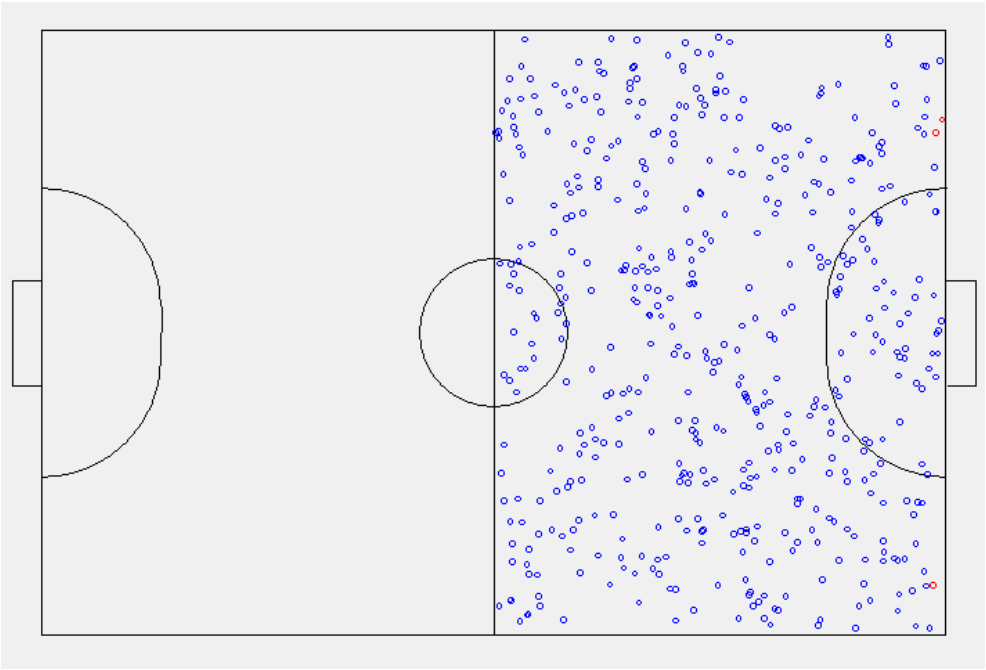


Figure 6.3: The initial position of the ball in 500 test episodes for Experiment 1. Failure cases are marked in red.

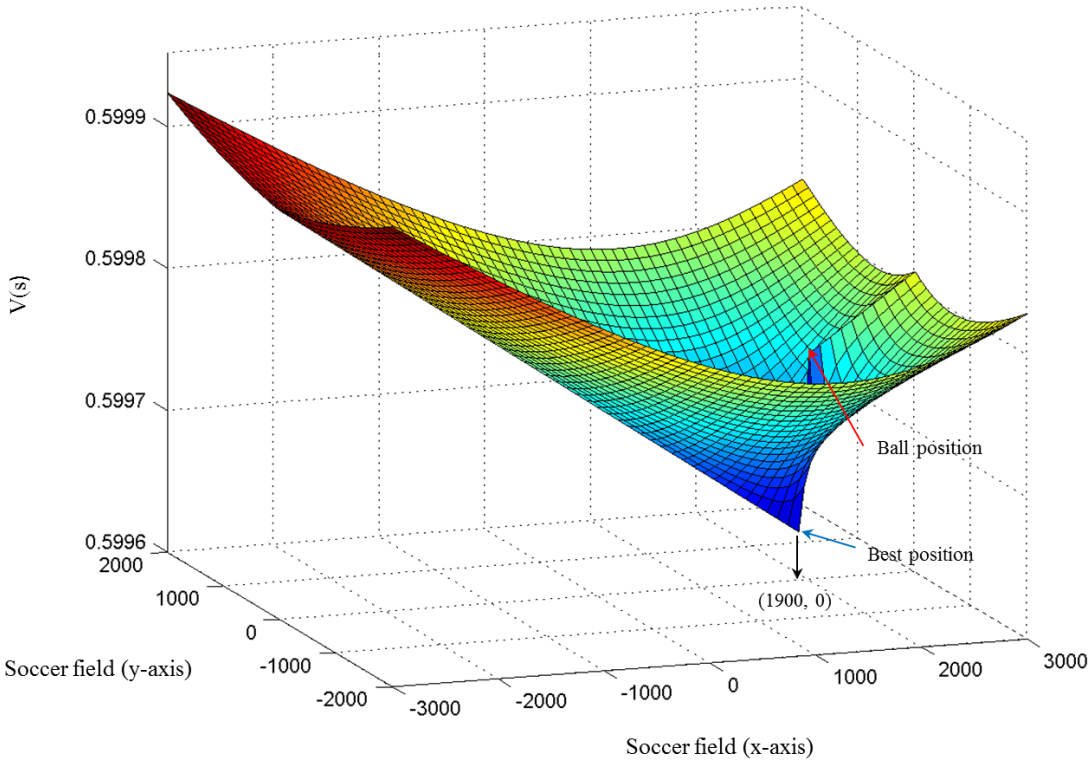


Figure 6.4: Learned value function $V(s)$ mapped on the field. It is assumed that the ball is initially positioned at $(2\,000, 0)$.

cases, the ball was placed too close to the goal line and the shooting angle from the position was too small.

Figure 6.4 depicts the learned value function $V(s)$ for a specific test scenario, i.e., when the initial position of the ball was set to $(2\,000, 0)$. The figure shows the value function mapped on the field. With a given ball position, the two state variables (D and β) depend on the position of the learning agent on the field, thus these two state variables can be mapped on the field. The field was divided into 60×40 grid blocks, each 100 mm $\times$ 100 mm, and the value of the value function $V(s)$ was calculated at each node of the grid block by feeding forward to the learned MLP. Since the other state variable α is irrelevant to the position on the field, the value function for a given position was repetitively computed with 360 different values of α, the minimum of which was displayed in Figure 6.4. (Since the cost-based method was used for the reward system (Section 5.5), the minimum means the best in the experiments.)

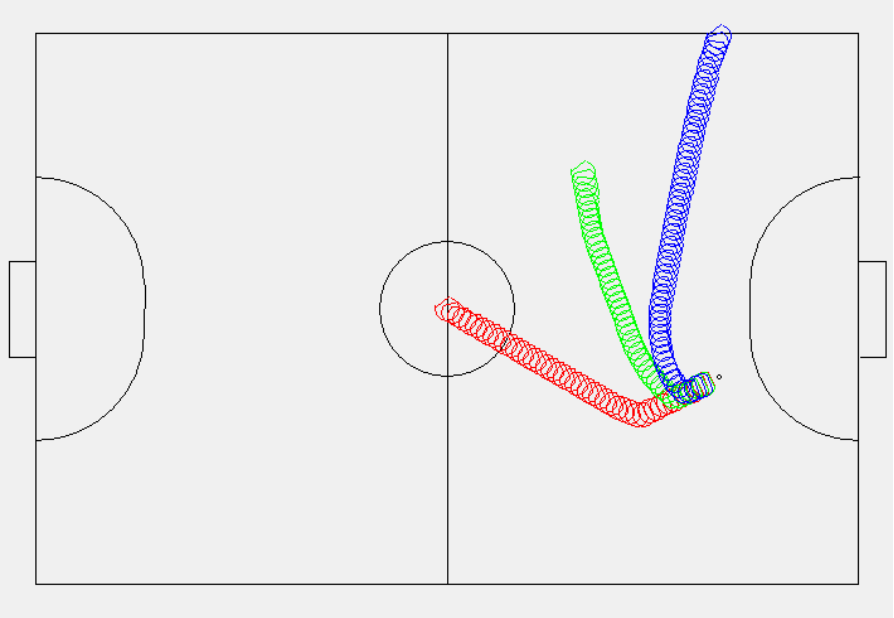


Figure 6.5: The trajectory of the learning agent with different initial positions. The ball is positioned at $(2\,000, -500)$, and the learning agent is initially located at $(0, 0)$ (marked in red), $(1\,000, 1\,000)$ (marked in green), and $(2\,000, 2\,000)$ (marked in blue).

Unsurprisingly, the position $(1\,900, 0)$, right in front of the ball in the direction of the centre of the goal $(3\,025, 0)$, has the minimum value of the value function. The quality of the position gets worse the further it is from the best position. Also, the figure shows that those positions on the line segment connecting the ball and the centre of the goal have particularly high values because, in these positions, the learning agent is between the ball and the goal, thus it is more difficult to get to the best position from these positions.

Figure 6.5 shows another example of the learning results. It compares the trajectories of the learning agent with three different starting positions. The ball position was set to $(2\,000, -500)$, and the learning agent was initially located at $(0, 0)$, $(1\,000, 1\,000)$, and $(2\,000, 2\,000)$ in each case. In all three cases, it was observed that the learning agent approaches the ball along a reasonably short path, but not along a straight line. The figure shows smooth curves in the trajectories of the learning agent, which are much more human-like than straight lines. The curved surface in Figure 6.4 explains the curved trajectories in Figure 6.5. This is a feature of the learned MLP, which is non-trivial to achieve with hand-coded approaches.

It was also observed that the learning agent always adjusts its orientation such that $\alpha = 0$ within the first several steps of the test episode. Initially the orientation of the learning agent was set to face the direction 120 degrees away anticlockwise from the positive x-axis. As the learning agent rotates clockwise, this orientation was changed to about 25 degrees from the positive x-axis, which would make the agent face the goal in the final position. The orientation was fixed after that.

6.1.3 Conclusion: Experiment 1

In this section an RL experiment to develop a shooting skill for a stationary ball was discussed. The result showed a success rate of 99%, which means the learning agent was able to score a goal in all cases, except when the ball was placed very close to the goal line, thus making the shooting angle from the position very small.

6.2 Experiment 2: shooting a moving ball

6.2.1 Experimental set-up

The task of Experiment 2 is to score a goal by shooting a moving ball. The goalkeeper is not considered. The ball is initially placed in a quarter of the field, i.e., in the lower half of the opponents' field (the grey area in Figure 6.6), while the learning agent is located in the other half of the opponents' field (the lined area in Figure 6.6). At the beginning of each episode, the ball moves at a constant speed of 1 m/s towards a random point between the learning agent and the goal from the position of the ball, i.e., the ball can move in any direction between the two dotted lines in Figure 6.6. As mentioned in Section 5.3, the learning agent can move in 36 different directions, with three different speeds, also with nine different angular velocities. The value $\rho = 0.75$ is used for the decreasing factor of the learning rate η.

6.2.2 Results and discussions

Figure 6.7 shows the result of Experiment 2. The graph shows that the learning agent achieves approximately 96% success rate after about 30 learning episodes. To examine the failure cases, a separate experiment was conducted in which 100 test episodes were run using the learned MLP. Two cases turned out to be failures. In both cases, the ball

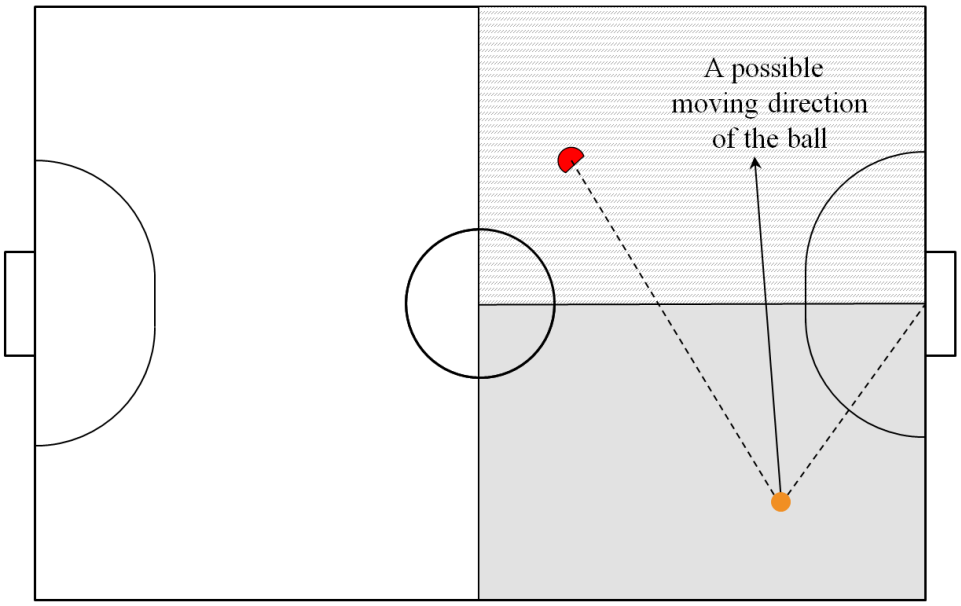


Figure 6.6: The initial state set-up for Experiment 2. In the beginning of each episode, the learning agent is positioned randomly in the upper half of the opponent's field (lined area) and the ball is located in the other half of the opponent's field (grey area). The ball moves with a constant speed of 1 m/s towards a random direction between the two dotted lines.

was placed relatively close to the goal while the initial position of the learning agent was far from the ball. This gave insufficient time for the learning agent to approach the ball. Figure 6.8 shows an example of the failure cases. The learning agent started from a point $(235, 818)$ and the initial position of the ball was $(2266, -844)$. The red arrow and blue arrow indicate the moving direction of the learning agent and the ball, respectively. The black arrow shows the corresponding positions of the learning agent and the ball at the same time-step. Figure 6.8 clearly shows that the possibility of scoring in this case is very low with the initial set-up of the episode.

The most interesting aspect of this experiment is the fast learning. When $\rho = 0.95$ was used for the decreasing factor of the learning rate η (as in Experiment 1), succeeding cases started to be seen as early as after the sixth learning episode (while they were found only after the 16^{th} learning episode in Experiment 1). This exceptionally fast learning causes the problem of over-updating, which is discussed in the following

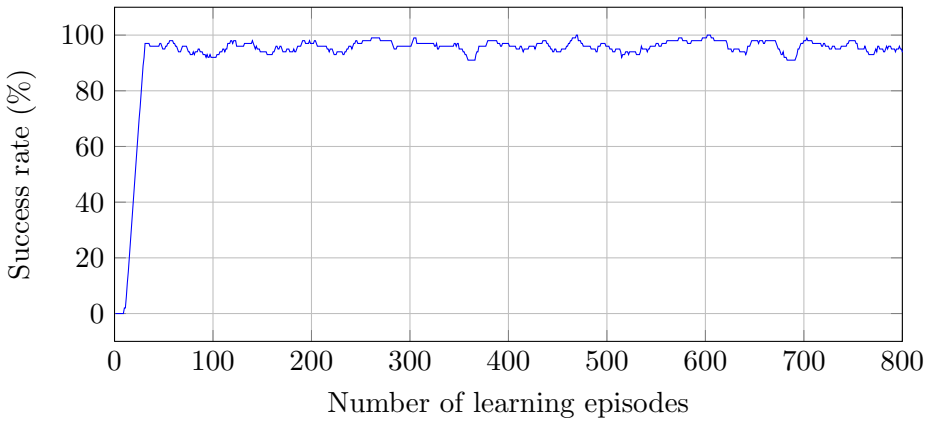


Figure 6.7: Result: shooting a moving ball with $\rho = 0.75$.

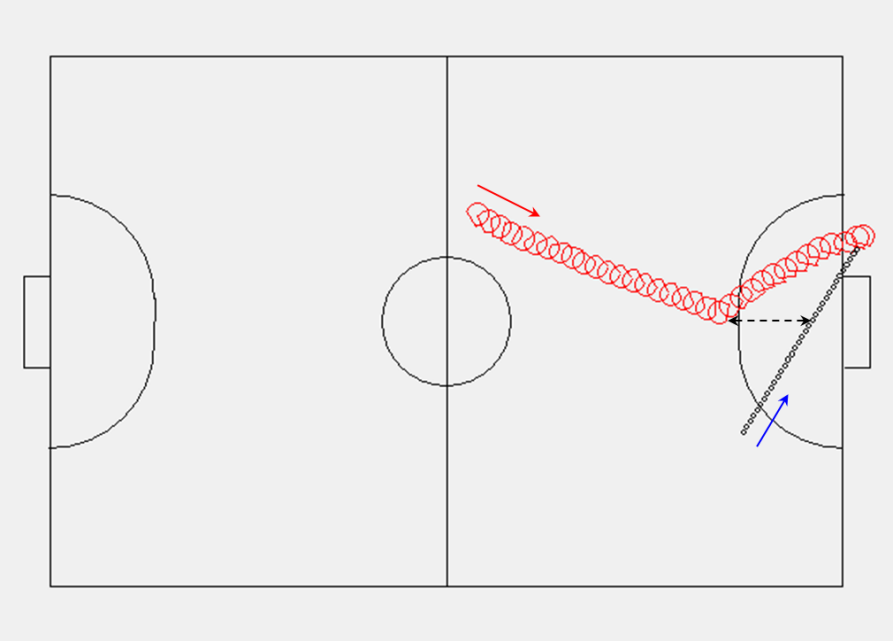


Figure 6.8: The trajectory of the learning agent and the ball in a failure case. The red arrow and blue arrow indicate the moving direction of the learning agent and the ball, respectively. The black arrow shows the corresponding positions of the learning agent and the ball at the same time-step.

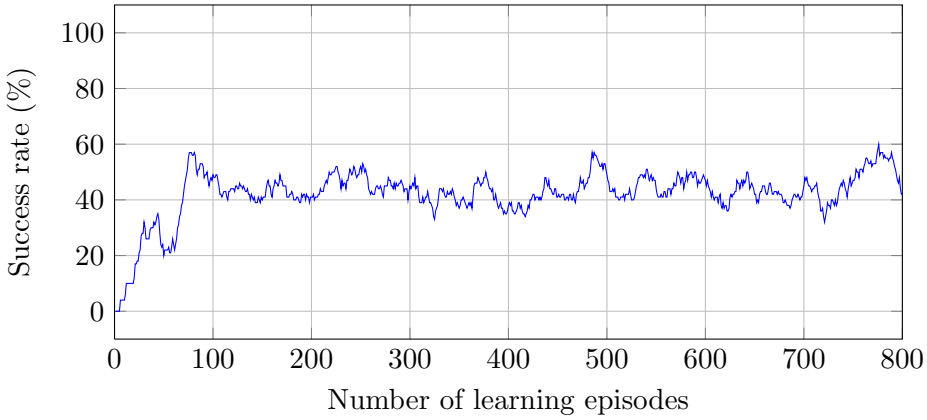


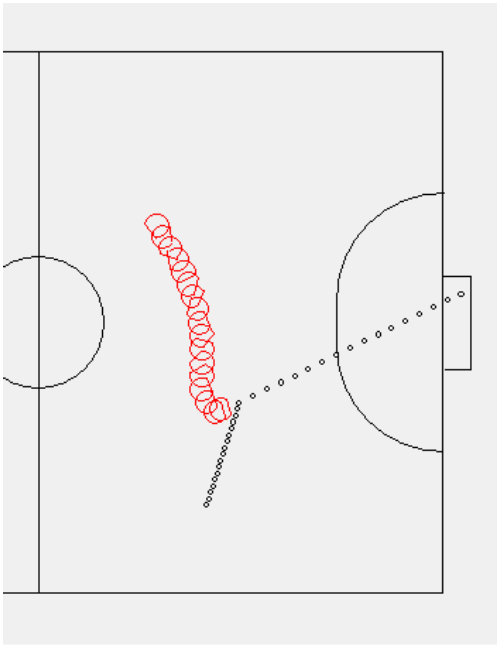
Figure 6.9: Result: shooting a moving ball with $\rho = 0.95$.

paragraphs.

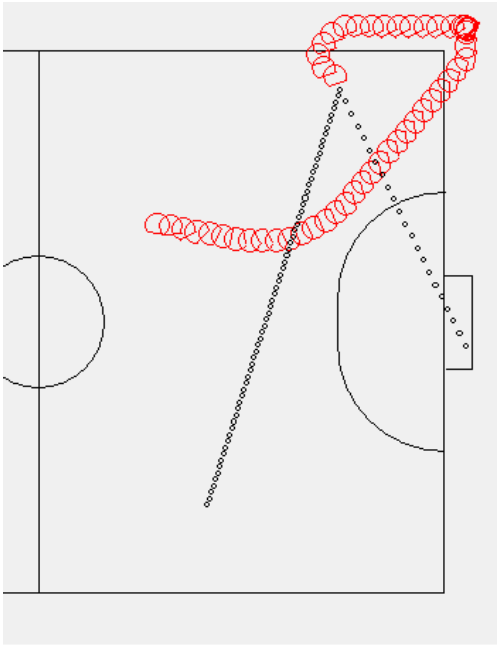
As discussed in Section 2.4, there are two estimations in reinforcement learning with function approximation: the estimation of the optimal value functions V^* or Q^*, and the estimation of the parameters $\vec{z}$. In this research, the former is estimated by the Temporal Difference (TD) target $\mathfrak{v} = \max_a [r_a + \gamma V(s'_a)]$ (refer to (5.2) in Section 5.7), while the latter is estimated by the MLP.

In this experiment, after a certain number of learnings have been applied, the learning agent was able to score a goal in some test episodes. This means both types of estimations have matured to some degree, if not to the point of perfection. The learning rate η, however, has not been decreased enough at this stage when $\rho = 0.95$ was used for the decreasing factor, which forced the already working parameters ($\vec{z}$), the weights of the MLP in this case, to be updated by too large an increment. These over-updated weights in return caused the unwanted effects on the estimation of the value functions V^* or Q^* eventually, as the value of the next state $V(s'_a)$ in the TD target of the value functions is estimated by means of the MLP.

With the learned MLP and $\rho = 0.95$, the learning agent was able to score a goal only half of the time (see Figure 6.9). More importantly, the learning agent showed a pattern of moving toward the upper right corner of the field before it approached the ball. This seems to be the effect of the over-updating because, from the experimental set-up described in Section 6.2.1, the ball is moving towards the upper right corner most of the time. Figure 6.10 compares the trajectories of the learning agent and the ball from the



(a) The trajectory of the learning agent and the ball from the well-trained MLP ($\rho = 0.75$).



(b) The trajectory of the learning agent and the ball from the over-updated MLP ($\rho = 0.95$).

Figure 6.10: Trajectory comparison between the well-trained MLP and the over-updated MLP.

well-trained MLP (i.e., when $\rho = 0.75$ was used) and the over-updated MLP ($\rho = 0.95$). In both cases, the ball and the learning agent were initially positioned at $(1\,258, -1\,363)$ and at $(885, 726)$, respectively, and the ball starts to move in the direction of 1.26225 rad anticlockwise from the direction of the positive x-axis. Figure 6.10(a) shows the results from the well-trained MLP, while Figure 6.10(b) illustrates the behaviour of the learning agent when moving towards the upper right corner of the field before approaching the ball and shooting.

Thus far, it was discussed that, in RL with an MLP as a function approximator, when the learning takes place quite fast, the learning rate of the MLP should accordingly be decreased quickly to avoid over-updating. However, the reason why the learning occurred fast in this experiment was not investigated in this research. This forms a good topic for future studies, along with the subject of how to match the learning speed of RL and the decreasing ratio of the learning rate in an MLP. In this research, the appropriate value of ρ in each experiment was determined empirically.

6.2.3 Conclusion: Experiment 2

An RL experiment was discussed in this section. The purpose of the experiment was to train a learning agent to shoot a moving ball. The result showed a 96% success rate. The failure cases occurred when the learning agent was positioned too far from the ball, thus it had insufficient time to approach the ball. It was observed that the learning proceeded fast in this experiment when the usual $\rho = 0.95$ was used. The accompanying over-updating issue was discussed. The decreasing factor of the learning rate ρ was eventually adjusted to the value of 0.75 to avoid over-updating.

6.3 Experiment 3: shooting against a static goalkeeper

In the previous two experiments the goalkeeper was absent. Tasks being handled in this and the following two sections (Sections 6.3, 6.4 and 6.5) involve the goalkeeper in the experiments. In all experiments, the learning agent deals with a stationary ball, i.e., the task is to learn to shoot a stationary ball against a goalkeeper, whose behaviour becomes more competitive in each experiment. Therefore, the task in each experiment becomes incrementally harder to achieve than the previous one.

6.3.1 Experimental set-up

In Experiment 3, the learning agent tries to learn to shoot a stationary ball while the goalkeeper stands still in the centre of the goal. The goalkeeper is positioned at $(3\,025, 0)$. Since the width of the goal is 700 mm and the diameter of the goalkeeper is 180 mm, the goal is divided into two narrow segments, each 260 mm wide. The learning agent needs to aim at one of the two segments to perform the task properly. Therefore, instead of using the centre of the goal $(3\,025, 0)$ as the target position (which is used in the calculation of the two state variables α and β), $(3\,025, 300)$ or $(3\,025, -300)$ is used, depending the initial position of the ball. The one which is closer to the initial position of the ball is adopted. Figure 6.11 illustrates this concept.

In addition, to deal with the difficulty caused by the narrowed target, the action space is extended to have a larger number of actions, i.e., the action set Θ for moving direction is discretised into 72 different moving directions. Therefore the learning agent can change its direction by five degrees at a time in this experiment. Also, the values of the nine angular velocities are reduced to halves, which means the learning agent turns

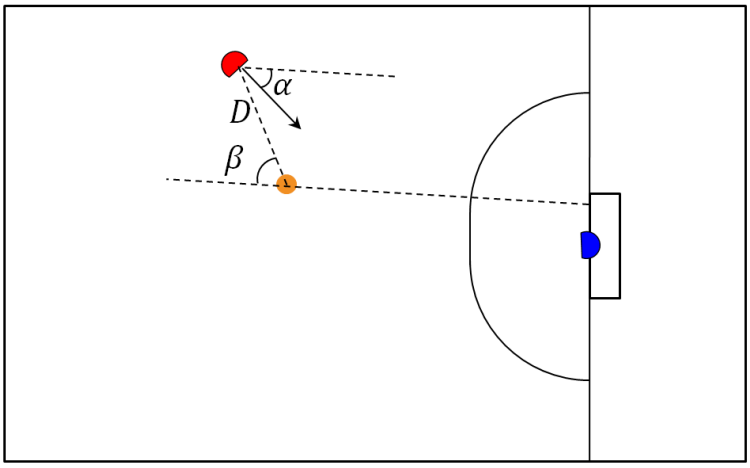


Figure 6.11: Aiming at one segment of the goal depending on the initial position of the ball.

slowly but it can adjust its orientation more precisely. This setting of action variables applies to all the following experiments.

The experimental set-up for the initial position of the ball and the learning agent was the same as described in Section 6.1.1.

6.3.2 Results and discussions

Figure 6.12 shows the learning curve of Experiment 3. According to the graph, the learning agent was able to learn the task at a success rate of approximately 95% after about 55 episodes. As in Experiment 1, an additional 500 test episodes were run using the learned MLP to see at which initial position of the ball the learning agent failed to score a goal. Figure 6.13 shows the result, with failure cases marked in red. As in Experiment 1, the learning agent tended to fail when the initial position of the ball was too close to the goal line and the shooting angle was too small. However, in Figure 6.13 other cases are present in which the learning agent failed to score a goal. This is due to the discrete nature of the action variables.

As the minimum angular velocity of the learning agent is $\pm\frac{\pi}{4}$ rad/s and the time-step in the simulator is 50 ms, the learning agent is able to aim at the target position only with an accuracy of $\frac{\pi}{80} = \frac{\pi}{4} \times \frac{1}{20}$ rad. This causes a difference between the ideal orientation and the actual orientation of the learning agent at the final position before

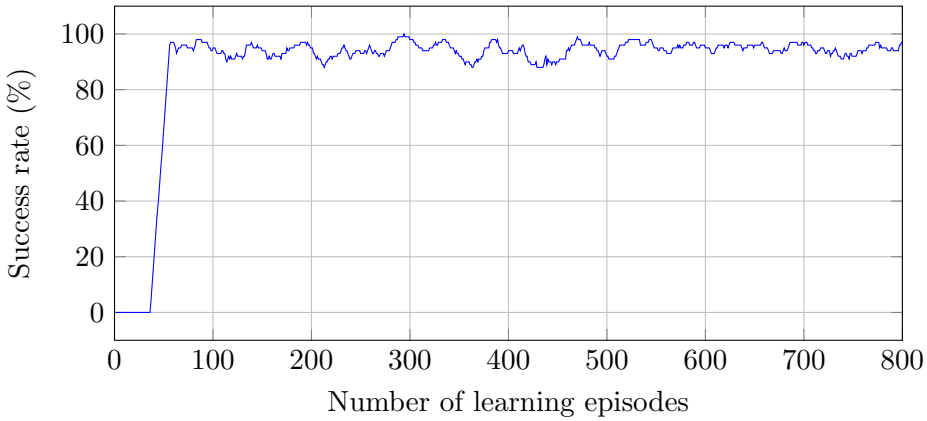


Figure 6.12: Result: shooting against a static goalkeeper.

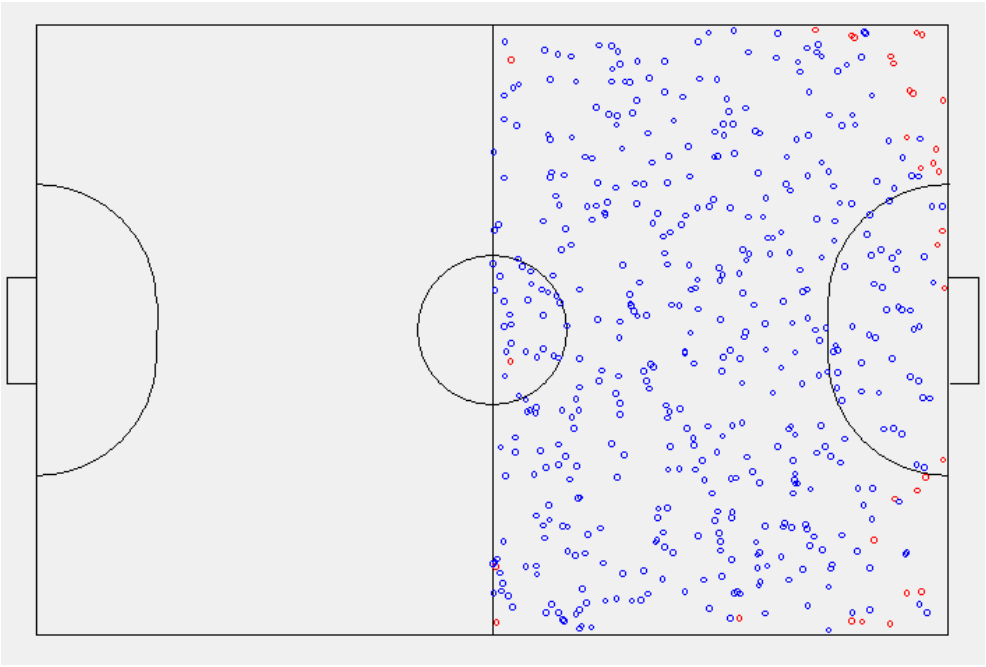


Figure 6.13: The initial position of the ball in 500 test episodes for Experiment 3. Failure cases are marked in red.

shooting. In most cases, the difference was not problematic. In some cases, however, the difference became significant depending on the initial orientation of the learning agent. For example, in the failure case marked in red inside the centre circle in Figure 6.13, the initial position of the ball was $(119, -208)$. This required the learning agent's ideal orientation at the final position before shooting to be $\arctan\left(\frac{-300-(-208)}{3025-119}\right) = -0.031648$ rad as it aimed at the target of $(3\,025, -300)$ in this case. However, with the accuracy mentioned above the best orientation the learning agent could make from its initial orientation of, in this case, 2.855020 rad, was -0.050950 rad. The difference between the ideal orientation and the actual orientation was $-0.031648-(-0.050950) = 0.019302$ rad, which was significant in this case, as one made the shooting successful and the other failure. It was significant in this case because the distance the ball must travel was long, therefore the slightest difference in the aiming angle had an effect on the results.

It should be pointed out that in most cases the difference between the ideal and the actual orientation of the learning agent was not significant even if the distance between the stationary ball and the target was long. The initial orientation of the learning agent (which is also determined randomly) plays an important role here. For example, in the case when the ball's initial position was $(93, -144)$, which is the closest ball position from the failure case discussed in the previous paragraph, the initial orientation of the learning agent was 5.66651 rad. In this case, the learning agent was able to adjust its orientation up to -0.066896 rad while the ideal orientation was $\arctan\left(\frac{-300-(-144)}{3025-93}\right) = -0.053156$ rad. The difference of 0.013740 rad had no significant impact on the result of the experiment, as shown by all the successful cases in Figure 6.13.

6.3.3 Conclusion: Experiment 3

The aim of the RL experiment in this section was to develop a skill for shooting a stationary ball against a static goalkeeper. It was assumed that the goalkeeper stood still in the centre of the goal all the time. The learning agent achieved a success rate of 95%. The failure cases were mainly due to the initial position of the ball, the same cause seen in Experiment 1. In addition, another type of failures was found in this experiment, i.e., the difference between the ideal and the actual orientation of the learning agent, due to the discrete nature of the action variables. The difference exists in all cases to some degree, but in some cases it had an effect on the result.

6.4 Experiment 4: shooting against a dynamic goalkeeper

6.4.1 Experimental set-up

In this experiment, the learning agent plays against a dynamic goalkeeper. The goalkeeper is initially located at a random position on the goal line between the two goal posts. It stands still at its initial position until the ball is kicked. Once the goalkeeper detects that the ball is moving, it moves to block the ball, but the goalkeeper is not allowed to move off the goal line. It can only move along the goal line. Also, the maximum speed of the goalkeeper is limited to 2 m/s, which is the same as the attacking robot's maximum speed.

When the goalkeeper detects that the ball is moving, it calculates the intersection of the ball's trajectory and the goal line from the current path of the ball. The goalkeeper moves to the intersection to block the ball at the maximum speed, but only *after* a certain time delay. This is to give some reality in the goalkeeper's behaviour. If the time delay is not modelled and thus the goalkeeper moves immediately at the maximum speed, it would be able to block most shootings. In reality, however, it takes time for a static goalkeeper to reach a certain velocity. Since acceleration is not modelled in the simulator, the concept of time delay was employed instead.

Two experiments were conducted, one with a 50 ms time delay (equivalent to one time-step in the learning process), and the other with a 100 ms time delay (equivalent to two time-steps). These values proved quite favourable to the goalkeeper. Reaching, for example, the maximum speed of 2 m/s in 100 ms, is equivalent to having an acceleration of 20 m/s^2. Given the fact that the acceleration usually reported in the literature is 3–3.5 m/s^2 (Goto *et al.*, 2013; Sharbafi *et al.*, 2011), the 50 ms and 100 ms time delays make the goalkeeper far more competitive.

As mentioned earlier, the initial position of the goalkeeper is randomly determined on the goal line between the two goal posts. In this case, to aim at one corner of the goal (the one farther from the initial position of the goalkeeper) is a better strategy than to aim at the centre of the goal. As in the case of Experiment 3, the strategy was implemented by using the farther end of the goal as the target position. A point slightly shifted to the centre of the goal produced better results than one at the very edge of the goal. Therefore $(3\,025, 330)$ and $(3\,025, -330)$ were used rather than $(3\,025, 350)$ and $(3\,025, -350)$, as presented in Figure 6.14.

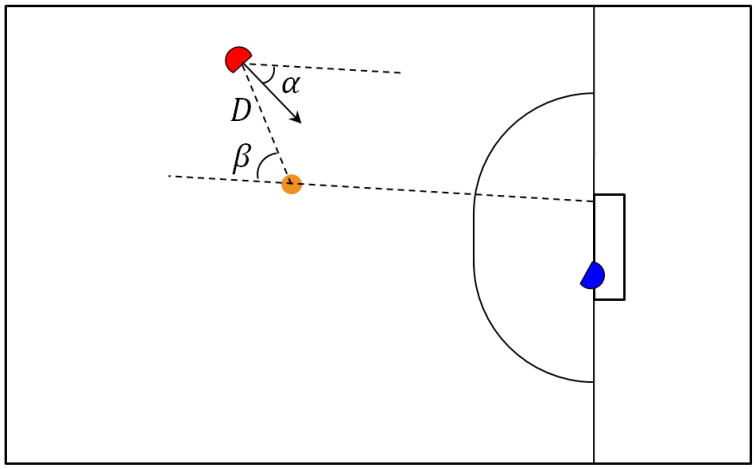


Figure 6.14: Aiming at one of the corners of the goal depending on the initial position of the goalkeeper.

6.4.2 Results and discussions

Figure 6.15 shows the learning curves of the two experiments, with a 50 ms delay (in red) and a 100 ms delay (in blue) in the goalkeeper's reaction time. With the 50 ms time delay, the success rate reached approximately 76% after about 65 episodes. In the case of learning against a goalkeeper with a 100 ms time delay, the learning agent was able to learn the task after about 65 episodes with an approximate success rate of 86%.

To analyse the failure cases, 100 test episodes were run for both experiments using the learned MLPs. The success rate was 78% in the experiment with the 50 ms time delay, and 91% in the other experiment. After a detailed analysis, it was found that the failure cases can be categorised into three types.

- Type 1: Failure occurred when the initial position of the goalkeeper was close to the centre of the goal and the distance between the initial position of the ball and the target was long.
- Type 2: Failure occurred when the shooting angle from the initial position of the ball was too small because of the position of the goalkeeper.
- Type 3: Failure occurred when the learning agent failed to aim at the target accurately due to the discrete action variables discussed in Section 6.3.2.

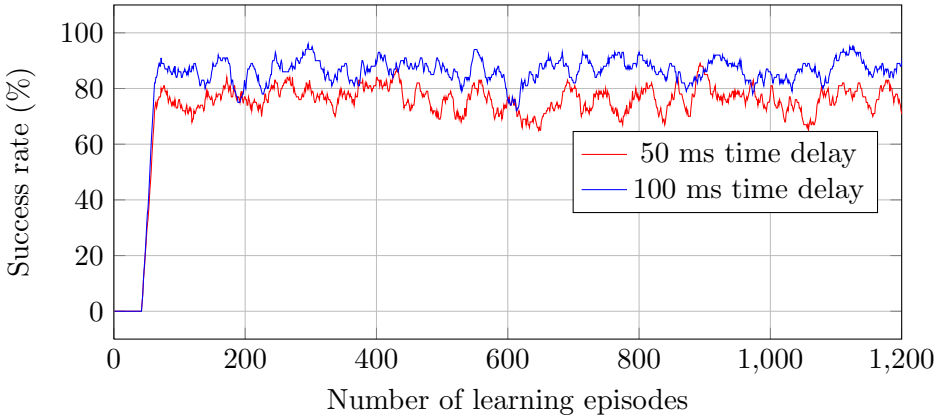


Figure 6.15: Result: shooting against a dynamic goalkeeper.

Table 6.1 shows the number of failure episodes in each failure type for both experiments. In the first failure type, the goalkeeper was able to block the shot even if it had a time delay. Because the goalkeeper was placed close to the centre of the goal (within 67 mm or less) and the ball was initially located far from the target (at least farther than 2 776 mm), the goalkeeper had more chance to block the shot. As might be expected, failure cases in this type were noticeably reduced in number when the goalkeeper moved with a longer time delay of 100 ms.

Table 6.1: The number of failure episodes in each category in Experiment 4.

	Number of failure episodes	
	50ms time delay	100ms time delay
Type 1	7	1
Type 2	4	4
Type 3	11	4
Total	22	9

The second failure type occurred when the ball was located initially close to the goal line and the goalkeeper was positioned to the same side as the ball in terms of the y-axis, therefore the shooting angle was effectively blocked by the goalkeeper. Figure 6.16 shows the initial position of the ball in an example failure episode in this type.

Besides these two failure types, the learning agent sometimes failed. This final type of failure results from the discrete nature of the action variables, as discussed in Section 6.3.2.

Another 100 test episodes were conducted to confirm the performance of the learned MLP under the condition of the penalty kick. As the law of the RoboCup SSL (Laws of the RoboCup Small Size League, 2013) requires the goalkeeper to remain on the goal line between the two goal posts when defending the penalty kick and prohibits the goalkeeper to move off the goal line before the ball is kicked, the goalkeeper's behaviour assumed in this experiment (described in Section 6.4.1) is close to the strategy any goalkeeper might adopt in defending the penalty kick, i.e., standing in the centre of the goal and trying to block the shot (once it detects that the ball has been kicked) by moving in the direction of the ball along the goal line. Therefore, in these additional test episodes, the goalkeeper was initially placed in the centre of the goal and its behaviour was set as described in Section 6.4.1. A time delay of 50 ms was applied to the goalkeeper. The ball was always placed on the penalty mark, i.e., at the position of $(2\,275, 0)$ (see Figure 4.6 for the position of the penalty mark). The learning agent was placed in a random position on a half circle with a radius of 750 mm and the ball in the centre. Figure 6.17 illustrates the initial position of the learning agent in a penalty kick condition. The learning agent is randomly positioned on the red half circle facing the ball. In all 100 test episodes the learning agent succeeded in scoring a goal from the penalty kick.

6.4.3 Conclusion: Experiment 4

In this section, an RL experiment was discussed to develop shooting skills against a dynamic goalkeeper. The ball was assumed to be stationary, and the goalkeeper was allowed to move along the goal line to block the shot. The results showed a success rate of 76% and 86% against a goalkeeper with a 50 ms and 100 ms time delay, respectively. Besides occurring due to the discrete nature of the action variables, the failures occurred only when the ball was positioned either far from the goal (thus the goalkeeper had more time to block the shot), or when the shooting angle was effectively blocked by the goalkeeper. Also, the learned MLP for this task worked very well for penalty kicks, showing a 100% success rate in an additional experiment.

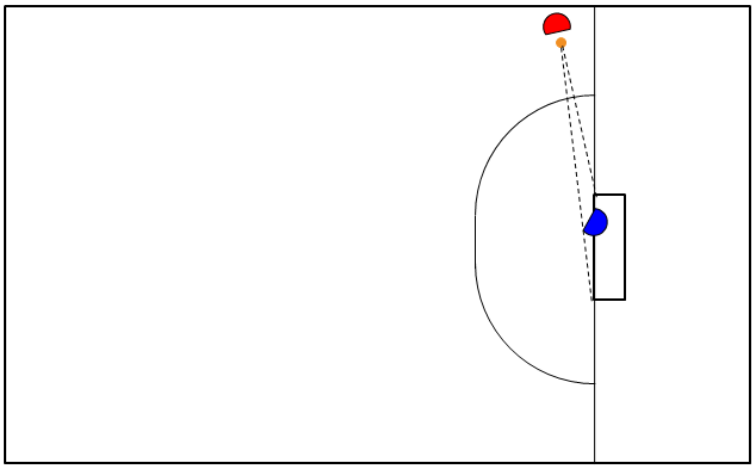


Figure 6.16: Example of an initial situation in the second failure type.

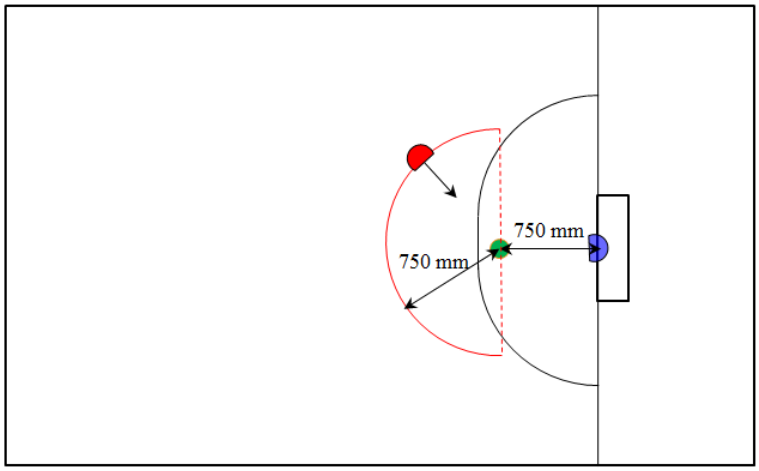


Figure 6.17: An example of the initial position of the learning agent for a penalty kick. The ball is always placed on the penalty mark, which is represented by the green circle in this figure. The learning agent is placed in a random position on the red half circle facing the ball.

6.5 Experiment 5: shooting against a more competitive goalkeeper

6.5.1 Experimental set-up

Experiment 5 is to score a goal against a more competitive goalkeeper. The goalkeeper is now allowed to come forward off the goal line as long as it stays within the defence area. The goalkeeper may also move before the ball is kicked. To minimise the open angle for shooting, the goalkeeper comes forward as soon as the episode starts. It moves towards the point on the border of the defence area where the line connecting the ball and the centre of the goal intersects. (See Figure 6.18. The target position is marked with a red arrow.)

If the goalkeeper reaches the target position before the ball is kicked, it waits there until the ball is moving since the goalkeeper is not allowed to move outside the defence area. When the goalkeeper detects that the ball is moving, whether it has reached the target position or not, it tries to block the ball by moving to the closest point on the path of the ball from its current position. Figure 6.19 shows the new target position of the goalkeeper in both cases.

The goalkeeper's maximum speed is limited to 2 m/s, as in Experiment 4. It moves towards its first and then to the second target position at the maximum speed, but a certain amount of time delay is applied before it starts to move to its second target position. As in Experiment 4, a 50 ms time delay and a 100 ms time delay were used in two separated experiments.

As the goalkeeper is allowed to move about within the defence area before the ball is kicked, the defence area is excluded when the algorithm determines the initial position of the ball. Also, the experiment is designed such that the learning agent aims at one of the corners of the goal depending on the position of the goalkeeper, as described in Section 6.4.1.

6.5.2 Results and discussions

The results of Experiment 5 are presented in Figure 6.20. Against the goalkeeper with the 50 ms time delay, the learning agent achieved a success rate of approximately 52% after about 40 learning episodes. The success rate rose to approximately 69% against the goalkeeper with the 100 ms time delay.

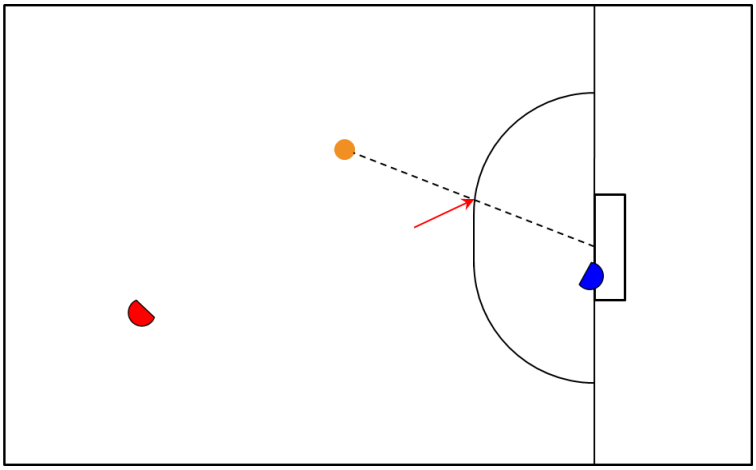


Figure 6.18: Target position of the goalkeeper when the episode starts.

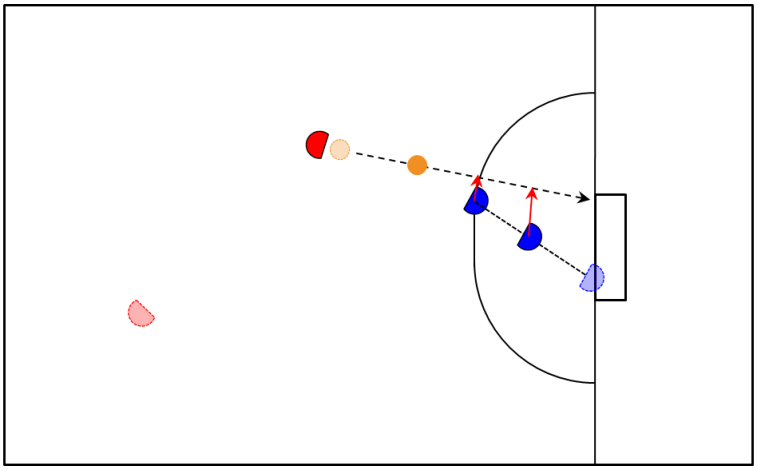


Figure 6.19: Target position of the goalkeeper when it detects that the ball is moving.

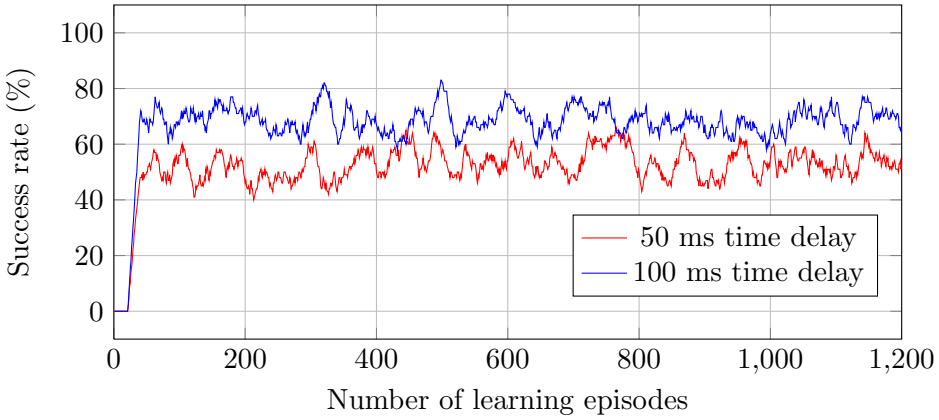


Figure 6.20: Result: shooting against a more competitive goalkeeper.

Table 6.2: The number of failure episodes in each category in Experiment 5.

	Number of failure episodes	
	50 ms time delay	100 ms time delay
Type 1	12	17
Type 2	9	5
Type 3	31	6
Total	52	28

To investigate the failure cases, 100 test episodes were run for both experiments with the learned MLPs. The success rates were 48% and 72% for the 50 ms and 100 ms time delays, respectively. Figure 6.21 and Figure 6.22 show the results, with failure cases marked in red. The causes of failures can be categorised as follows:

- Type 1: Failure occurred when the initial position of the ball was too close to the defence area, therefore the shooting angle was effectively blocked by the goalkeeper.
- Type 2: Failure occurred when the initial position of the ball was close to the goal line, therefore the shooting angle was effectively blocked by the goalkeeper.
- Type 3: Failure occurred when the learning agent failed to aim at the target accurately due to the discrete action variables discussed in Section 6.3.2.

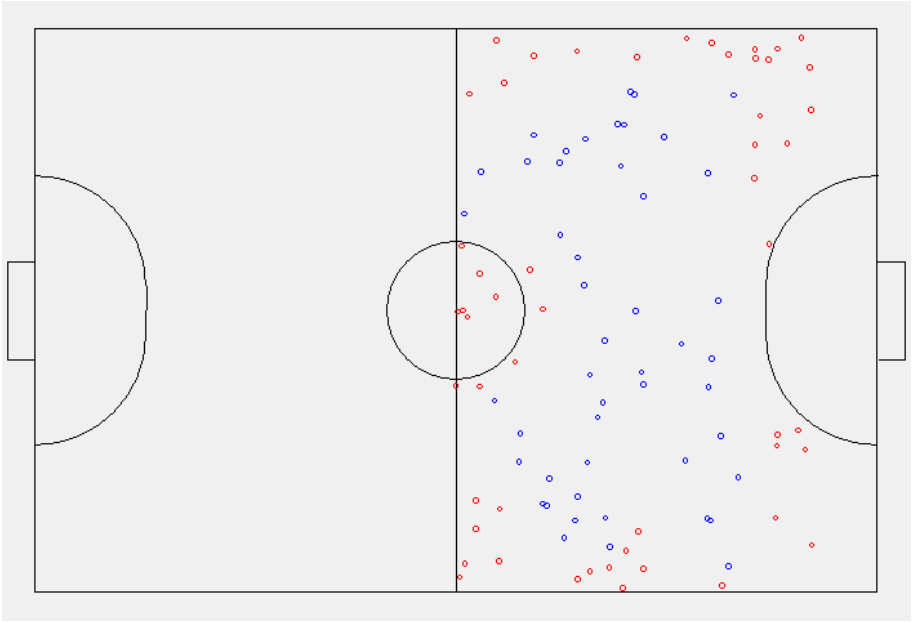


Figure 6.21: The initial position of the ball in 100 test episodes for Experiment 5 (against a goalkeeper with a time delay of 50 ms). The failure cases are marked in red.

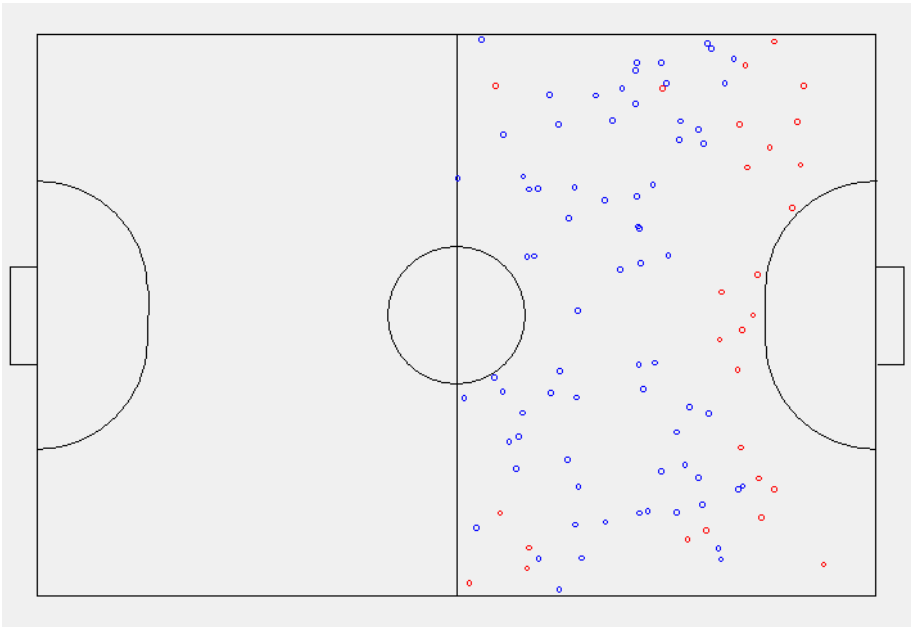


Figure 6.22: The initial position of the ball in 100 test episodes for Experiment 5 (against a goalkeeper with a time delay of 100 ms). The failure cases are marked in red.

Table 6.2 shows the number of failure episodes in each type for both experiments. The first two types occur when the goalkeeper blocks the shot by moving to its first target position. In most of the test episodes the goalkeeper was able to reach the target position on the border of the defence area before the ball was kicked. Depending on the initial position of the ball, in some cases it was enough for the goalkeeper to simply stand there in order to block the shot. Type 1 and Type 2 failures fall in this category. The causes of Type 3 failures were discussed in Section 6.3.2. The number of Type 3 failures has noticeably increased compared to the results of Experiment 4, especially in the experiment against the goalkeeper with the 50 ms time delay. It can be interpreted that with the goalkeeper moving forward, the shooting angle is considerably reduced in the episodes of this experiment (Experiment 5). In this case, the slightest gap between the ideal and the actual orientation of the learning agent often makes a big difference in the result.

6.5.3 Conclusion: Experiment 5

This section discussed an RL experiment to develop a shooting skill when the goalkeeper was allowed to come forward off the goal line in order to reduce the shooting angle. The learning agent was able to score a goal about 52% and 69% of the time against a goalkeeper with a 50 ms and a 100 ms time delay, respectively. As in the previous two experiments, the initial position of the ball was the reason for a considerable number of failures. That is, the learning agent failed when the ball was placed in a position where the shooting angle is effectively blocked by the goalkeeper. Also, the number of failures due to the discrete nature of the action variables has noticeably increased compared with the results of the previous two experiments, as the goalkeeper is more competitive in this experiment.

6.6 Experiment 6: shooting a moving ball against a dynamic goalkeeper

In the previous three sections (Sections 6.3 to 6.5), shooting skills against a goalkeeper (with different behaviours) were developed assuming that the ball is static. This section and the next one (Sections 6.6 and 6.7) deal with RL experiments to develop shooting skills against a goalkeeper when the ball is moving.

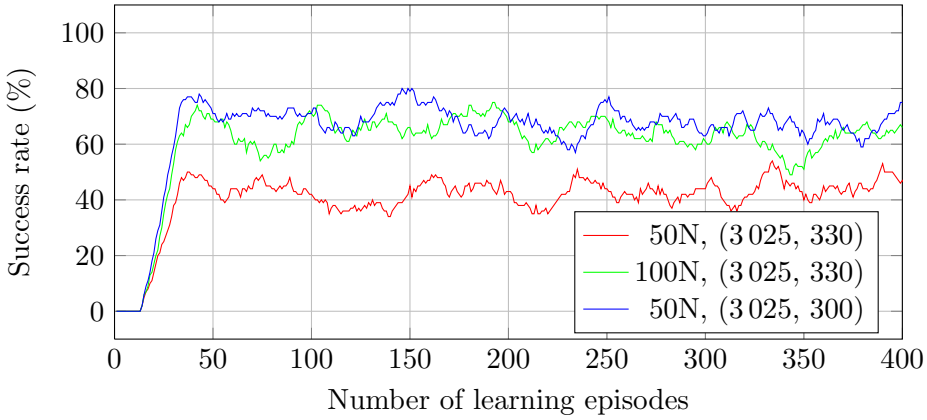


Figure 6.23: Result: shooting a moving ball against a dynamic goalkeeper.

6.6.1 Experimental set-up

The initial state set-up of this experiment is the same as in Experiment 2, which is described in Section 6.2.1. In the beginning of the episode, the ball is randomly positioned in the lower half of the opponents' field and the learning agent is located at a random position in the other half of the opponents' field. The ball moves with a constant speed of 1 m/s in a random direction between the initial position of the learning agent and the centre of the goal (Figure 6.6). The behaviour of the goalkeeper in this experiment is the same as in Experiment 4, which is described in Section 6.4.1. Only a 50 ms time delay was used for the sake of simplicity.

6.6.2 Results and discussions

In this experiment, the learning agent reached a success rate of 40% after about 30 learning episodes (see the red plot in Figure 6.23). The performance seems to be disappointing compared to the results of the corresponding experiments, i.e., 96% in Experiment 2 (shooting a moving ball without a goalkeeper) and 76% in Experiment 4 (shooting a stationary ball against a dynamic goalkeeper). An investigation into failure cases showed that the failure was mainly due to the slight difference between the target direction and the direction of the actual trajectory the ball followed after being kicked. As illustrated in Section 4.4.2 and Figure 4.8, when the ball is kicked, a force of 50N is exerted on the ball in the direction of the robot's orientation, which is aimed at the target position. However, the ball did not move exactly towards the target in

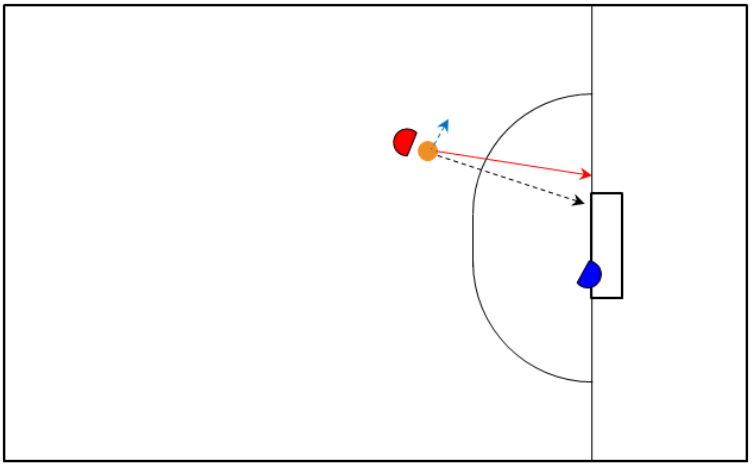


Figure 6.24: The difference between the target direction and the moving direction of the ball after being kicked. The blue dotted arrow represents the original moving direction of the ball and the black dotted arrow represents the target direction. The red arrow shows the actual direction in which the ball moves after being kicked.

this experiment, as it was moving in another direction when the contact was made. Figure 6.24 illustrates this concept. The blue dotted arrow and the black dotted arrow represent the moving direction of the ball and the target direction, respectively. The red arrow shows the actual direction in which the ball moves after being kicked.

The difference between the target direction and the actual trajectory of the ball after being kicked did not have a significant impact on the results in Experiment 2, where the goalkeeper was not considered and the target was the centre of the goal. In this experiment, however, it had an important effect on the results, especially when the initial position of the goalkeeper was on the lower half of the opponents' field, thus the target was set to $(3\,025, 330)$. As seen in Figure 6.24, the actual trajectory of the ball after being kicked tends to aim at a higher position than the target, for the ball moves originally to the upper half of the opponents' field (by the experimental design) and the target direction is always on the right side of the field. This caused a failure in many cases when the target was set to the upper corner of the goal $(3\,025, 330)$, by effectively aiming at a position outside of the goal. On the other hand, when the target was set to the lower corner of the goal $(3\,025, -330)$, it had the effect of aiming at a position closer to the centre of the goal, which would give the goalkeeper greater chance to block the shot. This did not occur very often, however, because in most cases the

ball was fast enough to beat the goalkeeper.

Many of these failure cases could be solved by either applying a bigger force when the ball is kicked, or changing the target of the upper corner of the goal to a position slightly lower than the current one. The blue and green plots in Figure 6.23 show the results of two additional experiments with the same experimental design as Experiment 6, except: 1) the force exerted on the ball (100N for the green plot), or 2) the target position for the upper corner of the goal ($(3\,025, 300)$ for the blue plot). The target for the lower corner of the goal was retained at $(3\,025, -330)$ for the reasons discussed in the previous paragraph. The success rate increased to between 60 and 70% in these experiments, which proves the significance of the difference in question.

It should be mentioned, however, that although the success rate has noticeably increased in these additional experiments, the difference between the target direction and the actual trajectory of the ball after being kicked was still responsible for about 60% of failure cases. (The remaining failure cases were mainly because the initial state of the episode was set such that it was impossible for the learning agent to score a goal.) This is because the magnitude of the difference varies depending on the original moving direction of the ball and the target direction at the shooting point, which differ in each episode. In some episodes the bigger force or the lowered target was not sufficient to score a goal.

If the learning agent had been given information about the difference between the moving direction of the ball and the target direction (as shown in Figure 6.25, denoted by φ), it might have been able to learn the task better by adjusting its target when necessary. A few preliminary experiments were conducted with the new state variable φ in the hope of getting better results. These attempts, however, did not manage to reach convergence. It seems that adding a new input to the MLP requires considerable changes to the current parameter settings (such as the structure of the MLP, the learning rate and the decreasing factor of the learning rate) for the new experiment to produce better results. It is believed that the algorithm would be able to perform better with a new state variable if the right parameter settings can be found and used appropriately. This work is left for future studies along with the effects of parameter settings.

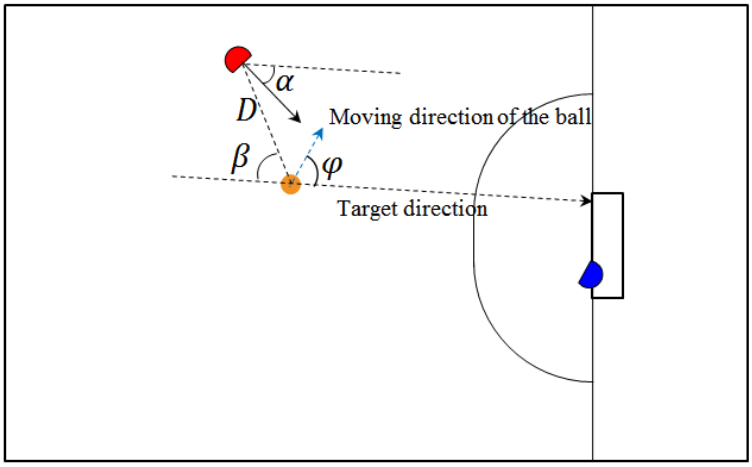


Figure 6.25: A new state variable required for the learning agent to adjust the target taking into account the moving direction of the ball and the target direction. The moving direction of the ball is indicated by the blue dotted arrow, and the black dotted arrow shows the target direction.

6.6.3 Conclusion: Experiment 6

In this section, the learning agent had to deal with a moving ball against a dynamic goalkeeper. It showed a success rate of about 40%, mostly due to a new type of failure cause, i.e., the difference between the target direction and the actual moving direction of the ball after being kicked. To work out this problem, two additional experiments were conducted: one with a bigger force, the other with a lowered target. These two experiments raised the success rate to between 60 and 70%, indicating the significance of the difference in question. It was mentioned that an additional state variable representing the difference between the moving direction of the ball and the target direction (denoted by φ in Figure 6.25) is required for the learning agent to be able to perform better. It was also mentioned that more attention needs to be paid to find the right parameter settings for this new experiment with the added state variable.

6.7 Experiment 7: shooting a moving ball against a more competitive goalkeeper

6.7.1 Experimental set-up

Experiment 7, the last learning experiment, is to score a goal against a more competitive goalkeeper when the ball is moving. The initial set-up of the episode is the same as described in Section 6.6.1, and the behaviour of the goalkeeper is the same as in Experiment 5, which was explained in Section 6.5.1, except that, with the ball moving in this experiment, the goalkeeper also moves after reaching the first target on the border of the defence area. It tries to locate itself at the position on the border of the defence area where the line connecting the current position of the ball and the centre of the goal intersects. Figure 6.26 shows an example of the behaviour of the goalkeeper in this experiment. The red arrow and the blue arrow indicate the moving direction of the ball and the goalkeeper, respectively. The black arrows show the corresponding positions of the ball and the goalkeeper at the same time-step.

6.7.2 Results and discussions

As might be expected from Figure 6.26, the task aimed at in this experiment turned out to be very challenging. In most cases, the goalkeeper was able to defend the goal effectively with the behaviour illustrated in Figure 6.26. Figure 6.27 shows the learning curve of the experiment (marked in red), as well as the results of two additional experiments (one with a bigger force, the other with a lowered target for the upper corner of the goal, marked in green and blue, respectively.) In the original experiment, the success rate reached approximately 20% after about 35 learning episodes. In the additional experiment with the bigger force the performance was slightly better (25% on average), but adjusting the target did not improve the result. This can be interpreted that most of the failures are due to the competence of the goalkeeper rather than the difference between the target direction and the actual moving direction of the ball after being kicked, which was discussed in Section 6.6.2.

It was observed that the shooting point, i.e., the position where the shot is taken, plays a key role in determining the success or failure of the episodes. Although this is true for other experiments as well, it is of the utmost importance in this experiment, as the possible area for a successful shooting point is tightly limited under the conditions

6.7 Exp. 7: shooting a moving ball against a more competitive goalkeeper

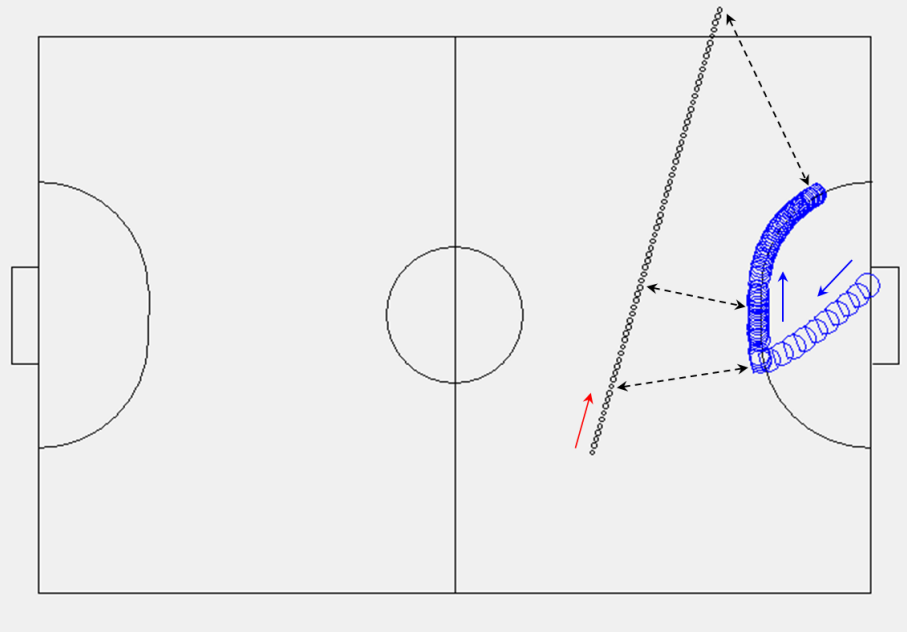


Figure 6.26: The behaviour of the goalkeeper in Experiment 7. The red arrow and the blue arrow indicate the moving direction of the ball and the goalkeeper, respectively. The black arrows show the corresponding positions of the ball and the goalkeeper at the same time-step.

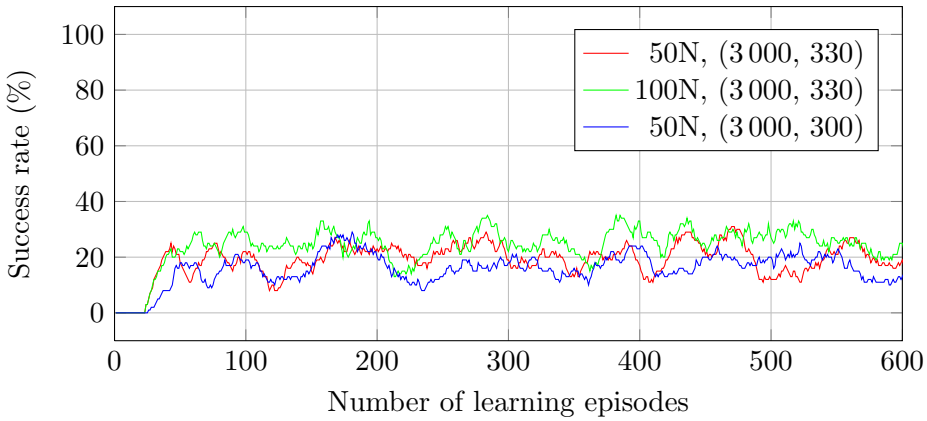


Figure 6.27: Result: shooting a moving ball against a more competitive goalkeeper.

6.7 Exp. 7: shooting a moving ball against a more competitive goalkeeper

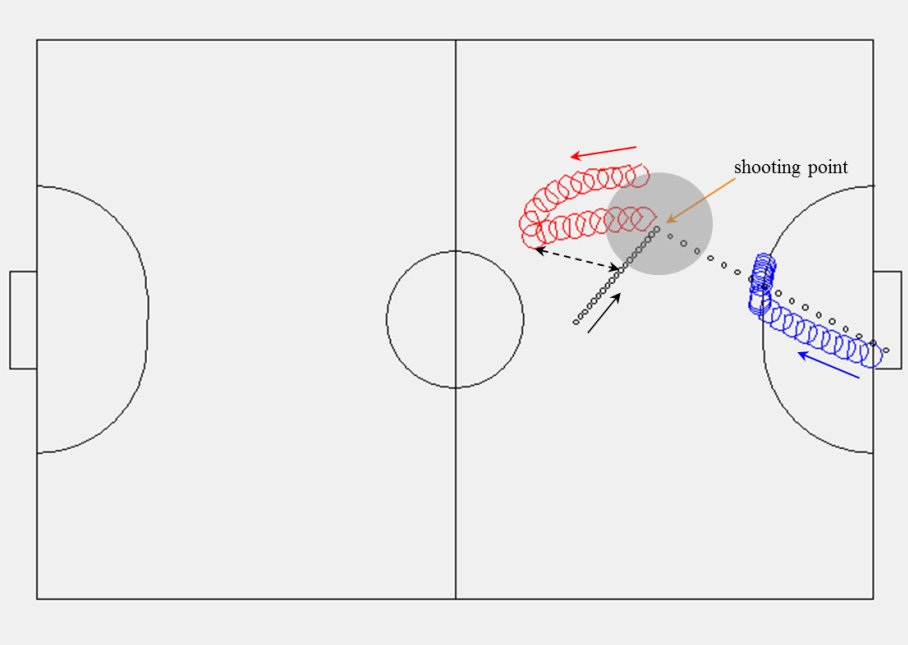


Figure 6.28: The trajectory of the ball and the robots in a success case.

of this experiment. Figure 6.28 shows the possible area (indicated by the grey circle), along with an example of a success case. It was almost impossible to score a goal when the shooting occurred outside of this area. If the shooting point is too close to the goalkeeper or if it is close to the goal line, the open angle is too small, which makes it hard to score a goal. On the other hand, if the shooting point is too far from the goal, a slight difference in aiming at the target could result in missing the goal or the goalkeeper blocking the shot.

In the example success case in Figure 6.28, the learning agent was able to score the goal by aiming at the lower corner of the goal. However, even if the shooting occurs in this area, it does not always lead to a score. The difference between the target direction and the actual moving direction of the ball after being kicked, which was discussed in Section 6.6.2, plays a role here. Depending on the original moving direction of the ball and the target direction at the shooting point, in some cases the difference was big enough to cause a failure.

In summary, the learning agent was able to score a goal in this experiment on rare occasions if all the following conditions were met:

- The ball passes through the possible area designated in Figure 6.28.

- The initial position of the learning agent is close enough to the possible area for the learning agent to reach the ball in time.
- The original moving direction of the ball is not far apart from the target direction at the shooting point, so that the difference between the target direction and the actual moving direction of the ball after being kicked does not have much of an effect on the result.

6.7.3 Conclusion: Experiment 7

In this section the most difficult task in this research was discussed: shooting a moving ball against a more competitive goalkeeper. The goalkeeper has turned out very competitive with the behaviour illustrated in Figure 6.26. The success rate reached was only 20–25%, even with the bigger force or the lowered target. This means most failures are due to the competence of the goalkeeper. With the behaviour employed by the goalkeeper in this experiment, the learning agent had little chance to succeed.

6.8 Experiments 8 and 9: developing passing skills

In the previous seven sections (Sections 6.1 to 6.7) RL experiments to develop shooting skills in various situations were discussed. This section focuses on passing skills in two scenarios: passing a stationary ball and passing a moving ball. These passing skills, however, were not developed by conducting RL experiments, as was the case for the shooting skills in the previous sections. Rather, the results of Experiments 1 and 2 were put to good use to implement the passing skills.

As mentioned in Section 5.2, the target position was not used directly as one of the state variables, but it was used indirectly by the algorithm in the calculation of the two state variables α and β. This opens the possibility that the learned MLPs for shooting tasks might also work for passing tasks when used with a different target position.

Two test experiments, Experiments 8 and 9, were designed and conducted to confirm the possibility. The aim of Experiment 8 is to develop a passing skill with a stationary ball, and that of Experiment 9 is to do the same with a moving ball. In both experiments no opponent players were assumed, nor was there a pass-receiving player. The pass-receiving skill should be developed separately as a different type of individual soccer skill. Also, high-level team strategies should be developed carefully so that passing

skills can be used only when there exists a clear path between the passing robot and the target position.

6.8.1 Experiment 8: passing a stationary ball

6.8.1.1 Experimental set-up

This experiment is to develop a passing skill with a stationary ball using the result of Experiment 1 (Section 6.1). It consists of 10 tests, each with a different target position. In each of the 10 tests, 20 test episodes were run with different initial positions of the ball and the learning agent, but the target position remained the same over the 20 test episodes.

The target position, the initial position of the ball and the initial position of the learning agent were determined randomly. The target position was determined first, and the ball was placed in an arbitrary position such that the distance between the target position and the ball was no more than 2 000 mm. The learning agent was randomly placed in any position on the field.

In the test episodes, the learning agent used the MLP learned from Experiment 1 to determine its next action. Instead of the centre of the goal the randomly selected target position was used in the algorithm to compute the state variables α and β. A force of 50N was exerted on the ball for the kick action. The episode was considered a success if the ball passed through a circle with a radius of 50 mm of which the centre is the target position. Therefore, it can be said that the target segment in this experiment (100 mm) is narrower than that of Experiment 1, which is the width of the goal (700 mm).

6.8.1.2 Results and discussions

The result of this experiment turned out to be very promising. In all 10 tests and in all 20 episodes in each test, the learning agent succeeded in passing the ball to the designated target position. Figure 6.29 presents the results of the experiment in the form of a bar chart, where each bar represents the success rate of each test (the number of successes in the 20 test episodes was presented as a percentage). In this experiment it shows a 100% success rate in all 10 tests, which is better than expected, considering the reduced target segment.

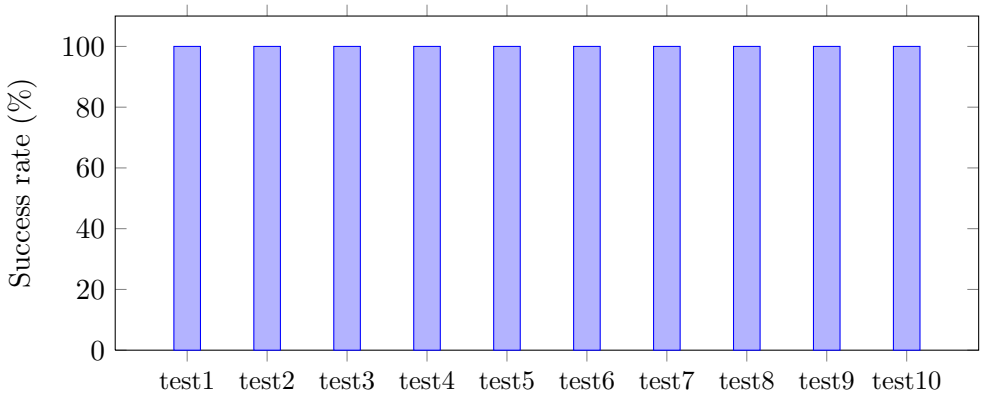


Figure 6.29: Result: passing a stationary ball.

6.8.2 Experiment 9: passing a moving ball

6.8.2.1 Experimental set-up

This experiment is to develop passing skills with a moving ball, using the result of Experiment 2 (Section 6.2).

As in Experiment 8, the experiment consists of 10 tests, each with 20 test episodes. The learning agent uses the MLP learned from Experiment 2, and each test episode starts with the ball moving with a constant speed of 1 m/s in a random direction between the initial position of the learning agent and the target. Figure 6.30 illustrates the initial state set-up for this experiment. In addition, another experiment is performed with a force of 100N, as the problem of the difference between the target direction and the actual moving direction of the ball after being kicked (discussed in Section 6.6.2) is expected to occur in this experiment.

6.8.2.2 Results and discussions

The results of this experiment are presented in Figure 6.31. With the force of 50N, the learning agent was able to pass the ball to the target area approximately 74% of the time on average, while the success rate increased to an average of 96% with the force of 100N.

The failure cases were analysed and it was observed that failure occurred either because the initial state of the episode was determined such that a failure was unavoidable, or due to the difference between the target direction and the actual moving

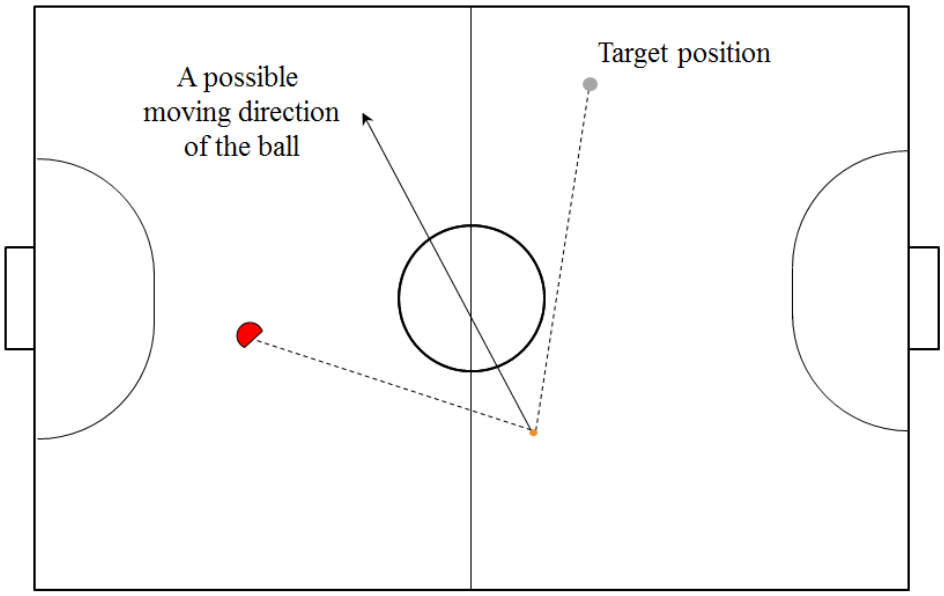


Figure 6.30: The initial state set-up for Experiment 9. The target position, the initial position of the ball and the initial position of the learning agent are randomly determined. At the beginning of each episode, the ball moves with a constant speed of 1 m/s, in a random direction between the two dotted lines.

direction of the ball after being kicked, which was discussed in Section 6.6.2. When the initial set-up was responsible for the failure, the learning agent was initially positioned too far from the ball, thus it had insufficient time to approach the ball, or the ball was placed relatively close to the edge of the field and rolled out of the field as soon as the episode started, which gave the learning agent little opportunity to follow the ball. In the experiment with the 50N force, approximately 23% of the failure cases were due to the initial set-up of the episode, and the rest was because of the aiming issue. In the other experiment with the 100N force, however, all failures were due to the initial set-up of the episode, i.e., the learning agent was able to aim at the target area in all cases when the circumstance allowed it.

6.8.3 Conclusion: Experiments 8 and 9

This section dealt with additional experiments to implement passing skills in two situations: when the ball is stationary and when the ball is moving. The skills were not

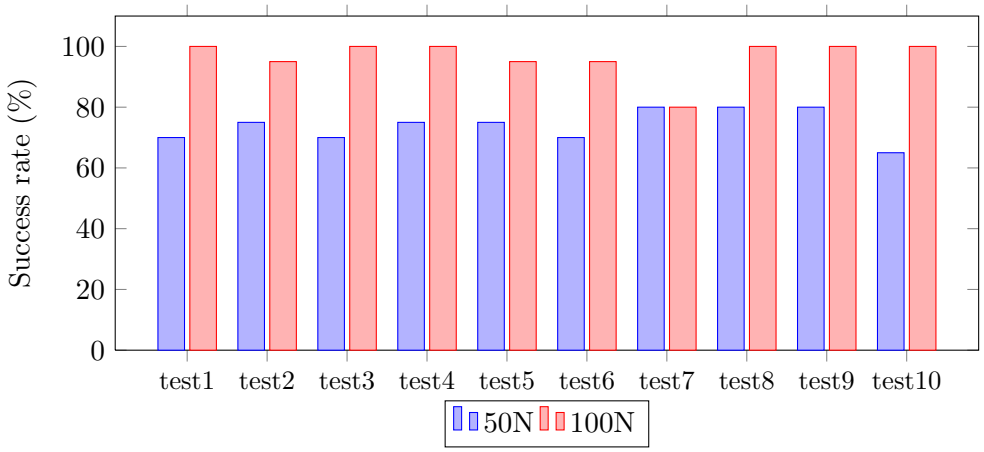


Figure 6.31: Result: passing a moving ball.

developed by means of RL experiments, but were implemented using the results of Experiments 1 and 2 discussed in Sections 6.1 and 6.2, respectively. For the task of passing a stationary ball, the experiment produced a very promising result of a 100% success rate despite the fact that the target segment was significantly reduced, compared to the one in Experiment 1. The learning agent also showed good performance in passing a moving ball, with an average success rate of 74% and 96% when equipped with a 50N and a 100N force, respectively. Failure cases occurred only when the initial set-up of the episode was determined such that failure was unavoidable, or when the learning agent could not kick the ball as hard as required to send the ball in the exact direction of intention.

6.9 Conclusion: Experimental results

This chapter provided details of seven RL experiments defined for developing shooting skills and two test experiments for implementing passing skills, which were not developed by means of RL but implemented using the results of the experiments done for shooting skills. Each section, from Sections 6.1 to 6.7, dealt with a separate RL experiment, describing the experimental set-up and discussing the results. Section 6.8 focused on the passing skills.

Table 6.3 presents a summary of all the experiments done along with the success rate achieved (rounded up) and, in case of learning experiments, the number of learning

episodes required to reach the performance. Although N_{Max}, the maximum number of learning episodes, was determined as discussed in Section 5.9 to confirm that there is no more updating in the algorithm, it was observed that in practice the performance of the learned MLP did not change much after the number of learning episodes presented in Table 6.3.

Table 6.3: Summary of the experiments.

	Exp1	Exp2	Exp3	Exp4 (50ms)	Exp4 (100ms)	Exp5 (50ms)	Exp5 (100ms)	Exp6 (50N, (3 025, 330))	Exp6 (100N, (3 025, 330))	Exp6 (50N, (3 025, 300))	Exp7 (50N, (3 025, 330))	Exp7 (100N, (3 025, 330))	Exp7 (50N, (3 025, 300))	Exp8	Exp9 (50N)	Exp9 (100N)
Stationary ball	✓		✓	✓	✓	✓	✓							✓		
Moving ball		✓						✓	✓	✓	✓	✓	✓		✓	✓
Goalkeeper			✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓			
Shooting skills	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓			
Passing skills														✓	✓	✓
Success rate (%)	99	96	95	76	86	52	69	40	64	70	20	25	18	100	74	96
Number of episodes	35	30	55	65	65	40	40	30	30	30	35	40	50			

The failure cases were analysed along with the results for each experiment. It was found that the learning agent failed in cases when either the initial set-up of the episode (such as the initial position of the ball and of the learning agent) is determined such that achieving the given task is very difficult, or when the learning agent could not aim at the target as accurately as required due to the discrete nature of the action variables discussed in Section 6.3.2 in the experiments dealing with a stationary ball. In the experiments handling a moving ball, failures occurred due to the difference between the target direction and the actual moving direction of the ball after being kicked, as discussed in Section 6.6.2. In Experiment 6 in particular, the learning agent had difficulty in kicking the ball in the exact target direction due to the momentum of the moving ball. With the current design of the experiment, it was impossible for the learning agent to learn to adjust the target, taking into account the moving direction of the ball. However, in this case too, the learning agent was able to score a goal in

episodes where the difference in question did not have a significant impact. Therefore, it can be summarised that the learning agent was able to score a goal, or pass the ball to the designated area, whenever circumstances allowed.

From the discussions above, it is concluded that RL combined with MLP has effectively trained the learning agent to acquire shooting skills in various situations. Moreover, the learned MLPs for shooting tasks worked very well for the corresponding passing tasks. The learning agent showed the best performance under the given conditions and the ability with which it was equipped.

The thesis is concluded in the next chapter.

Chapter 7

Summary and conclusions

This chapter presents a summary of the work done in this research, the main findings of the study, the contribution of the research to the body of knowledge, and recommendations for future work.

7.1 Summary of the research

The primary purpose of this research was to lay a foundation for the development of a Decision-Making Module (DMM) for a robot soccer team at Stellenbosch University (SU). The project was focused on the RoboCup Small Size League (SSL) as it is the platform chosen by the research team at SU.

An intensive literature study was conducted (and presented in Section 3.1) to investigate how teams in the RoboCup SSL have developed their DMMs and the strategies it uses. The literature study revealed the following:

- Being considered as the easiest league, the RoboCup SSL served as an entry league for new teams joining the robot soccer projects. Therefore the main focus of the teams in this league falls on hardware development.
- Strategies are relatively less focused, thus not many studies about the development of strategies or DMMs have been published. Most of these published studies (presented in Table 3.1) are about individual strategies established given a situation during play.

- These strategies were developed using a hand-coded approach, i.e., the strategies were programmed using human expertise, while in other leagues the Machine Learning (ML) approach is widely used along with the hand-coded approach.
- Another distinguishing feature regarding the development of DMMs is that they are developed in a hierarchical structure, where individual soccer skills are implemented first as complete, independent modules and then used in the development of high-level team behaviours. Therefore, individual soccer skills can be defined as the basic building blocks of the DMM.

The findings above opened a research opportunity in the area of developing individual soccer skills as the basic building blocks of the DMM using the ML approach. Shooting and passing skills in particular were considered as they are the most important basic soccer skills. The published literature on the subject of ML applications in the development of soccer strategies was surveyed and presented in Section 3.2. The literature study showed the following:

- Among many ML techniques, Reinforcement Learning (RL) is the most widely used in the robot soccer domain due to the reasons discussed in Section 2.2.
- No individual soccer skills have been developed using ML in the RoboCup SSL context.
- Many strategies (including some individual soccer skills) have been developed using ML in other RoboCup leagues or in other robot soccer domains. But none of them were exactly the same as the shooting and passing skills intended to be developed in this research, which are complete, independent modules that can be used as the basic building blocks for the development of the DMM.

This led to the following research objective:

> **Developing shooting and passing skills as the basic building blocks of the DMM for the robot soccer team at SU using RL.**

RL was discussed in detail in Chapter 2. To be brief, the purpose of RL is to find the optimal state-value function V^*, or the optimal action-value function Q^*. Various RL algorithms to find these values were presented and explained, among which

the Temporal-Difference (TD) value iteration algorithm with state-value functions (Algorithm 5) was chosen to be used in this research. The concept of the Multi-Layer Perceptron (MLP), a typical supervised learning method, was also introduced in Chapter 2 as a function approximator to deal with the continuous state variables in RL. With a function approximator, the value functions V^* or Q^* are represented not by lookup tables but by a set of parameters $\vec{z}$, and the MLP uses the Back-Propagation (BP) algorithm to find a set of parameters $\vec{z}$ that best approximates V^* or Q^*. The batch mode BP algorithm (Algorithm 7) was used in this research as it turned out to be more stable than the sequential mode BP algorithm.

Meanwhile, it was realised that a simulation environment is essential in this research. In particular, a high-level simulator was required to deal with individual soccer skills or strategies to be included in the DMM. Existing simulators were investigated and it was found that none of them were appropriate for this research due to the reasons discussed in Section 4.1. Therefore, it was decided that a high-level simulator should be developed by the researcher, which added a second objective to this research:

> **Developing a high-level simulator to facilitate the development of the DMM.**

Chapter 4 describes in detail the developed high-level simulator as well as the simulation environment established in this research where the simulator and the DMM communicate with each other to simulate a soccer game. An open-source library called Open Dynamics Engine (ODE), was incorporated in the simulator to handle the motion dynamics of the robots and the ball accurately. Details of the developed simulator were presented in Section 4.4, including the validation of the simulator. Another type of simulator was developed separately for learning purposes. The learning simulator was used as a stand-alone program in the RL experiments discussed in Chapters 5 and 6.

Seven RL experiments were defined in Section 5.1 to develop shooting skills in various situations, i.e., shooting a stationary ball or a moving ball, shooting against a goalkeeper or without a goalkeeper, etc. Two test experiments were added to implement passing skills using the learned MLPs for some shooting skills. Table 5.1 summarises all nine experiments conducted in this research. The rest of Chapter 5 presents details of the experimental design including state and action variables used in each experiment, the definition of terminal states, the rewarding schemes, the algorithm applied in the

experiments (RL combined with MLP using batch mode BP), stopping criteria and parameter settings.

RL with MLP was successfully applied to develop the individual soccer skills aimed at in this research. Sections 6.1 to 6.7 present the seven RL experiments to develop shooting skills. In each section, the learning curves were presented in order to show the success rate and the number of learning episodes required to reach the success rate. Section 6.8 focuses on the passing skills implemented using the results of the first two experiments which dealt with the corresponding shooting skills. The results of all experiments are summarised in Table 6.3. A discussion of the failure cases followed in each section after the results were presented, demonstrating that the learning agent had not failed without reasonable causes and showing that the learning agent achieved the best performance under the circumstances.

7.2 Important findings

The important findings from this research are as follows:

- RL combined with MLP works very well for the tasks given in this research, i.e., developing individual soccer skills. As mentioned in the previous section, the learning agent showed in all experiments the best performance under the given conditions and given the ability with which it was equipped.

- For RL combined with MLP, it is very important in the performance of the algorithm to match the learning speeds of the two types of estimations, i.e., one for the estimation of the optimal value functions V^* or Q^*, the other for the estimation of the parameters $\vec{z}$ to represent these value functions as accurately as possible. This issue was discussed in Section 6.2.2.

- The batch mode BP algorithm works better than the sequential mode BP algorithm for the tasks given in this research. The learning takes place quite fast in the batch mode BP algorithm in terms of the number of batches. In the seven learning experiments conducted in this research, the algorithm reached convergence after a maximum of 65 learning episodes, which corresponds to a maximum of 6 500 training data points (as a maximum of 100 iterations is allowed in an episode).

- Although the learning takes place quite fast in terms of the number of learning episodes, it took hours, or days in some experiments, for the algorithm to run until it reached the maximum number of learning episodes defined in Section 5.9. The greater the total number of actions is, the longer it takes for the algorithm to run until the ${N_{\text{Max}}}^{th}$ learning episode. Also, it tends to take longer in experiments where the task is more challenging, thus more failure cases are present in the experiment (e.g. Experiment 6 and Experiment 7).

- Parameters, especially the learning rate η and the decreasing factor ρ for η, have a strong influence on the performance of the learning experiments. The over-updating issue due to a too-high decreasing factor for the learning rate was discussed in Section 6.2.2. It was also observed that if a too-small value is used for the decreasing factor ρ, the algorithm converges to a bad solution in which the learning agent never succeeded in scoring a goal.

- Randomness is also important in the performance of the algorithm. It was easier for the learning agent to learn a task with episodes starting with random positions of the ball and the learning agent, than with fixed positions. In order to develop penalty kicking skills, for example, using the learned MLP in Experiment 4 in the set-up of the penalty kick was a better idea than designing a new learning experiment in which the ball is always placed at the penalty mark.

7.3 Contribution to the field

This research started as part of a robot soccer project at Stellenbosch University (SU) of which the aim is to build a team of soccer-playing robots for the RoboCup Small Size League (SSL). The task given to the researcher was to develop a Decision-Making Module (DMM) for the team. Being the first and only member of the software development part of this project, the researcher had to start from the very beginning, i.e., by developing a simulator. A high-level simulator was designed, developed and validated by the researcher, and a simulation environment was built using the developed simulator. In the simulation environment the simulator and a communication module of the DMM communicate with each other to simulate a soccer game. The result of

this research is a completed framework for simulation (except the DMM itself), which can be used for the testing of strategies and the validation of the DMM in the future.

For the development of the DMM itself, the researcher developed two types of important individual skills: shooting skills and passing skills. These individual skills form part of the basic building blocks of the DMM (along with other individual skills such as dribbling, receiving a pass, marking, intercepting, etc.), hence they can be used by high-level team strategies that constitute the high level of the hierarchical structure of the DMM.

The shooting skills and passing skills were developed by means of the Machine Learning (ML) approach. To the researcher's knowledge, this is the first application of ML to the development of strategies in the RoboCup SSL domain.

In conclusion, this research built a simulation environment and developed two most important individual soccer skills using Reinforcement Learning (RL) as the basic building blocks for the DMM. Although the DMM was not completely developed, the contribution of the work to the SU team is worth being recognised, especially considering the fact that a robot soccer development team usually consists of dozens of members over various disciplines.

7.4 Recommended future work

The following are some suggestions for future studies related to the work done in this research.

- To completely develop the DMM for the robot soccer team:
  - More individual soccer skills would have to be developed to add to the basic building blocks of the DMM.
  - High-level team strategies can then be developed to complete the DMM.
  - A heuristic function would need to be developed to reduce the time searching for the next action at each time-step.
- As mentioned in Section 4.4.2, for a more realistic simulator:
  - The acceleration application mechanism being applied to real robots should be investigated and incorporated in the simulator.

  - The friction coefficient should be measured using real robots and the ball, and this measured friction coefficient should be used in the simulator.

- To improve the performance of currently developed individual skills:
  - A new experiment can be designed and performed with a new state variable φ (see Figure 6.25) for the learning agent to handle a moving ball more precisely, as discussed in Section 6.6.2.
  - Using a variable speed for the ball would be required to enhance the competence of the learning agent in the experiments dealing with a moving ball (it was assumed that the ball moves at a constant speed of 1 m/s all the time).
  - Various magnitudes can be used for the force exerted on the ball when kicked.

- Regarding the RL combined with MLP algorithm:
  - A sensitivity analysis to learn the effects of the parameters used in the RL experiments, the learning rate η and its decreasing factor ρ in particular, would be interesting.
  - Possible reasons of the fast learning observed in Experiment 2, which caused poor performance of the MLP by over-updating the parameters, should be investigated, as discussed in Section 6.2.2. This would provide a clue for another future study of how to match the learning speeds of the RL and the decreasing factor of the learning rate of the MLP.
  - While outside the scope of this thesis, a comparision on the performance or efficiency between the ML approach and the hand-coded approach would be a topic worthy to study.

7.5 Conclusion: Summary and conclusions

This chapter concluded the research by presenting a summary of the research, important findings, contribution to the field and recommendations for future work. Following are some thoughts of the researcher in closing this research.

- Experience teaches one more than one thinks. Even failure experiences point one in the right direction. The learning agent experienced only failures until it finally succeeded for the first time.

- In RL combined with MLP, where the two algorithms cooperate to achieve a common goal, the performance becomes worse if the one learns too fast compared to the other. Going slowly together is a far better way than being first but alone.
- In most cases Machine Learning (ML) tries to learn the easiest type of tasks for human beings, for example, face or voice recognition or, as in this research, how to shoot at a goal. Computers may solve highly difficult problems without errors, some of which humans probably never can, but in soccer they need to be taught where to go, in which direction to aim and when to kick, to be able to shoot a stationary ball, while a three-year-old human child would understand them as soon as he or she hears the task, even though he or she might not yet be able to carry it out. It is amazing that human beings know such things without being taught. In this respect, we all, even the least capable, are more than we think we are.

References

Aamodt, A. & Plaza, E. (1994). Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Communications*, **7**, 39–59. 45

Alpaydin, E. (2010). *Introduction to Machine Learning*. Adaptive computation and machine learning, MIT Press. 9, 21, 28, 29

Back, T. (1996). *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, New York. 48

Ball, D. & Wyeth, G. (2004). UQ RoboRoos 2004: Getting Smarter. In *RoboCup 2004 Symposium CDROM, SSL Team Description Papers*. 39

Beck, D., Ferrein, A. & Lakemeyer, G. (2008). A simulation environment for middle-size robots with multi-level abstraction. In *RoboCup 2007: Robot Soccer World Cup XI*, 136–147, Springer. 52

Bianchi, R.A., Ribeiro, C.H. & Costa, A.H. (2008). Accelerating autonomous learning by using heuristic selection of actions. *Journal of Heuristics*, **14**, 135–168. 46

Bianchi, R.A., Ros, R. & De Mantaras, R.L. (2009). Improving reinforcement learning by using case based heuristics. In *Case-Based Reasoning Research and Development*, 75–89, Springer. 46

Bowling, M., Browning, B., Chang, A. & Veloso, M. (2004a). Plays as team plans for coordination and adaptation. In *RoboCup 2003: Robot Soccer World Cup VII*, 686–693, Springer. 39

Bowling, M., Browning, B. & Veloso, M. (2004b). Plays as effective multiagent plans enabling opponent-adaptive play selection. In *ICAPS*, 376–383. 39

Browning, B. & Tryzelaar, E. (2003). Übersim: a multi-robot simulator for robot soccer. In *Proceedings of the second international joint conference on autonomous agents and multiagent systems*, 948–949, ACM. 52

Browning, B., Bruce, J., Bowling, M. & Veloso, M. (2005). STP: Skills, tactics, and plays for multi-robot control in adversarial environments. *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, **219**, 33–52. 38

Buck, S., Beetz, M. & Schmitt, T. (2002). M-ROSE: A multi robot simulation environment for learning cooperative behavior. In *Distributed Autonomous Robotic Systems 5*, 197–206, Springer. 52

Budden, D. (2012). *Applications of Machine Learning Techniques to Humanoid Robot Platforms*. Ph.D. thesis, University of Newcastle, Australia. 41

Budden, D., Fenn, S., Mendes, A. & Chalup, S. (2013). Evaluation of colour models for computer vision using cluster validation techniques. In *RoboCup 2012: Robot Soccer World Cup XVI*, 261–272, Springer. 41

Busoniu, L., Babuska, R., De Schutter, B. & Ernst, D. (2010). *Reinforcement learning and dynamic programming using function approximators*. CRC Press. 24

Carpin, S., Lewis, M., Wang, J., Balakirsky, S. & Scrapper, C. (2007). USARSim: a robot simulator for research and education. In *Robotics and Automation, 2007 IEEE International Conference on*, 1400–1405, IEEE. 52

Celiberto, L., Matsuura, J.P., Lopez de Mantaras, R. & Bianchi, R.A. (2012). Reinforcement learning with case-based heuristics for robocup soccer keepaway. In *Robotics Symposium and Latin American Robotics Symposium (SBR-LARS), 2012 Brazilian*, 7–13, IEEE. 46

Chen, B., Xiao, C. & Song, X. (2012). Intelligent decision system based on SOM-LVQ neural network for soccer robot. In *Intelligent Human-Machine Systems and*

Cybernetics (IHMSC), 2012 4th International Conference on, vol. 1, 323–326, IEEE. 48

Chuengsatiansup, C., Charoensripongsa, T., Wongsuphasawat, K., Rattana, K., Doungsodsri, P., Buathong, A., Wejwittayaklung, K., Wongsaisuwan, M. & Wannasuphoprasit, W. (2009). Plasma-z extended team description paper. In *Proceedings of the 15th International RoboCup Symposium, Graz, Austria*. 38, 39

D'Andrea, R. (2005). The Cornell RoboCup Robot Soccer Team: 1999–2003. In *Handbook of Networked and Embedded Control Systems*, 793–804, Springer. 39

Duan, Y., Liu, Q. & Xu, X. (2007). Application of reinforcement learning in robot soccer. *Engineering Applications of Artificial Intelligence*, **20**, 936–950. 44, 72

Duan, Y., Cui, B.X. & Xu, X.H. (2012). A multi-agent reinforcement learning approach to robot soccer. *Artificial Intelligence Review*, **38**, 193–211. 44

Farahnakian, F. & Mozayani, N. (2009). Reinforcement Learning for Soccer Multi-agents System. In *Computational Intelligence and Security, 2009. CIS'09. International Conference on*, vol. 2, 50–52, IEEE. 41

Gabel, T. & Riedmiller, M.A. (2006). Learning a partial behavior for a competitive robotic soccer agent. *KI*, **20**, 18–23. 41

Gabel, T., Riedmiller, M. & Trost, F. (2009). A case study on improving defense behavior in soccer simulation 2D: The NeuroHassle approach. In *RoboCup 2008: Robot Soccer World Cup XII*, 61–72, Springer. 42

Gerkey, B., Vaughan, R.T. & Howard, A. (2003). The player/stage project: Tools for multi-robot and distributed sensor systems. In *Proceedings of the 11th international conference on advanced robotics*, vol. 1, 317–323. 52

Goto, R., Asakura, T., Fujii, N., Matsuoka, K., Mizuno, M., Sano, T., Onuma, Y., Nagata, H., Watanabe, M. & Sugiura, T. (2013). KIKS 2013 Extended Team Description. http://robocupssl.cpe.ku.ac.th/_media/robocup2013:etdp:kiks_etdp_2013.pdf, accessed on 2014-11-17. 98

Hausknecht, M. & Stone, P. (2011). Learning powerful kicks on the aibo ers-7: The quest for a striker. In *RoboCup 2010: Robot Soccer World Cup XIV*, 254–265, Springer. 41

Haykin, S. (2009). *Neural Networks and Learning Machines*. No. v. 10 in Neural networks and learning machines, Prentice Hall. 24, 26, 28, 29, 30

Huang, Z., Yang, Y. & Chen, X. (2003). An approach to plan recognition and retrieval for multi-agent systems. In *Workshop on Adaptability in Multi-Agent Systems, First RoboCup Australian Open*. 47

Inagaki, T., Ishikawa, A., Murakami, K. & Naruse, T. (2012). Robust algorithm for safety region computation and its application to defense strategy for RoboCup SSL. In *RoboCup 2011: Robot Soccer World Cup XV*, 484–494, Springer. 39

Ishikawa, A., Yasui, K., Inagaki, T., Sawaguchi, H., Yasui, T., Murakami, K. & Naruse, T. (2011). *RoboDragons 2011 Extended Team Description*. RoboCup 2011. 38

Jolly, K., Ravindran, K., Vijayakumar, R. & Sreerama Kumar, R. (2007). Intelligent decision making in multi-agent robot soccer system through compounded artificial neural networks. *Robotics and Autonomous Systems*, **55**, 589–596. 47

Kalman, R.E. (1960). A new approach to linear filtering and prediction problems. *Journal of Fluids Engineering*, **82**, 35–45. 45

Kalyanakrishnan, S., Liu, Y. & Stone, P. (2007). Half field offense in RoboCup soccer: A multiagent reinforcement learning case study. In *RoboCup 2006: Robot Soccer World Cup X*, 72–85, Springer. 43

Kaufmann, U., Mayer, G., Kraetzschmar, G. & Palm, G. (2005). Visual robot detection in RoboCup using neural networks. In *RoboCup 2004: Robot Soccer World Cup VIII*, 262–273, Springer. 41

Kitano, H. & Asada, M. (1998). The RoboCup humanoid challenge as the millennium challenge for advanced robotics. *Advanced Robotics*, **13**, 723–736. 2

Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., Osawa, E. & Matsubara, H. (1997). RoboCup: A challenge problem for AI. *AI magazine*, **18**, 73. 1

Kleiner, A., Dietl, M. & Nebel, B. (2003). Towards a life-long learning soccer agent. In *RoboCup 2002: Robot Soccer World Cup VI*, 126–134, Springer. 45

Koenig, N. & Howard, A. (2004). Design and use paradigms for Gazebo, an open-source multi-robot simulator. In *Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, vol. 3, 2149–2154, IEEE. 52

Kok, J., De Boer, R., Vlassis, N. & Groen, F.C. (2003). Towards an optimal scoring policy for simulated soccer agents. In *RoboCup 2002: Robot Soccer World Cup VI*, 296–303, Springer. 41

Kolodner, J.L. (1992). An introduction to case-based reasoning. *Artificial Intelligence Review*, **6**, 3–34. 45

Kuo, J.Y., Huang, F.C., Ma, S.P. & Fanjiang, Y.Y. (2013). Applying hybrid learning approach to RoboCup's strategy. *Journal of Systems and Software*, **86**, 1933–1944. 47

Lattner, A.D., Miene, A., Visser, U. & Herzog, O. (2006). Sequential pattern mining for situation and behavior prediction in simulated robotic soccer. In *RoboCup 2005: Robot Soccer World Cup IX*, 118–129, Springer. 47

Laue, T., Spiess, K. & Röfer, T. (2006). SimRobot – a general physical robot simulator and its application in RoboCup. In *RoboCup 2005: Robot Soccer World Cup IX*, 173–183, Springer. 52

Lauer, M. & Riedmiller, M. (2000). An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In *In Proceedings of the Seventeenth International Conference on Machine Learning*, Citeseer. 42

Laws of the RoboCup Small Size League (2013). http://robocupssl.cpe.ku.ac.th/_media/rules:ssl-rules-2013-2.pdf, accessed on 2014-10-29. 56, 58, 60, 101

Lee, J., Ji, D., Lee, W., Kang, G. & Joo, M.G. (2005). A tactics for robot soccer with fuzzy logic mediator. In *Computational Intelligence and Security*, 127–132, Springer. 49

Li, B., Hu, H. & Spacek, L. (2003). An adaptive color segmentation algorithm for Sony legged robots. In *Applied Informatics*, 126–131, Citeseer. 41

Littman, M.L. (1994). Markov games as a framework for multi-agent reinforcement learning. In *ICML*, vol. 94, 157–163. 47

Liu, Q., Jiachen, M. & Wei, X. (2012). Action selection in cooperative robot soccer using Q-learning with Kalman filter. *Journal of Computational Information Systems*, **8**, 10367–10374. 44

Maeno, J., Ishikawa, A., Murakami, K. & Naruse, T. (2010). Safety region: an index for evaluating the situation of RoboCup Soccer game. *JSAI Technical Report, SIG-Challenge-B001-2*. 39

Marling, C., Tomko, M., Gillen, M., Alexander, D. & Chelberg, D. (2003). Case-based reasoning for planning and world modeling in the RoboCup Small Size League. In *IJCAI workshop on issues in designing physical agents for dynamic real-time environments*. 45

Marsland, S. (2011). *Machine Learning: an Algorithmic Perspective*. CRC Press. 10, 24, 26, 31

McCulloch, W.S. & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, **5**, 115–133. 24

Meriçli, Ç., Veloso, M. & Akin, H.L. (2011). Task refinement for autonomous robots using complementary corrective human feedback. *International Journal of Advanced Robotic Systems*, **8**. 7

Merke, A. & Riedmiller, M. (2002). Karlsruhe brainstormers – A reinforcement learning approach to robotic soccer. In *RoboCup 2001: Robot Soccer World Cup V*, 435–440, Springer. 42

MFC Document/View Architecture (2014). http://msdn.microsoft.com/en-us/library/4x1xy43a.aspx, accessed on 2014-10-29. 57

Michel, O. (2004). WebotsTM: Professional mobile robot simulation. *International Journal of Advanced Robotic Systems*, **1(1)**, 39–42. 52

Miene, A., Visser, U. & Herzog, O. (2004). Recognition and prediction of motion situations based on a qualitative motion description. In *RoboCup 2003: Robot Soccer World Cup VII*, 77–88, Springer. 47

Mitchell, T.M. (1997). *Machine Learning*. WCB/McGraw-Hill. 10

Monajjemi, V., Koochakzadeh, A. & Ghidary, S.S. (2012). grSim – RoboCup Small Size robot soccer simulator. In *RoboCup 2011: Robot Soccer World Cup XV*, 450–460, Springer. 52

Nakanishi, R., Murakami, K. & Naruse, T. (2008). Dynamic positioning method based on dominant region diagram to realize successful cooperative play. In *RoboCup 2007: Robot Soccer World Cup XI*, 488–495, Springer. 39

Nakanishi, R., Maeno, J., Murakami, K. & Naruse, T. (2010). An approximate computation of the dominant region diagram for the real-time analysis of group behaviors. In *RoboCup 2009: Robot Soccer World Cup XIII*, 228–239, Springer. 39

Nakashima, T., Takatani, M., Udo, M. & Ishibuchi, H. (2004). An evolutionary approach for strategy learning in RoboCup soccer. In *Systems, Man and Cybernetics, 2004 IEEE International Conference on*, vol. 2, 2023–2028, IEEE. 48

Nelson, B.L., Carson, J.S. & Banks, J. (2001). *Discrete Event System Simulation*. Prentice hall. 51

Neri, J.R.F., Zatelli, M.R., Farias dos Santos, C. & Fabro, J.A. (2012). A proposal of Q-learning to control the attack of a 2D robot soccer simulation team. In *Robotics Symposium and Latin American Robotics Symposium (SBR-LARS), 2012 Brazilian*, 174–178, IEEE. 43

Niknejad, M.R., Neyshabouri, S.A.S., GhaziMirSaeed, S.A., Kamali, E., Fazeli, M.H., Piran, Y. & Khouzani, M.T. (2011). *Immortals 2011 Team Description Paper*. RoboCup 2011. 38

Noda, I., Matsubara, H., Hiraki, K. & Frank, I. (1998). Soccer server: A tool for research on multiagent systems. *Applied Artificial Intelligence*, **12**, 233–250. 40, 52

Ogino, M., Katoh, Y., Aono, M., Asada, M. & Hosoda, K. (2004). Reinforcement learning of humanoid rhythmic walking parameters based on visual information. *Advanced Robotics*, **18**, 677–697. 41

Oubbati, M., Schanz, M. & Levi, P. (2005). Kinematic and dynamic adaptive control of a nonholonomic mobile robot using a rnn. In *Computational Intelligence in Robotics and Automation, 2005. CIRA 2005. Proceedings. 2005 IEEE International Symposium on*, 27–33, IEEE. 41

Panyapiang, T., Thanomyart, R. & Sukvichai, K. (2013). *Skuba Extended Team Description 2013*. RoboCup 2013. 38

Park, J.H., Stonier, D., Kim, J.H., Ahn, B.H. & Jeon, M.G. (2007). Recombinant rule selection in evolutionary algorithm for fuzzy path planner of robot soccer. In *KI 2006: Advances in Artificial Intelligence*, 317–330, Springer. 48

Poole, D., Mackworth, A. & Goebel, R. (1998). *Computational Intelligence*. Oxford University Press, Oxford. 1

Poudeh, A.G., Dastjerdi, S.A., Esmaeelpourfard, S., Mohammadi, H.B. & Adhami-Mirhosseini, A. (2013). *MRL Extended Team Description 2013*. RoboCup 2013. 38

Rabiee, A. & Ghasem-Aghaee, N. (2004). A scoring policy for simulated soccer agents using Reinforcement Learning. In *2nd International Conference on Autonomous Robots and Agents, New Zealand*, Citeseer. 41

Riedmiller, M. & Gabel, T. (2007). On experiences in a complex and competitive gaming domain: Reinforcement learning meets robocup. In *Computational Intelligence and Games, 2007. CIG 2007. IEEE Symposium on*, 17–23, IEEE. 41, 42

Riedmiller, M., Merke, A., Meier, D., Hoffmann, A., Sinner, A., Thate, O. & Ehrmann, R. (2001). Karlsruhe Brainstormers – a reinforcement learning way to

robotic soccer. In *RoboCup 2000: Robot Soccer World Cup IV*, 367–372, Springer. 41, 42

Riedmiller, M., Gabel, T., Hafner, R. & Lange, S. (2009). Reinforcement learning for robot soccer. *Autonomous Robots*, **27**, 55–73. 7, 45, 76, 79

RoboCup Simulation League Webpage (2013). http://wiki.robocup.org/wiki/Soccer_Simulation_League, accessed on 2014-10-29. 50

RoboCup SSL webpage (2014). http://robocupssl.cpe.ku.ac.th/, accessed on 2014-10-29. 2, 3, 54

RoboCup Standard Platform League webpage (2014). http://www.robocup.org/robocup-soccer/standard-platform/, accessed on 2014-10-29. 46

RoboCup webpage (2014). http://www.robocup.org/, accessed on 2014-10-21. 1

Ros, R., Arcos, J.L., Lopez de Mantaras, R. & Veloso, M. (2009). A case-based approach for coordinated action selection in robot soccer. *Artificial Intelligence*, **173**, 1014–1039. 46

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, **65**, 386. 26

Rosenblatt, F. (1962). *Principles of Neurodynamics*. Spartan Book. 26

Rumelhart, D.E., Hintont, G.E. & Williams, R.J. (1986). Learning representations by back-propagating errors. *Nature*, **323**, 533–536. 29

Sharbafi, M.A., Azidehak, A., Hoshyari, M., Bakhshandeh, O., Babarsad, A.A.M., Zareian, A., Esmaeely, D., Ganjali, A., Esmaeelpourfard, S., Ziadloo, S. *et al.* (2011). *MRL Extended Team Description 2011*. RoboCup 2011. 98

Simon, P. (2013). *Too Big to ignore: The Business Case for Big Data*. Wiley.com. 10

Smit, D.G.H. (2014). *Robocup Small Size League: active ball handling system*. Master's thesis, Stellenbosch University. 3

Smith, R. (2014a). ODE – Open Dynamics Engine. http://www.ode.org/, accessed on 2014-10-29. 55

Smith, R. (2014b). Open Dynamics Engine User Guide. http://ode.org/ode-latest-userguide.html, accessed on 2014-10-29. 55

Sng, H., Gupta, G.S. & Messom, C.H. (2002). Strategy for collaboration in robot soccer. In *Electronic Design, Test and Applications, 2002. Proceedings. The First IEEE International Workshop on*, 347–351, IEEE. 49

Sokolowski, J.A. & Banks, C.M. (2011). *Principles of Modeling and Simulation: A Multidisciplinary Approach*. John Wiley & Sons. 51

Srisabye, J., Wasuntapichaikul, P., Onman, C., Sukvichai, K., Damyot, S., Munintarawong, T., Phuangjaisri, P. & Tipsuwan, Y. (2009). Skuba 2009 extended team description. In *Proceedings of the international RoboCup symposium*. 53

Stone, P., Sutton, R.S. & Kuhlmann, G. (2005). Reinforcement learning for RoboCup soccer keepaway. *Adaptive Behavior*, **13**, 165–188. 42

Sutton, R.S. & Barto, A.G. (1998). *Reinforcement Learning: An Introduction*, vol. 1. Cambridge University Press. 11, 13, 14, 15, 16, 17, 18, 19, 22, 24

Taylor, M.E. & Stone, P. (2009). Transfer learning for reinforcement learning domains: A survey. *The Journal of Machine Learning Research*, **10**, 1633–1685. 46

Tews, A. & Wyeth, G. (2000). MAPS: A system for multi-agent coordination. *Advanced Robotics*, **14**, 37–50. 39

Treptow, A. & Zell, A. (2004). Real-time object tracking for soccer-robots without color information. *Robotics and Autonomous Systems*, **48**, 41–48. 41

Veloso, M. & Stone, P. (1998). Individual and collaborative behaviors in a team of homogeneous robotic soccer agents. In *Multi Agent Systems, 1998. Proceedings. International Conference on*, 309–316, IEEE. 39

Veloso, M., Bowling, M. & Stone, P. (1998). The CMUnited-98 champion small-robot team. *Advanced Robotics*, **13**, 753–766. 39

Visser, U. & Burkhard, H.D. (2007). RoboCup: 10 years of achievements and future challenges. *AI magazine*, **28**, 115. 2

Watkins, C.J. & Dayan, P. (1992). Q-learning. *Machine Learning*, **8**, 279–292. 18

Watkins, C.J.C.H. (1989). *Learning from Delayed Rewards*. Ph.D. thesis, University of Cambridge. 18

Wu, C.J. & Lee, T.L. (2004). A fuzzy mechanism for action selection of soccer robots. *Journal of Intelligent and Robotic Systems*, **39**, 57–70. 49

Wu, J., Snašel, V., Ochodkova, E., Martinovič, J., Svatoň, V. & Abraham, A. (2013a). Analysis of strategy in robot soccer game. *Neurocomputing*, **109**, 66–75. 41

Wu, Y., Zhao, Y., Shen, Y., Li, C., Fang, L., Tong, H. & Xiong, R. (2013b). *ZJUNlict Extended Team Description Report for RoboCup 2013*. RoboCup 2013. 38

Zagal, J.C. & Ruiz-del Solar, J. (2005). UCHILSIM: A dynamically and visually realistic simulator for the RoboCup four legged league. In *RoboCup 2004: Robot Soccer World Cup VIII*, 34–45, Springer. 52

Zickler, S., Biswas, J., Luo, K. & Veloso, M. (2010). CMDragons 2010 team description. 39

Appendix A

The maximum number of learning episodes required

This appendix introduces and contains a simple code to determine the maximum number of learning episodes. As mentioned in Section 2.3.2, the learning rate η in the back-propagation algorithms is gradually decreased in time for convergence. In this research the learning rate η was decreased by multiplying a constant value ρ after each learning episode, as discussed in Section 5.9. Therefore, η_n, the learning rate after the n^{th} learning episode, can be defined as follows:

$$\eta_n = \eta_0 \times \rho^{n-1}, \tag{A.1}$$

where η_0 denotes the initial value assigned to η, and ρ denotes the decreasing factor. The learning rate η_n converges to zero as $n \to \infty$.

Algorithm 9 can be used to find the value n that makes the learning rate η_n effectively 0. The algorithm stops when η_n becomes effectively 0 and returns the value n, which means that the learning has no effect after the n^{th} iteration.

Algorithm 9 The algorithm to determine the maximum number of learning episodes.

1: $n \leftarrow 0$, $\eta \leftarrow \eta_0$
2: **while** $((1 - \eta) \neq 1)$
3: $\quad \eta \leftarrow \rho \times \eta$
4: $\quad n \leftarrow n + 1$
5: **end while**
6: return n

Appendix B

The learned MLPs

This appendix presents the weights of the learned Multi-Layer Perceptrons (MLPs) in the seven Reinforcement Learning (RL) experiments performed in this research. Each section provides one or more tables that contain the weights of the learned MLP from each experiment.

As mentioned in Section 5.7, all MLPs used in this research have one hidden layer, thus they have two layers of weights. In the tables presented in this appendix, the weights in the first layer of the MLP are referred to as v_{ij} (the weight connecting the i^{th} input and the j^{th} neuron in the hidden layer) and the weights in the second layer of the MLP are denoted by w_{jk} (the weight connecting the j^{th} neuron in the hidden layer and the k^{th} neuron in the output layer). The structure of the MLPs used in the experiments dealing with the stationary ball was 3–7–1, while it was set to 7–21–1 in the experiments handling the moving ball (refer to Table 5.4).

B.1 The learned MLP in Experiment 1

Table B.1: The weights of the learned MLP in Experiment 1.

Weights in the first layer (v_{ij})

	j						
i	1	2	3	4	5	6	7
1	-0.005070	0.005765	-0.001800	0.010540	0.006533	0.004622	0.002108
2	0.008603	0.007017	0.005447	-0.005997	0.007839	0.004928	0.001007
3	0.002903	-0.003376	-0.002040	0.003461	-0.000315	0.000289	0.016840

Weights in the second layer (w_{jk})

	j						
k	1	2	3	4	5	6	7
1	0.118001	0.111681	0.108995	0.110080	0.117130	0.120058	0.121895

B.2 The learned MLP in Experiment 2

Table B.2: The weights of the learned MLP in Experiment 2.

Weights in the first layer (v_{ij})

	j										
i	1	2	3	4	5	6	7	8	9	10	11
1	-0.009536	0.001843	-0.005619	0.006584	0.002282	0.000202	-0.002523	0.008426	0.007078	0.005286	-0.005869
2	-0.000301	-0.006595	-0.008984	-0.009090	-0.001400	0.001717	0.002267	0.002944	0.003257	-0.006043	0.004424
3	0.008265	0.007451	-0.000186	-0.007960	-0.001840	-0.006411	0.006340	-0.006773	0.004449	0.006155	0.008749
4	-0.009215	-0.010241	0.007993	-0.004804	-0.004965	0.001322	0.003465	0.006372	0.004083	-0.000590	-0.006354
5	0.002714	-0.001143	-0.005494	-0.004313	0.000163	0.007780	0.001697	0.011256	0.006800	-0.002056	-0.004777
6	-0.003799	-0.001190	-0.008370	0.009442	0.003951	-0.006625	0.007718	0.006641	0.001969	-0.006118	-0.006083
7	0.004953	0.000176	-0.006935	-0.002163	0.007377	-0.008588	0.000893	0.003855	-0.000746	-0.007491	0.009748
i	12	13	14	15	16	17	18	19	20	21	
1	0.007704	0.004575	0.000777	-0.003510	-0.009094	-0.007633	-0.002159	-0.006646	-0.006073	0.010162	
2	-0.000043	-0.002305	-0.007953	0.002887	0.006753	0.007017	0.001386	-0.003233	0.008611	0.005234	
3	0.009002	0.001493	-0.003110	-0.005484	-0.005175	0.005791	-0.010559	-0.003286	-0.009273	0.002764	
4	0.004484	-0.000927	-0.001219	0.008660	0.004451	-0.008232	0.001575	-0.002616	0.004261	0.001867	
5	0.004652	0.000902	-0.007278	0.005183	0.001824	-0.005510	0.010568	-0.005989	0.009837	0.004996	
6	0.006573	-0.000431	-0.006677	0.000187	0.004953	-0.001644	-0.004154	0.001483	0.003958	0.005206	
7	0.009044	0.001420	-0.002486	-0.000084	-0.001789	0.007572	-0.003015	-0.000400	-0.003847	0.010119	

Weights in the second layer (w_{jk})

	j										
k	1	2	3	4	5	6	7	8	9	10	11
1	0.068119	0.076752	0.072746	0.066232	0.077781	0.079315	0.070288	0.073255	0.080421	0.062311	0.081909
k	12	13	14	15	16	17	18	19	20	21	
1	0.074047	0.063347	0.072336	0.066475	0.079380	0.074700	0.075993	0.065933	0.079840	0.065448	

B.3 The learned MLP in Experiment 3

Table B.3: The weights of the learned MLP in Experiment 3.

Weights in the first layer (v_{ij})

	j						
i	1	2	3	4	5	6	7
1	-0.007323	0.003636	-0.003878	0.008446	0.004300	0.002332	-0.000214
2	0.008407	0.006856	0.005299	-0.006148	0.007649	0.004722	0.000796
3	0.000624	-0.005517	-0.004128	0.001359	-0.002573	-0.002032	0.014484

Weights in the second layer (w_{jk})

	j						
k	1	2	3	4	5	6	7
	0.122645	0.116254	0.113693	0.114482	0.121622	0.124592	0.126143

B.4 The learned MLPs in Experiment 4

Table B.4: The weights of the learned MLP in Experiment 4 with a time delay of 50 ms.

Weights in the first layer (v_{ij})

	j						
i	1	2	3	4	5	6	7
1	-0.006442	0.004478	-0.003052	0.009279	0.005174	0.003225	0.000689
2	0.009461	0.007876	0.006304	-0.005136	0.008699	0.005788	0.001871
3	0.001319	-0.004860	-0.003487	0.002006	-0.001884	-0.001325	0.015201

Weights in the second layer (w_{jk})

	j						
k	1	2	3	4	5	6	7
	0.098702	0.092332	0.089725	0.090560	0.097726	0.100672	0.102286

Table B.5: The weights of the learned MLP in Experiment 4 with a time delay of 100 ms.

Weights in the first layer (v_{ij})

	j						
i	1	2	3	4	5	6	7
1	-0.006407	0.004512	-0.003019	0.009312	0.005209	0.003260	0.000725
2	0.009392	0.007811	0.006241	-0.005200	0.008631	0.005718	0.001800
3	0.001220	-0.004954	-0.003578	0.001914	-0.001982	-0.001426	0.015099

Weights in the second layer (w_{jk})

	j						
k	1	2	3	4	5	6	7
	0.100239	0.093877	0.091268	0.092107	0.099268	0.102214	0.103816

B.5 The learned MLPs in Experiment 5

Table B.6: The weights of the learned MLP in Experiment 5 with a time delay of 50 ms.

Weights in the first layer (v_{ij})

	j						
i	1	2	3	4	5	6	7
1	-0.007432	0.003579	-0.003914	0.008408	0.004198	0.002205	-0.000346
2	0.010162	0.008550	0.006969	-0.004470	0.009394	0.006494	0.002585
3	0.001326	-0.004867	-0.003491	0.001995	-0.001885	-0.001322	0.015207

Weights in the second layer (w_{jk})

	j						
k	1	2	3	4	5	6	7
	0.126433	0.119890	0.117388	0.118004	0.125281	0.128244	0.129897

Table B.7: The weights of the learned MLP in Experiment 5 with a time delay of 100 ms.

Weights in the first layer (v_{ij})

	j						
i	1	2	3	4	5	6	7
1	-0.007769	0.003256	-0.004231	0.008089	0.003864	0.001864	-0.000690
2	0.009993	0.008385	0.006806	-0.004634	0.009226	0.006324	0.002414
3	0.000889	-0.005287	-0.003905	0.001579	-0.002320	-0.001764	0.014761

Weights in the second layer (w_{jk})

	j						
k	1	2	3	4	5	6	7
	0.101908	0.095359	0.092871	0.093452	0.100740	0.103708	0.105326

B.6 The learned MLPs in Experiment 6

Table B.8: The weights of the learned MLP in Experiment 6 (original).

Weights in the first layer (v_{ij})

	j										
i	1	2	3	4	5	6	7	8	9	10	11
1	-0.010042	0.001202	-0.006202	0.006108	0.001630	-0.000473	-0.003062	0.007850	0.006385	0.004867	-0.006590
2	-0.000031	-0.006258	-0.008677	-0.008838	-0.001056	0.002072	0.002553	0.003248	0.003621	-0.005821	0.004804
3	0.008335	0.007535	-0.000108	-0.007890	-0.001756	-0.006325	0.006413	-0.006694	0.004537	0.006220	0.008838
4	-0.009173	-0.010204	0.008032	-0.004760	-0.004928	0.001358	0.003506	0.006412	0.004119	-0.000545	-0.006321
5	0.002935	-0.000878	-0.005250	-0.004108	0.000433	0.008058	0.001928	0.011499	0.007084	-0.001872	-0.004481
6	-0.003752	-0.001110	-0.008303	0.009481	0.004034	-0.006536	0.007773	0.006704	0.002062	-0.006092	-0.005983
7	0.004930	0.000155	-0.006958	-0.002190	0.007355	-0.008609	0.000870	0.003832	-0.000767	-0.007518	0.009729
i	12	13	14	15	16	17	18	19	20	21	
1	0.007109	0.004144	0.000203	-0.003987	-0.009771	-0.008241	-0.002781	-0.007120	-0.006753	0.009704	
2	0.000271	-0.002076	-0.007651	0.003142	0.007112	0.007340	0.001714	-0.002980	0.008971	0.005478	
3	0.009081	0.001559	-0.003032	-0.005416	-0.005091	0.005870	-0.010478	-0.003218	-0.009188	0.002832	
4	0.004523	-0.000881	-0.001179	0.008704	0.004486	-0.008194	0.001614	-0.002573	0.004296	0.001912	
5	0.004900	0.001091	-0.007039	0.005392	0.002107	-0.005252	0.010829	-0.005781	0.010121	0.005196	
6	0.006641	-0.000403	-0.006612	0.000227	0.005042	-0.001572	-0.004079	0.001523	0.004047	0.005240	
7	0.009021	0.001393	-0.002510	-0.000109	-0.001808	0.007550	-0.003037	-0.000425	-0.003867	0.010093	

Weights in the second layer (w_{jk})

	j										
k	1	2	3	4	5	6	7	8	9	10	11
1	0.075132	0.083526	0.079542	0.072911	0.084603	0.086181	0.077240	0.080073	0.087235	0.069002	0.088854
k	12	13	14	15	16	17	18	19	20	21	
1	0.080840	0.070114	0.079041	0.073419	0.086443	0.081702	0.082914	0.072830	0.086906	0.072235	

Table B.9: The weights of the learned MLP in Experiment 6 (with a bigger force).

Weights in the first layer (v_{ij})

	j										
i	1	2	3	4	5	6	7	8	9	10	11
1	-0.010031	0.001215	-0.006190	0.006119	0.001642	-0.000460	-0.003051	0.007862	0.006399	0.004878	-0.006576
2	-0.000030	-0.006257	-0.008676	-0.008838	-0.001056	0.002073	0.002554	0.003249	0.003622	-0.005820	0.004804
3	0.008336	0.007536	-0.000106	-0.007889	-0.001754	-0.006323	0.006414	-0.006693	0.004538	0.006221	0.008839
4	-0.009174	-0.010204	0.008031	-0.004761	-0.004929	0.001357	0.003505	0.006411	0.004118	-0.000545	-0.006322
5	0.002931	-0.000882	-0.005254	-0.004112	0.000429	0.008054	0.001924	0.011495	0.007079	-0.001875	-0.004485
6	-0.003753	-0.001111	-0.008304	0.009480	0.004033	-0.006537	0.007772	0.006703	0.002060	-0.006093	-0.005984
7	0.004929	0.000154	-0.006959	-0.002190	0.007354	-0.008610	0.000869	0.003831	-0.000768	-0.007519	0.009728
i	12	13	14	15	16	17	18	19	20	21	
1	0.007122	0.004155	0.000215	-0.003976	-0.009758	-0.008229	-0.002768	-0.007109	-0.006740	0.009715	
2	0.000271	-0.002075	-0.007650	0.003143	0.007113	0.007340	0.001715	-0.002979	0.008972	0.005478	
3	0.009083	0.001560	-0.003031	-0.005415	-0.005089	0.005871	-0.010476	-0.003217	-0.009187	0.002833	
4	0.004523	-0.000882	-0.001180	0.008703	0.004486	-0.008195	0.001613	-0.002573	0.004295	0.001911	
5	0.004896	0.001088	-0.007043	0.005388	0.002102	-0.005257	0.010824	-0.005785	0.010117	0.005192	
6	0.006640	-0.000404	-0.006613	0.000226	0.005041	-0.001573	-0.004080	0.001522	0.004046	0.005239	
7	0.009021	0.001392	-0.002511	-0.000110	-0.001809	0.007550	-0.003038	-0.000425	-0.003868	0.010092	

Weights in the second layer (w_{jk})

	j										
k	1	2	3	4	5	6	7	8	9	10	11
1	0.075221	0.083617	0.079631	0.073002	0.084694	0.086271	0.077330	0.080164	0.087326	0.069093	0.088944
k	12	13	14	15	16	17	18	19	20	21	
1	0.080931	0.070205	0.079132	0.073508	0.086532	0.081792	0.083003	0.072919	0.086994	0.072326	

Table B.10: The weights of the learned MLP in Experiment 6 (with a lowered target).

Weights in the first layer (v_{ij})

	j										
i	1	2	3	4	5	6	7	8	9	10	11
1	-0.009569	0.001748	-0.005689	0.006564	0.002184	0.000093	-0.002571	0.008364	0.006962	0.005290	-0.005999
2	-0.000261	-0.006522	-0.008926	-0.009060	-0.001324	0.001798	0.002314	0.002999	0.003342	-0.006026	0.004518
3	0.008323	0.007519	-0.000122	-0.007902	-0.001772	-0.006341	0.006400	-0.006709	0.004519	0.006210	0.008820
4	-0.009169	-0.010198	0.008037	-0.004757	-0.004923	0.001363	0.003511	0.006417	0.004124	-0.000541	-0.006315
5	0.002833	-0.000993	-0.005358	-0.004205	0.000317	0.007940	0.001824	0.011390	0.006964	-0.001963	-0.004604
6	-0.003737	-0.001093	-0.008287	0.009495	0.004051	-0.006518	0.007788	0.006720	0.002080	-0.006080	-0.005964
7	0.004927	0.000152	-0.006961	-0.002192	0.007352	-0.008612	0.000867	0.003829	-0.000770	-0.007521	0.009726
i	12	13	14	15	16	17	18	19	20	21	
1	0.007632	0.004576	0.000711	-0.003529	-0.009203	-0.007712	-0.002243	-0.006666	-0.006182	0.010152	
2	0.000018	-0.002286	-0.007897	0.002919	0.006837	0.007083	0.001453	-0.003201	0.008694	0.005260	
3	0.009066	0.001549	-0.003046	-0.005428	-0.005107	0.005855	-0.010493	-0.003229	-0.009204	0.002821	
4	0.004528	-0.000878	-0.001174	0.008708	0.004492	-0.008189	0.001619	-0.002569	0.004302	0.001915	
5	0.004790	0.000998	-0.007146	0.005293	0.001987	-0.005365	0.010715	-0.005879	0.010001	0.005099	
6	0.006658	-0.000390	-0.006597	0.000241	0.005060	-0.001555	-0.004062	0.001537	0.004065	0.005254	
7	0.009019	0.001390	-0.002513	-0.000112	-0.001811	0.007547	-0.003040	-0.000428	-0.003870	0.010090	

Weights in the second layer (w_{jk})

	j										
k	1	2	3	4	5	6	7	8	9	10	11
1	0.071694	0.080216	0.076188	0.069667	0.081280	0.082815	0.073858	0.076772	0.083921	0.065722	0.085438
k	12	13	14	15	16	17	18	19	20	21	
1	0.077548	0.066812	0.075741	0.070024	0.082990	0.078265	0.079527	0.069449	0.083460	0.068945	

B.7 The learned MLPs in Experiment 7

Table B.11: The weights of the learned MLP in Experiment 7 (original).

Weights in the first layer (v_{ij})

	j										
i	1	2	3	4	5	6	7	8	9	10	11
1	-0.009096	0.002374	-0.005127	0.007007	0.002821	0.000754	-0.002060	0.008915	0.007644	0.005671	-0.005286
2	-0.000013	-0.006281	-0.008683	-0.008804	-0.001084	0.002035	0.002562	0.003247	0.003580	-0.005768	0.004751
3	0.008287	0.007502	-0.000147	-0.007947	-0.001787	-0.006351	0.006368	-0.006736	0.004511	0.006156	0.008818
4	-0.009219	-0.010240	0.007992	-0.004811	-0.004964	0.001325	0.003462	0.006370	0.004086	-0.000599	-0.006351
5	0.002387	-0.001591	-0.005890	-0.004610	-0.000294	0.007299	0.001341	0.010866	0.006305	-0.002302	-0.005296
6	-0.003620	-0.000988	-0.008178	0.009615	0.004156	-0.006416	0.007903	0.006833	0.002181	-0.005955	-0.005867
7	0.005056	0.000291	-0.006826	-0.002061	0.007492	-0.008473	0.000999	0.003964	-0.000628	-0.007393	0.009868
i	12	13	14	15	16	17	18	19	20	21	
1	0.008206	0.004968	0.001264	-0.003088	-0.008540	-0.007124	-0.001643	-0.006226	-0.005518	0.010574	
2	0.000264	-0.002028	-0.007651	0.003171	0.007072	0.007324	0.001694	-0.002949	0.008929	0.005517	
3	0.009041	0.001497	-0.003075	-0.005469	-0.005115	0.005834	-0.010510	-0.003272	-0.009211	0.002773	
4	0.004483	-0.000935	-0.001221	0.008655	0.004454	-0.008232	0.001576	-0.002622	0.004264	0.001860	
5	0.004246	0.000644	-0.007665	0.004882	0.001342	-0.005927	0.010136	-0.006286	0.009352	0.004714	
6	0.006768	-0.000266	-0.006486	0.000361	0.005162	-0.001448	-0.003955	0.001656	0.004167	0.005377	
7	0.009156	0.001518	-0.002377	0.000017	-0.001673	0.007683	-0.002904	-0.000299	-0.003732	0.010220	

Weights in the second layer (w_{jk})

	j										
k	1	2	3	4	5	6	7	8	9	10	11
1	0.080340	0.089127	0.085056	0.078666	0.090150	0.091507	0.082630	0.085535	0.092772	0.074711	0.094292
k	12	13	14	15	16	17	18	19	20	21	
1	0.086464	0.075718	0.084733	0.078700	0.091620	0.087079	0.088119	0.078270	0.091984	0.077871	

Table B.12: The weights of the learned MLP in Experiment 7 (with a bigger force).

Weights in the first layer (v_{ij})

	j										
i	1	2	3	4	5	6	7	8	9	10	11
1	-0.009092	0.002378	-0.005123	0.007011	0.002825	0.000759	-0.002056	0.008919	0.007648	0.005675	-0.005281
2	-0.000024	-0.006293	-0.008694	-0.008814	-0.001096	0.002022	0.002551	0.003235	0.003568	-0.005778	0.004739
3	0.008290	0.007505	-0.000143	-0.007943	-0.001783	-0.006347	0.006372	-0.006732	0.004515	0.006159	0.008822
4	-0.009215	-0.010235	0.007996	-0.004807	-0.004959	0.001329	0.003467	0.006375	0.004091	-0.000595	-0.006346
5	0.002378	-0.001600	-0.005899	-0.004618	-0.000304	0.007290	0.001332	0.010857	0.006295	-0.002310	-0.005306
6	-0.003624	-0.000992	-0.008182	0.009611	0.004151	-0.006421	0.007899	0.006829	0.002176	-0.005959	-0.005871
7	0.005049	0.000283	-0.006833	-0.002068	0.007484	-0.008481	0.000992	0.003957	-0.000636	-0.007400	0.009860
i	12	13	14	15	16	17	18	19	20	21	
1	0.008210	0.004971	0.001268	-0.003085	-0.008536	-0.007120	-0.001639	-0.006222	-0.005513	0.010578	
2	0.000252	-0.002038	-0.007662	0.003160	0.007059	0.007313	0.001682	-0.002959	0.008917	0.005507	
3	0.009045	0.001500	-0.003071	-0.005465	-0.005111	0.005838	-0.010507	-0.003268	-0.009207	0.002776	
4	0.004487	-0.000931	-0.001216	0.008659	0.004459	-0.008228	0.001581	-0.002618	0.004269	0.001864	
5	0.004237	0.000636	-0.007674	0.004874	0.001332	-0.005937	0.010127	-0.006294	0.009342	0.004706	
6	0.006763	-0.000269	-0.006490	0.000357	0.005157	-0.001452	-0.003959	0.001652	0.004163	0.005373	
7	0.009148	0.001511	-0.002384	0.000010	-0.001681	0.007676	-0.002912	-0.000305	-0.003740	0.010213	

Weights in the second layer (w_{jk})

	j										
k	1	2	3	4	5	6	7	8	9	10	11
1	0.080370	0.089159	0.085090	0.078699	0.090180	0.091537	0.082661	0.085565	0.092803	0.074744	0.094323
k	12	13	14	15	16	17	18	19	20	21	
1	0.086494	0.075749	0.084766	0.078730	0.091649	0.087108	0.088148	0.078301	0.092013	0.077900	

Table B.13: The weights of the learned MLP in Experiment 7 (with a lowered target).

Weights in the first layer (v_{ij})

	j										
i	1	2	3	4	5	6	7	8	9	10	11
1	-0.009148	0.002315	-0.005183	0.006957	0.002761	0.000694	-0.002114	0.008859	0.007582	0.005624	-0.005349
2	0.000043	-0.006219	-0.008623	-0.008749	-0.001020	0.002099	0.002620	0.003307	0.003645	-0.005716	0.004818
3	0.008299	0.007516	-0.000134	-0.007935	-0.001773	-0.006336	0.006381	-0.006723	0.004525	0.006167	0.008832
4	-0.009193	-0.010212	0.008019	-0.004785	-0.004935	0.001354	0.003489	0.006398	0.004116	-0.000575	-0.006321
5	0.002359	-0.001622	-0.005920	-0.004637	-0.000326	0.007268	0.001313	0.010836	0.006273	-0.002328	-0.005329
6	-0.003596	-0.000959	-0.008152	0.009639	0.004185	-0.006387	0.007928	0.006859	0.002211	-0.005933	-0.005836
7	0.005030	0.000261	-0.006854	-0.002086	0.007461	-0.008504	0.000972	0.003936	-0.000660	-0.007417	0.009836
i	12	13	14	15	16	17	18	19	20	21	
1	0.008149	0.004919	0.001209	-0.003139	-0.008601	-0.007181	-0.001701	-0.006276	-0.005579	0.010524	
2	0.000324	-0.001975	-0.007592	0.003226	0.007136	0.007385	0.001756	-0.002894	0.008994	0.005572	
3	0.009055	0.001509	-0.003062	-0.005457	-0.005100	0.005848	-0.010497	-0.003260	-0.009197	0.002785	
4	0.004510	-0.000911	-0.001194	0.008680	0.004483	-0.008205	0.001604	-0.002597	0.004293	0.001885	
5	0.004216	0.000618	-0.007695	0.004855	0.001310	-0.005957	0.010106	-0.006313	0.009320	0.004687	
6	0.006795	-0.000243	-0.006460	0.000385	0.005191	-0.001420	-0.003927	0.001680	0.004197	0.005400	
7	0.009127	0.001494	-0.002405	-0.000009	-0.001704	0.007654	-0.002934	-0.000324	-0.003763	0.010195	

Weights in the second layer (w_{jk})

	j										
k	1	2	3	4	5	6	7	8	9	10	11
1	0.080989	0.089766	0.085704	0.079305	0.090791	0.092152	0.083286	0.086176	0.093417	0.075352	0.094942
k	12	13	14	15	16	17	18	19	20	21	
1	0.087105	0.076359	0.085374	0.079352	0.092281	0.087734	0.088762	0.078922	0.092642	0.078511	