Hypernetworks Analysis of RoboCup Interactions

Thesis

Hypernetworks Analysis of RoboCup Interactions

Ruggero Rossi

A thesis submitted in partial fulfilment for the degree of Doctor of Philosophy

in the

Faculty of Science, Technology, Engineering and Mathematics

The Open University

22nd December 2021

Supervisors: Professor Jeffrey Johnson and Dr Anthony Lucas-Smith

Dedicated to my mother Gabriella,
who taught me how to never give up

Acknowledgements

I would like to thank my supervisor, professor Jeffrey H. Johnson whose assistance was invaluable in formulating the research questions and methodology. Discussing with you about any topic has been the best way to broaden my perspective and see things that were hidden to my eye.

I would like to also thank my co-supervisor Dr Anthony Lucas-Smith, whose experience helped me understanding how to structure my research and whose feedbacks helped me in focusing on the right path.

A grateful thank you is also due to my partner Stela, who is my greatest support and my stronger motivation.

Last, but not least, I would like to thank my brother Emanuele and my father Ovidio for always believing in me and being always there for me.

Contents

Abstract

Robotic soccer simulations are controlled environments in which the rich variety of interactions among agents make them good candidates to be studied as complex adaptive systems. The challenge is to create an autonomous team of soccer agents that can adapt and improve its behaviour as it plays other teams. By analogy with chess, the movements of the soccer agents and the ball form ever-changing networks as players in one team form structures that give their team an advantage. For example, the *Defender's Dilemma* involves relationships between an attacker with the ball, a team-mate and a defender. The defender must choose between tackling the player with the ball, or taking a position to intercept a pass to the other attacker. Since these structures involve more that two interacting entities it is necessary to go beyond networks to multidimensional hypernetworks. In this context, this thesis investigates (i) is it possible to identify patterns of play, that lead a team to obtain an advantage ?, (ii) is it possible to forecast with a good degree of accuracy if a certain game action or sequence of game actions is going to be successful, before it has been completed ?, and (iii) is it possible to make behavioural patterns emerge in the game without specifying the behavioural rules in detail ? To investigate these research questions we devised two methods to analyse the interactions between robotic players, one based on traditional programming and one based on Deep Learning. The first method identified thousands of Defender's Dilemma configurations from RoboCup 2D simulator games and found a statistically significant association between winning and the creation of the defender's dilemma by the attackers of the winning team. The second method showed that a feedforward Artificial Neural Network trained on thousands of games can take as input the current game configuration and forecast to a high degree of accuracy if the current action will end up in a goal or not. Finally, we designed our own fast and simple robotic soccer simulator for investigating Reinforcement Learning. This showed that Reinforcement Learning using Proximal Policy Optimization could train two agents in the task of scoring a goal, using only basic actions without using pre-built hand-programmed skills. These experiments provide evidence that it is possible: to identify advantageous patterns of play; to forecast if an action or sequence of actions will be successful; and to make behavioural patterns emerge in the game without specifying the behavioural rules in detail.

Chapter 1. Introduction

1.1 RoboCup Soccer and Complex Systems

RoboCup is an international robotic competition held yearly since 1997 with the aim of promoting robotics and artificial intelligence research. As the RoboCup website reports (web reference [23]), the first mention of the idea of robots playing soccer is probably by Professor Alan Mackworth in his paper “On Seeing Robots” presented at VI-92, later published in [Mackworth 1993]. Independently, after a workshop in 1992 and a series of investigation, a group of Japanese researchers including Minoru Asada, Yasuo Kuniyoshi, Hiroaki Kitano, Itsuki Noda, Eiichi Osawa, decided to develop a prototype of a robotic soccer simulator and in 1993 launched a competition, then called J-League. The international reaction was so intense that soon the initiative became an international joint project and was renamed RoboCup [Kitano, et al. 1995]. The official goal of the project, as stated in the website, is:

> "*By the middle of the 21st Century, a team of fully autonomous humanoid robot soccer players shall win a soccer game, complying with the official rules of [football governing body] FIFA, against the winner of the most recent World Cup."*

During the years other robotic soccer competitions were organized from different organizations, such as FIRA WorldCup (formerly MIROSOT, web reference [24]). Meanwhile RoboCup extended the types of challenges, and nowadays it includes many competition domains, some of which are about making teams of robots playing soccer matches, while others are dedicated to different tasks such as rescue.

RoboCupSoccer	RoboCupRescue	RoboCup@Home	RoboCupIndustrial	RoboCupJunior
Humanoid	Robot	Open Platform	RoboCup@Work	Soccer
Standard platform	Simulation	Domestic Standard Platform	Logistics	OnStage
Middle Size		Social Standard Platform		Rescue
Small Size				
Simulation				

Table 1.1 : The RoboCup leagues organized by category

Among the competitions related to soccer, “RoboCup Soccer Simulation 2D” uses only software components inside a simulated environment, avoiding the usage of hardware robots, that allows focusing on some aspects of the artificial intelligence without concerning with physical hardware issues [see web ref. 1 and 2]. This is illustrated in Figure 2.1.

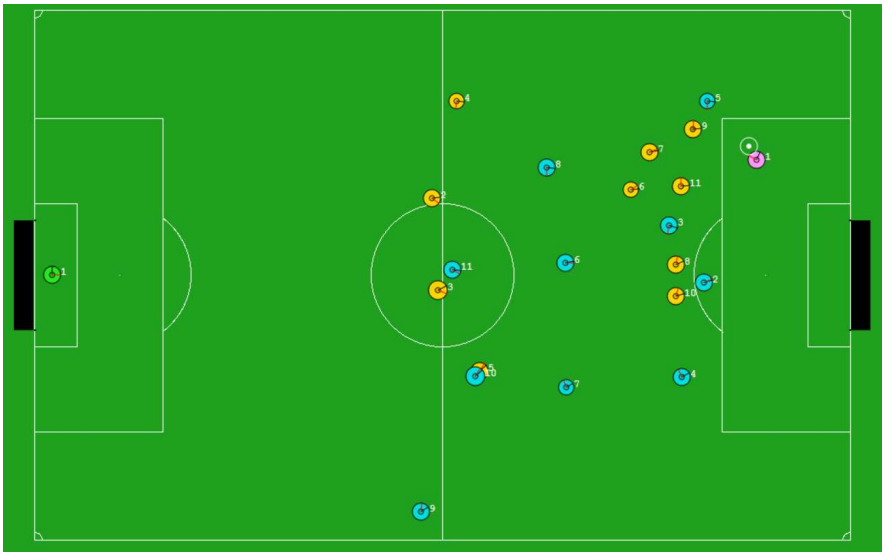


Figure 1.1 : A RoboCup 2D soccer game as it appears on the LogPlayer

A RoboCup tournament is a competition between teams of software player agents in multiagent systems. In every match of this competition two opposing teams composed of artificial intelligences play a soccer game sending commands representing their actions to a server that simulates the environment of a 2-dimensional playing field. Such a simulation mimics aspects of the real life soccer game, with a variable number (usually 11) of players for each team and similar rules, in a pitch that usually is 105 units long and 68 units wide (units may be intended as meters).

Each player is a separate intelligent software agent that obtains simulated sensors data from the game server and can communicate (with some limitation) with other players as well as with a separate coach agent, and can choose actions such as “dash”, “kick”, “turn”, “turn the neck”, “tackle”, “catch the ball” (“catch” is reserved only to goalkeepers), with some constraints given by bounded sight and limited stamina. The game server saves two log files for every match containing the information to reconstruct the game state at each instant: one log file with suffix “.rcg” contains information such as positions and velocity of the players and the ball, the other log file with suffix “.rcl” contains information about the commands sent to the server by the agents, such as the actions that they intended to do. Those log files are helpful for

subsequent analysis. It goes without saying that in such a controlled yet rich environment it is possible to actuate a broad set of tactics, and the task of recognizing them is not trivial.

Robot Soccer can be regarded as a Complex System: a system composed of interacting elements whose behaviour cannot be easily forecasted, as instead it is possible for the ideally opposed complicated systems, which are composed of interacting elements that express a well-defined function whose effects are limited to the local interaction. Typical of complex systems is the characteristic to self-organize in some hierarchy, to react and adapt to changes, and to give rise to the emergence of new structures. This may be modelled as multilevel systems, or “systems of systems of systems”. As it will be extensively illustrated later, a way to formalize multilevel systems is by using Hypernetworks.

When complex systems are sensitive to initial conditions, a small change in initial conditions can lead to large divergence between system trajectories through time. In practice this makes point predictions, that the system will be a given state at a given point in time, impossible. However, it may be possible to estimate the probability of being close to a given state in an interval of future time.
Complicated systems such as clocks and motor cars have predictable behaviour but tangled complex systems such as a football game apparently do not. This does not mean that the future behaviour of complex systems is entirely unknowable. For example, the statistics of the past games of two football teams can be an indicator of their ability and be used in probabilistic models to decrease the uncertainty about their future performances. Another approach to forecasting complex systems is to create computational models of them as they change in time. These models allow the short-term future state of the system to be calculated from the present state. By iterated calculation a trajectory into the future can be calculated for a sequence of ticks of the clock. This is an example of computer simulation. Since most complex systems are sensitive to initial conditions, a single simulation run may have little information, but many runs can give information on the ‘space of possible futures’.

1.2 Aim of the Research

The focus of this research is to find a method to analyse actions played by agents in RoboCup 2D Soccer, in order to be able to find advantageous game configurations or even recognize tactics and understand which ones are successful. The choice of using RoboCup 2D as a

study environment will allow us to have a great amount of data about collective agent interactions coming from played games, and in case of need new data may be generated by new matches. RoboCup 2D Soccer offers the opportunity to collect log data files ready to be processed, coming from an environment of great complexity and with the advantages of having equal players for each team (so there will not be the problem of having one team physically stronger or faster than the other) and uncorrupted data (since it comes directly from the simulation).

As it will be explained below, the method of analysis will utilize deep learning and hypernetworks. In addition to using RoboCup 2D as environment, we created our own robotic soccer simulator, with the characteristic of being very fast in execution, so to be used in cases where a great number of samples are needed, such as in Reinforcement Learning algorithms. Being able to analyse game configurations may result useful for many purposes: it could be employed to improve the performance of the robotic player agents, or it could be applied to real life human soccer games, since real player tracking systems are nowadays being used in major soccer leagues such as Premier League or La Liga (for instance as Tracab [3]).

1.3 Research Questions

The investigation in this thesis deals with the study of different processes to extract the peculiar characteristics of the game situation or of the teams, and will try to answer specifically three questions:

Research Question 1

Is it possible to identify patterns of play, that lead a team to obtain an advantage ?

Research Question 2

Is it possible to forecast with a good degree of accuracy if a certain game action or sequence of game actions is going to be successful, before it has been completed ?

Research Question 3

Is it possible to make behavioural patterns emerge in the game without specifying the behavioural rules in detail ?

1.4 Research Methodologies

As it will be clear in the literature review chapter, currently there are no completely satisfactory solutions to automatically analyse game configurations or tactics, both in robotic and human soccer.

The first two research questions require methodologies that are capable of recognizing patterns (first and second research questions) and make a prediction (second research question) in a multilevel complex system. This suggests the need for a conceptual framework for discussing complex system structures and interactions. As it will be detailed later, Hypernetworks are mathematical abstractions that can be employed to formalize and analyse complex systems, they can be considered as a generalization of networks in which relationships involve possibly more than 2 entities. They allow formulation of part-whole relationships, hierarchies, emergence, multilevel dynamics, adaptation. Tactics in (robotic) soccer games has a strong component of adaptation and may entail the existence (and hence the modelling) of emergent relationships, temporary hierarchies and multilevel dynamics. We think that Hypernetworks may be the language to describe all that.

For what concerns recognizing patterns and make predictions, in the last years we assisted to the successful utilization of deep learning methods, that consist in different kind of artificial neural networks architectures and in various probabilistic graphical models (in the literature review section we will see that their application specifically to robot soccer seems promising). Deep learning methods have the characteristic of being able to capture patterns, nonlinear relationships and high level concepts from data [Goodfellow et al. 2016]. For this reason they seem the best candidate for this task.

There are architectural and mathematical similarities between some deep learning models and some kind of hypernetworks (as it will be described in a later section), so an open question is if there is some sort of equivalence of one with respect to the other, or in other words “if I am using a deep learning model, am I using a representation of an equivalent hypernetwork model” ?

The third research questions is about the emergence of a behaviour without specifying the rules in detail: this seems right the case for the application of Reinforcement Learning, where an agent learns how to act without any prior behavioural rule. Since Reinforcement Learning requires a big amount of samples, we also developed a simplified robotic soccer simulator to quickly generate many simulations.

In every experiment that we conduct, the resulting data is analysed statistically, and statistical tests are run when necessary.

To avoid misunderstandings it is necessary to point out that with the term “Hypernetwork” we are referring to the meaning originated in the Complex Systems field ([Johnson 2006, 2013]) and not to the different meaning originated later in the Deep Learning community in which for “hypernetwork” it is meant a neural network that creates the weights for another neural network ([Ha et al. 2016]).

1.5 Thesis Outline

Chapter 2 reviews the literature and concludes that currently there are not satisfactory answers to our research questions. The literature review also introduces the theoretical background on which we build our methodology: Hypernetworks, Deep Learning and Reinforcement Learning.

Chapter 3 explores the Defender’s Dilemma and describes an experiment that shows its association with the success of a team in robotic soccer. The algorithm to recognize it is defined.

Chapter 4 presents the value function perspective and shows an artificial neural networks application that allows to forecast if a certain robotic soccer configuration will successfully end up in a goal scored or not.

Chapter 5 describes a robotic soccer simulator built for this research that is simpler than the RoboCup soccer simulator, and whose characteristics make it well-suited for reinforcement learning experiments. An experiment is reported about the creation of a simple reactive agent for this simulator, which basing on simple rules displays the emergence of Defender’s Dilemmas.
Another experiment is run in our robotic soccer simulator, about training a soccer agent to score goals with Deep Reinforcement Learning.

Chapter 6 gives the conclusions to thesis and our answers to the research questions, and presents further research that arises out of the thesis.

Chapter 2. Literature Review

2.1 Previous Works on Robotic Soccer Tactics Analysis

In the last 25 years many different approaches have been taken in the study of robotic soccer's tactics and strategies, in particular in the RoboCup environments, and also some study of human soccer tactics may be relevant and applied to robots. We present a selection of previous works, grouped by similar approach.

2.1.1 High-Level Actions Analysis

One type of analysis has been following the idea that it is possible to identify high-level actions, such as "dribble", "pass to a companion", parsing the RoboCup 2D log file that contains all base-level actions, and using some criterium to decide that a certain low-level sequence of actions constitutes a higher-level action. [Voelz et al. 1999] implemented such a higher-order events finder in their commentator system. [Abreu et al. 2010] wrote an analyser with the same purpose, basing only on cartesian coordinates and velocity of players and ball, discarding all other log information such as kick information and game stages (as the event "goal_l" after a goal). That resulted in a high-level parser that may miss some event in RoboCup 2D, but could be used also in real world human soccer. Once those low level and high level events are available, it is possible to analyse them to understand which chain of events may lead to a goal. For instance it is possible to identify a goal and then follow backwards all the preceding events to build a chain of actions that the scorer team performed.

[Almeida et. al 2013] described a similar procedure to find "goal plans" called "Setplays" (introduced by [Mota et a. 2010]), with the aim of reusing goal plans during matches, especially when it is easier to reposition players and actuate a plan as during kick-ins, kick-offs, or corner kicks.

2.1.2 Voronoi Diagrams and Space Subdivision

Another kind of analysis is about the spatial usage of the pitch dividing it through Voronoi diagrams, as suggested in [Matsubara et al. 1999] and [Shahri 2008]. A Voronoi diagram partitions a plane into different regions depending on a set of "seed points": every seed point generates a region and every point in space belongs to the region generated by the nearest seed point [Aurenhammer & Klein 2000]. Now, if we consider players as seed points, a Voronoi

diagram illustrates which are the areas “owned” by every player. In other words, those areas are the ones in which each player has a competitive advantage to reach the ball in case it falls inside it, or equivalently the ones in which there is relatively more freedom of action for the player with respect to other players’ actions. This subdivision may be made considering either players from only one team at once, or both teams together. Some RoboCup 2D soccer agent may internally use Voronoi Diagram to decide the player’s position, as described in [Dashti et al. 2006]. Efficient algorithms to calculate Voronoi Diagrams can be found in [Oishi & Sugihara 1995].

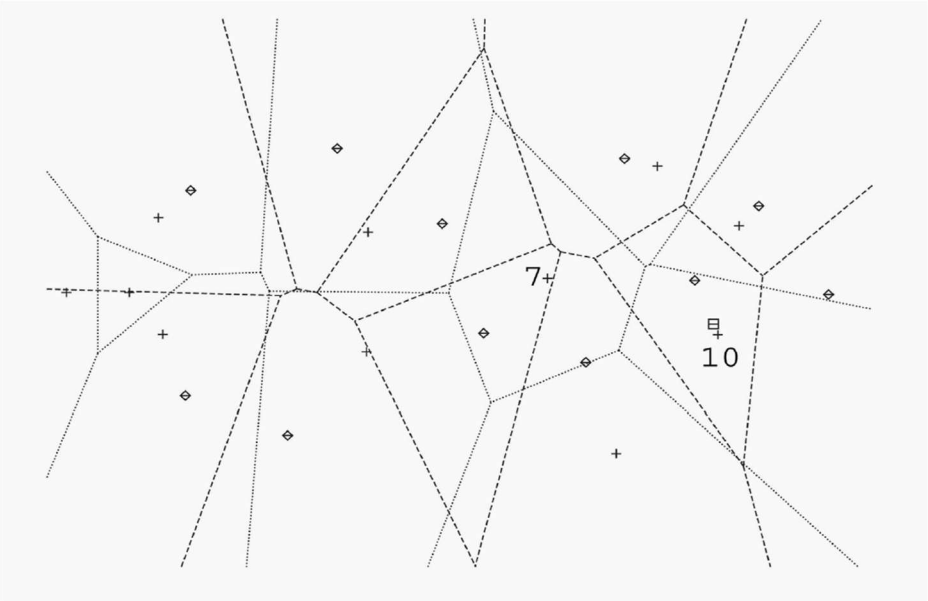


Figure 2.1: An example of Voronoi diagram [Matsubara et al. 1999], with each team generating separately a different diagram (one team has the space subdivided by dotted lines, the other by dashed lines).

To take in account the velocity and acceleration of the players and the ball, [Taki & Hasegawa 1998] introduced the concept of “Dominant Regions”, a sort of Voronoi Diagram modified in order to consider also the moving ability of the players in the human soccer game. [Nakanishi et al. 2008, 2010] applied Dominant Regions concepts to RoboCup Soccer and created efficient algorithms to calculate Dominant Regions.

Anther spatial statistics that is also possible to calculate is the area of the convex hull that encloses all players from one team and its centroid (the average position of the players), as done by [Frencken et al. 2011] and reported by [Rein & Memmert 2016] in human soccer, or measure pitch occupancy and numerical superiority in a certain area, based on the elliptical position range of each player, as proposed by [Silva et al. 2014] again for human soccer, and of course those statistics may be computed also in robotic soccer.

2.1.3 Network Analysis

The previous methods based on spatial statistics certainly reveal some characteristics of the opponent's playing style, but are not identifying defined tactics. Another approach is to analyse the interactions between players using social network analysis: players may be represented by nodes, and their interactions may be represented by edges. In such a framework, social network analysis algorithms may be used to calculate the importance or role of a player during some game action using statistics such as betweenness centrality, connection degree, and so on.

[Grund 2012] examined matches played by Premier League soccer teams and built a network where players were nodes and links between them were weighted depending on the total of passes made and received by the two players during the game. The author found that a greater interaction between players leads to better results, and a more centralized network leads to worse results.

[Ramos et al. 2018] provided a short review of network analysis in human sports, including soccer, as well as some idea about new methodologies using time intervals and bipartite networks. They suggest a partition of nodes composed of players and another partition composed of actions and create a network between those nodes depending on the actions taken in a certain time interval. The aim is to try to capture the dynamical aspect of the interactions and not only time-independent statistics. That kind of analysis can be applied to robotic soccer too.

In the study by [Cintia et al. 2015] about human football the pitch is divided into zones, and a network is created in which each zone is a node, and links between nodes are weighted depending on the number of passages between the two zones. Then standard network analysis can be done in such a model to answer many questions. For instance it could reveal from what zone to what other zone usually the game action moves. While authors use networks (dual relationships), using hypernetworks instead of networks could provide a richer analysis: instead of using a link between two vertices (zones) it's possible to use a hypersimplex connecting orderly three or more vertices/zones, that is a sequence of passages, and then do hypernetworks analysis.

This methodology looks useful, but is not complete because it excludes many aspects of the game such as the movement of the players through time and how they collaborate.

2.1.4 Coarse Spatio-Temporal Patterns

Methods that look for spatial-temporal relationships between players (and ball) may be more effective for the purpose of identifying tactics, since one of the most salient aspect of a tactic is the disposition of the players in the field and the dynamic of their interaction.

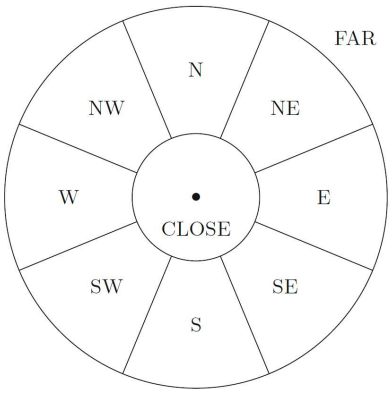


Figure 2.2: Qualitative positional relationships between 2 players [Warncke & Uhrmacher 2016].

A method to match spatial-temporal patterns is described in [Warncke & Uhrmacher 2016]: player positions from RoboCup 2D games are employed to build a graph, and then some subgraph pattern matching algorithm may be run to find configurations of interest. The method works as follows: first of all we establish a simplification mechanism to describe in a qualitative way the position of one object (player or ball) with respect to another. We define a minimum distance under which the spatial relationship between two objects is generically “close”, a maximum distance over which the relationship is generically “far”, and within those two distances the relationship is defined by a 8-fold subdivision of the space, using for instance cardinal points: North, NorthEast, East, SouthEast, South, SouthWest, West, NorthWest.

Then for each object at each time frame we instantiate a node in the network. Each object node has links towards all other objects and the link has a label identifying the qualitative relative position. Each object node has also a link towards a node that identifies the time frame. In addition, each object node has a link towards a node that identifies the object itself.

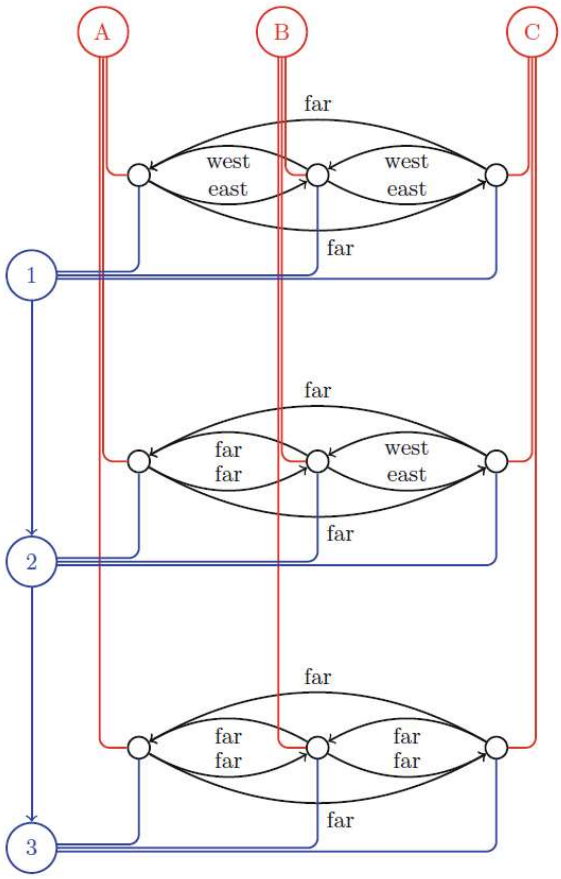


Figure 2.3: example of spatiotemporal graph in [Warncke & Uhrmacher 2016]: the blue nodes with numbers 1/2/3 identify the timeframes, the red nodes with labels A/B/C identify the different objects, the other nodes (small black circles) are instantiation of objects inside a time frame and are linked to each other by links whose labels are the qualitative spatial relationships.

Once such a graph is built, it is possible to apply pattern-matching algorithms to find occurrences of some configurations. For instance one may want to find occurrences in which at time t player A is West of player B and near the ball and at time t+1 player A is East of player B and still near the ball, with B not near the ball. That would mean that player A has been able to dribble player B.

Three criticalities arise with this method: the first is that finding a subgraph configuration (a problem named “subgraph isomorphism”) is NP-Hard. Some algorithms as [Ullman 1976] may perform well on some configuration, as well as algorithms implemented in graph databases like Neo4j [4], but still they can’t generally guarantee an acceptable speed. The second criticality is that this method is just a pattern matching method, meaning that we should already be knowing exactly which configuration we are looking for. It’s not useful if instead we want to learn which configurations are relevant with no prior knowledge. The third criticality is about the fact that this method uses “qualitative” relationships, that means that we create a categorical variable out of a continuous one, and in doing so we do some sort of arbitrary

approximation that may lose important information. Once the variable is qualitative, it's not possible anymore to calculate "how much" a certain configuration is similar to another. A quantitative method instead could be able to find patterns that are similar at least within a certain threshold or at least in some aspect, or even that are similar at scale.

A substantial number of other publications is also about reducing the spatial-temporal relations of players to qualitative variables before analysing them, but differently from the previous method instead of organizing that information into an object-based graph, they rather insert it into a TRIE (a particular form of ordered data tree).

One of those is the research by [Ramos & Ayanegui 2008]: they analysed both the spatial relationships of players and the trajectory of the ball in RoboCup simulation. Players' positions populate a planar graph whose topological properties are checked to see if the formation changed (as when the graph is not planar anymore). The ball trajectory is encoded using the "Freeman codification" that consists in segmenting the trajectory and coding each segment with a number from 1 to 8 that approximates the direction of the segment in the pitch (for instance "north", "north-west", "west" and so on). Then chains of those segments may form higher-level components, and an analysis of the most frequent sequences of those higher-level components may be carried on using trees (in particular TRIEs) to find the most frequent path types (trajectory of the ball). While informing on the ball dynamics, this method only approximately analyses the movement of all players, resulting not enough to the scope of identifying full tactics.

[Lattner et al. 2006] used a qualitative method to express spatial relations among objects in RoboCup 2D, dividing the 2D space in 8 directions and using 5 "qualitative" (or better to say "coarse") values of distance. So, objects such as players and ball can be put in relation with each other using those coarse properties to build predicates. Then a sequence of predicates that includes also time constraints describing the duration for each predicate may define a pattern. Hence a pattern represents a rule: whenever a predicate is true within its time constraint, for each predicate of the pattern, then an instance of the pattern is found. So it becomes possible to apply pattern matching algorithms, and for frequent patterns it is possible to learn to predict future behaviour through methods such as those based by association rules. A consideration may be done about the fact that with this method is not easy to find "similar" patterns that are not reproducing exactly the original: many combinatorial versions of the original with little modifications should have to be taken in account to be compared, and depending on pattern length and predicate variety this could lead to combinatorial explosion.

2.1.5 Action Sequences Pattern Mining

[Kaminka et al. 2003] used a preprocessor for the RoboCup 2D log file to obtain a stream of atomic events and actions for each team, inside each stream an uninterrupted (not interrupted by the opponent or the referee) sequence of those atomic actions forms a “segment”. In the following stage each segment is inserted in a TRIE (using a different TRIE for each team): that is motivated by the fact that in such a data structure its spatially and computationally cheap to explicitly represent subsequences and count them. Finally the TRIE is traversed to find useful subsequences and calculate statistical dependencies (testing whether in all the segments the co-occurrence of the prefix events and the inserting event is due to chance). Using the values of statistical dependencies, the most likely “not-by-chance” segments are identified as to be actual tactics. As the previous method, this looks to be brittle in case of having similar but not identical patterns.

Comparable techniques have been realized by [Huang et al. 2003], [Fathzadeh et al. 2006] and by [Iglesias et al. 2008].

[Ze-Kai et al. 2013] applied some pattern mining algorithm to RoboCup 2D. They divided the pitch into small regions and sequentially mined past games logs to find successful sequences of actions. Then each sequence was processed extracting the series of regions traversed, and the sequences that were more frequently repeated were retained, using an algorithm derived from AprioriAll. Then, basing on a visual examination of the results, authors elaborated manually some new tactics for their RoboCup 2D agent. An evident problem here is that slightly different patterns would be counted as different patterns because of the nature of the used algorithm: that algorithm is unable to generalize. So, many slight variations of the same pattern would be discarded if not frequent enough to reach the “support threshold”, while ideally all together they would be much bigger than the threshold in number. It is clear that a much better method would be one able to generalize and to smoothly extract regularities.
[Sprado and Gottfried 2009] studied the interactions between Robocup 2D soccer players: they identified 25 qualitative atomic patterns that express how two teammates evolve their relative position from a period t0 to a period t1. Those patterns describe facts such as “the first player moves North East and the second moves South”.

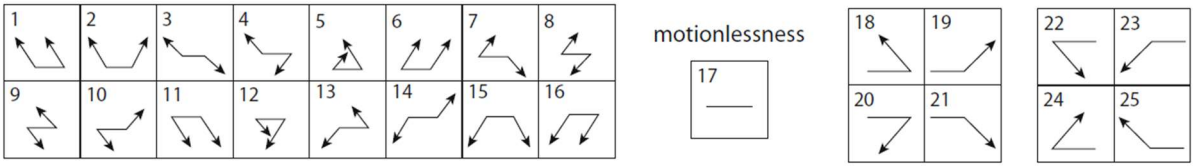


Figure 2.4: two players atomic patterns in [Sprado and Gottfried 2009].

Subsequently authors analysed the presence of those patterns within the actions extracted from game logs of different matches and noted that the distribution was similar among different matches and different teams, revealing a probable high component of reactivity in the actions of agents. Authors then introduced the concept of “complex motion pattern” that is formed by a set of many atomic patterns of consecutives time instants and related to more than one player, and advocated the usage of it within the tasks of plan recognition, but didn’t elaborate further the idea and did not provide any algorithm.

The criticalities of qualitative variables methods and of symbolic pattern matching seen above may be overcome by Machine Learning algorithms, since they may be employed to find patterns that are “similar” without the need to be exactly the same.

Speaking of machine learning, some studies used clustering as a method to analyse the behaviour of teams.

2.1.6 Similarity Clustering

One kind of analysis is to evaluate the frequency and modalities of kicks. RoboCup 2D log files contain the chronological list of all actions taken by all agents, so it is possible to parse it to obtain that information. [Nakashima et al. 2015] followed the hypothesis that similar sequences of kicks may be hinting similar tactics. They extracted passes and dribbles from logs, and used as features the position of the first kick and the distance between consecutive kicking points. From that, they derived a distribution of kicks for each team, and they used an agglomerative clustering algorithm, using the Wasserstein Metric (also known as Earth Mover Distance) as similarity distance, to cluster the teams. They found that this kind of clustering helped them in reducing the uncertainty in predicting the result of a game (they did not specify what method was used for prediction). The usage of Wasserstein Metric makes this method computationally unfeasible when the number of kicks in the kicks distributions grows. [Fukushima et al. 2019b] did a similar work of comparing teams by computing distributional distances in their kick distributions. To improve the computational speed, they considered the kicks distribution as a continuous one, defined by gaussian kernels, and used not only the

Wasserstein Metric but also L^2 Distance and Jensen-Shannon Divergence. They found out that the similarity measure computed in this way was correlated with human subjective evaluations. Nonetheless these experiments do not appear to strongly prove the thesis that similar kicks distributions really imply similar tactics. Moreover, they completely disregard other spatial information and the overall positions of players.

Even if these approaches allow to cluster teams depending on the frequencies of high-level actions, they are not generalizing the higher-level sequences of actions into styles of games or tactics, and they do not include methods to find similar sequences in other games.

[Erdogan and Veloso 2011] analyzed the log files from RoboCup Small Size League games and extracted sequences of coordinates of the attacking players. Each "episode" contained a certain number of contemporary trajectories of players, forming a set of trajectories. A distance metric between trajectories was defined, based on Hausdorff distance:
For trajectories $T1$ and $T2$, the distance is:

$$H(T1,T2) = max\{\, max_{p\epsilon T1} min_{q\in T2} d(p,q)\,, max_{p\epsilon T2} min_{q\in T1} d(p,q)\,\}$$

Where $d(p,q)$ is the Euclidean distance between point p and point q . The algorithmic complexity of that distance is $O(mk)$ where m is the length of the first trajectory and k the length of the second.

Basing on that, a Similarity metric between sets of trajectories $S1$ and $S2$ was defined:

$$Sim(S1,S2) = min_{P\in P^n} \sum_{i=1}^{n} H(S1[i], S2[P(i)])$$

Where n is the number of trajectories in a set, $P \in P^n$ is a permutation of all the trajectories in a set, and $S1[i]$ is the trajectory number i in the set $S1$. Such a similarity is measured for each horizontal/vertical symmetry of the trajectories, and the minimum value is selected. The complexity of such similarity metric is $O(mk!\,n)$.

An aggregative clustering algorithm has been used to cluster sets of trajectories based on the similarity metric.

Once clusters are defined, the similarity metric may be used again to compute the average of that metric between a new trajectories set and those of each cluster, and ideally assigning the cluster with minor average metric as the one where the new trajectories set belongs.
While this method seems to work in clustering together similar trajectories sets, as we already said for clustering algorithms, it does not allow to generalize any strategy or tactics, it just groups together similar attack behaviour. The algorithmic complexity may be an obstacle if used with large volumes of data.

[Adachi et al. 2017] used the logged data from RoboCup Small Size League 2015 competition and with hand-made algorithms they detected some actions such as pass, pass wait, shoot, ball-keeping, clear the ball and so on. Then used that information to build a sequence of actions for each robot. Then they invented a metric for the similarity of action sequences, and used that metric to cluster actions in games. The same note about action clustering applies here as well: clustering actions does not mean finding strategies.

[Larik & Haider 2015] presented a method to find high-level actions such as kicks and dribbles in RoboCup 3D soccer matches. Then they used that data to classify teams either in “weak”, “medium”, or “strong”, using Naïve Bayes classifier and Neural Networks. They also used a clustering algorithm on the same data to cluster teams in two different clusters with different abilities. The limit of this approach is that classifying or clustering teams with just a “goodness” score is a very poor information and is not capturing anything that may resemble tactics or strategy.

2.1.7 Machine Learning Classification

[Almeida et al. 2009] applied some machine meaning algorithms to classify the type of formation (the player distribution on the pitch, for instance 4-4-2, or 3-5-2 and so on) of a RoboCup 2D team using the players positions (x, y) as input variables. The best results in that study were obtained through Support Vector Machines with sequential minimal optimization. Note that this was not about recognizing complete tactics, but only about the formation.

[Floyd et al. 2008] applied Case-Based Reasoning to RoboCup 2D: they created an agent that was able to repeat actions from previously examined log files. Authors analysed existing match logs and extracted the so called “cases”: a description of a certain game situation as perceived by agents (for instance containing the distance and position of other players or objects as descriptive variables) followed by the successive behaviour sequence. When during the play of a game the agent finds a similar game situation, it retrieves the correspondent behaviour

sequence and enacts it. To find the most similar game situation they used a k-nearest neighbour classifier, together with a genetic algorithm that adjusts the weights of the variables in the classifier's distance function. Even if this seems to work for the purpose of choosing the most similar case when repeating the behavioural sequence, it is not doing any generalization on the cases, the cases are just stored in memory for comparison and there it is not actually any "learning" or information extraction: so, it is not useful for analysing purposes. As a side note, it is also not a perfect solution for action selection purposes: using k-nearest neighbours means that the number of possible cases must be maintained low, with obvious limitations on the completeness of the replicable scenarios.

[Laviers et al. 2009] used Support Vector Machines to classify the tactics played by opponents in a simulated American Football game. SVMs have been trained using data that was generated in the following way: starting from well-known initial game formation configuration, a simulation created controlled spatial-temporal players traces for a combination of different attack and defence plays. Those traces were provided to SVMs for training. Then the trained SVMs was employed to classify the kind of tactics that an opponent was playing, and the system was able to do the classification right after a few of opponent actions, at the beginning of each tactic. This enabled to change dynamically the play to choose the best plan as soon as the opponent scheme was discovered. The fact that this study was based on tactics beginning only from initial formation configurations may pose the question of how well it would perform on soccer on game actions that are not beginning from starting formation. Moreover, SVMs alone are not able to learning complex and hierarchical combinations of concepts, but this may be fixed using neural networks instead.

2.1.8 Machine Learning Prediction

[Raines et al. 2000] wrote a software to analyse logs from RoboCup 2D and understand if a sequence of actions is going to have success or failure. The analysis was carried on parallelly on 3 aspects: Individual Key Agents Actions, Key Pattern of Interactions, Global Team Model. The first aspect analyses if a certain individual action of an agent such as a dribble or a pass is going to be successful. Individual actions from past logs are used as input to train a Decision Tree with algorithm C5.0. The trained Decision Tree is then supposedly able to predict success or failure for new cases, and being a decision tree it explicitly encodes the rules for success or failure that are understandable by humans.

For the second analysed aspect, "Key Pattern of Interactions" the procedure takes sequences of actions of different agents as input and processes them through a pattern matching algorithm to possibly find a match against previously (user-defined) predefined patterns. For

the third aspect, “Global Team Model”, the method calculates global statistics about the game, as for instance the distance between the ball and the closest defender, or the angle of closest defender with respect to goal, and trains a Decision Tree with those statistics as input. This system was also able to interpret the forecast and give advices about how to correct a likely-to-fail play into a successful one. The choice of using decision trees in the first and third aspect brings indeed the advantage of producing learned rules that are human readable, but in this context I would suspect that they may not generalize well given the richness of the variety of possible scenarios, since decision trees are known to overfit when applied to data with complex structure [Hastie et al. 2009]. Using pattern matching in the second aspect exposes to the usual problems of pattern matching, as written above: it is not robust with respect to variables with a certain number of (even small) differences in configurations, and it is computationally expensive when the variables’ space grows in dimensions. Moreover, pattern matching is computed against user-predefined patterns, the algorithm is not learning new patterns.

An analytic system that may have some point in common with the previous one is the one included in the RoboCup 2D coach agent by [Kuhlmann et al. 2005], The UT Austin Villa 2003 Champion Simulator Coach. It reads log files of past matches and extracts the position of the players, of the ball and the distance between them to use as input features. Then it trains a set of decision trees, one for each player, using algorithm C4.5 (J48 from Weka package) in order to predict the next high-level event to occur, given the current state of the game. Then those decision trees could be used in real time during matches to make predictions, and a subsystem uses those predictions to elaborate advice for players. The coach agent is also able to recognize formations. To which extent this whole system is reliable is unclear: the agent was a winner of the coach world competition but since it was composed of many subsystems it’s not possible to measure the actual advantage given by this technique.

According to the authors the coach agent was not statistically better than the second classified agent, who was not using this kind of methods, but it was statistically better than the third classified and all the others. It is also possible that this prediction techniques were giving an advantage even in case of low accuracy: imagine an accuracy of 30% that is considered generally a low accuracy for classification in standard well-structured data, it is still a good advantage with respect to problems in which a random guess would account for a much lower accuracy, as when the cardinality of possible predicted actions is big. So we could regard this technique as an important starting point, keeping in mind that decision trees may not be the best choice for structurally complex data with many dimensions: in this case it made sense because they could be quickly interpretable in order to improve the planning of next actions,

but if the aim is focused especially on identifying tactics then neural networks could generalize better, and may represent a preferable investigation path.

Another study that used decision trees to predict opponent's actions using players' and ball's positions as input was made by [Karimi and Ahmazadeh 2014].

[Beetz et al. 2005] made an analysis software for RoboCup 2D log files, but designed it to be applicable also to human soccer data. It implemented various functionalities: it could predict the success of a shot using decision trees with input variables such as the distance between the ball and goal, the number of defenders between the ball and the goal, the minimum distance the attacker could carry the ball, the largest unblocked angle segment toward the goal. It could also compare properties of the two different playing teams such as shooting skills, mining for situations in which one team succeeded and the other failed. It's not described how this was concretely realized, but it seems that the representation of actions was made by predicate logic. If that is the case, then the usual problems of lack of generalization of "Good Old-Fashioned Artificial Intelligence" could arise. Another feature of the software was to infer a Gaussian distribution on the position of each player during the game. In addition, it calculated statistics about passing, dribbling, shots.

2.1.9 Pattern Recognition with Neural Networks

Several studies consider artificial neural networks to be good methods for identifying tactics. [Visser et al. 2001] recognized formation in RoboCup 2D dividing the pitch into 64 zones through a 8x8 subdivision, and using the presence of players into the zones as input features built an artificial neural network able to classify formations.

Other studies use Self Organized Maps [Kohonen 1990] (also named "Kohonen Feature Maps") to reduce the dimensionality of player actions so to examine their closeness in that reduced space. [Dutt-Mazumder et al. 2011] propose their use, while [Wünstel et al. 2001] used that method to project in a 2D Self Organized Map the individual actions of RoboCup 2D players and compare in what zones of the map the actions of two players of different teams end up more frequently. This may apparently hint about similarities in play style, but it does not guarantee that the dimensionality reduction is really at tactical level. Also, this seems applicable only to individual actions.
Some authors, [Perl 2002, 2004], [Uthmann, & Dauscher 2005] , [Perl & Dauscher 2006] , proposed using a derived version of Self Organized Maps, called Dynamically Controlled Network to analyze RoboCup, soccer games or other human sports.

[Grunz et Al. 2012] used Dynamically Controlled Network to analyse dynamics from 2006 FIFA World Football Championship matches. In this architecture the input neurons should be set to players' coordinates within the field (only 4 players in this research), and the map neurons during training will self organize in a way that (once the training is complete) similar inputs would be mapped to the same neuron or to neighbouring neurons. The map neurons' weights will represent "prototypical configurations". This means that a new, not seen before game configuration could be identified as similar to an example configuration if its neuronal mapping will not result far from the neuronal mapping of the example. In our case it would mean recognize similar disposition of players in the pitch. By "similar" it is meant not only with a similar position, but also with a similar shape but different scale - even if this last property can't be equally guaranteed for each configuration due to the lower dimension (2 dimensions in this case) of the Kohonen Map with respect to the input dimension (8 dimensions: 4 players multiplied by 2 coordinates).

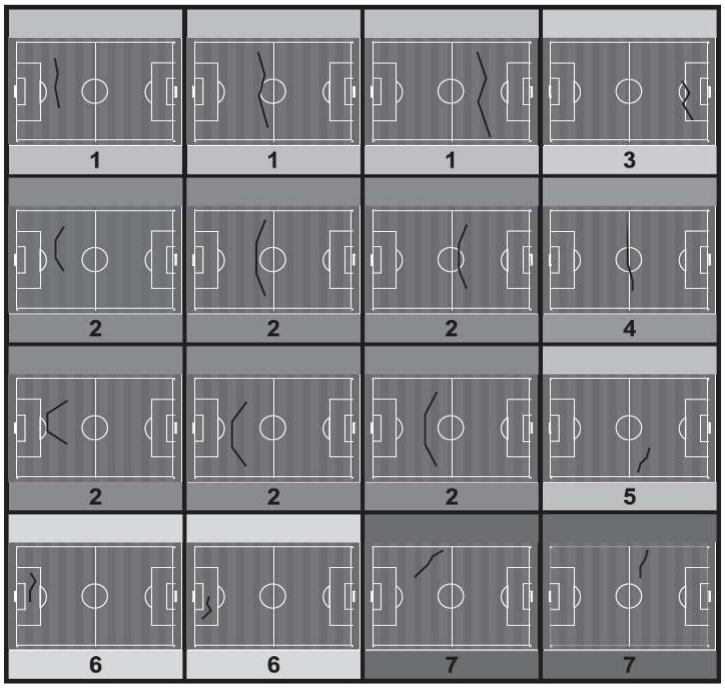


Figure 2.5: [Grunz et Al. 2012] An example of Dynamically Controlled Network of 4x4 neurons. Every square represents a map neuron, and inside its game field quadrant there is depicted its prototypical game situation. The black polylines are realized connecting the positions of 4 defending players. We notice that when two neurons encode prototypical configurations that are similar, they are neighbours. Hence it is possible to cluster similar neurons: the number in the low part of the squares indicates the cluster.

After noticing that the dimensionality reduction was losing details, authors decided to create two different Dynamically Controlled Networks: one for for x coordinates and one for y coordinates. Then they computed the output of those networks for each time point. The

resulting multidimensional time series was read through a sliding time window and each time block was treated as input to feed another Dynamically Controlled Network to typify the movement. At this point authors repeated the same procedure with offense players data.
Then they used these two time series and the ball position time series to feed another Dynamically Controlled Network, resulting in the typification of the interaction between defence, offense and ball. The final step then was to give a semantic meaning to the neurons of the last Dynamically Controlled Network, in order to interpret the output when the network was employed to analyse new cases.

Authors used categorized data as input of the whole architecture, and an expert (person) associated manually a meaning to the mapped neuron. This last part has evident room for improvement: a completely automated procedure is advisable, both for efficiency and quality purposes. Another criticality of this system is the usage of sliding windows: since SOMs/DyCoNs have fixed input lengths, the sliding windows must be of fixed length, but this implies that the analysed tactics should be of fixed length too. This is obviously a great problem, since some tactics' duration could be smaller or bigger of the sliding window size. Authors propose a partial solution to this: calculating the whole process different times with sliding windows of different lengths. Nonetheless, other problems remain: since many similar tactics may have some difference in execution speed or may have some missing block but still be similar, and a fixed windows system would fail in discover their similarities.

2.1.10 Prediction with Neural Networks

[Copete et al. 2015] used neural networks to predict the motion direction of players and ball in RoboCup 2D games. They used as input features the position of the 22 players and the ball extracted from past games' logs. Then they trained a shallow network with only one hidden layer to predict the position of the ball: it had 46 neurons in the input layer, 46 neurons also in the hidden layer, and 2 neurons in the output layer. To predict the players' directions they used a deep autoencoder made by 8 layers (4 mirrored layers) with a number of neurons for the layers respectively of 250, 150, 80, 30, 30, 80, 150, 250, a temporal sliding window for the input features and a recurrent connection from the predicted positions of time t to the last input positions of time t+1. Considering the huge space of possible solutions for such a prediction, the results were encouraging, predicting the ball direction with an average success of 84% and predicting the players' directions with an average success ranging from 50% to 69% depending on the analysed team. Anyway since the training set was composed of only 10 games, it's not possible to exclude some overfitting.

[Zare et al. 2018] implemented a multilayer neural network in their Cyrus2D soccer agent (for RoboCup 2D) and used it to try to predict some aspect of opponent behaviour. In particular they trained a neural network with the positions and velocity of the players and the ball of other teams' past games and then used the network in real-time to try to forecast which opponent player was more likely to be the target of a ball passing during any opponent team's attack. They results were promising since they claim to have been able to forecast the target of a ball passing among two top-scored players (two players that received the higher score by another criterion such as closeness to the goal) with an accuracy of 90%.

[Michael et al. 2018] used a recurrent neural network in self-supervised way: it was fed by players' and ball's coordinates of RoboCup 2D games, and it was trained to predict the next tick's positions. The internal representation of such a network captures information about the dynamics of a team, so the vector of activations of "reservoir" neurons may be used to distinguish one sequence of actions from another, hopefully "resuming" their characteristics. Those activations were then clustered with the X-means algorithm (a derivation of k-means algorithm) and was able to cluster together seemingly similar action sequences.

The problem with this method as well as with the previous two by [Zare et al. 2018] and [Copete et al. 2015], is that they used the players coordinates as input data for the neural network with the implicit assumption that the player with a certain uniform number will play always in the same role/position: putting players coordinates in fixed input slots unavoidably assumes that. This may or may be not true in reality, depending on how it's implemented the soccer agent. Just switching the uniform number for a couple of players would change the forecast even if the players are still playing in the same exact way.

Because of this mechanism, the algorithm can't generalize well with different teams: you must train a different neural network for each team you play against, and also hope that the opponent team does not just switch uniform number in different training games or when playing with you. This issue can be tackled with a Convolutional neural network, or more generally any network that uses as input not the ordered list of player coordinates, but some kind of discretized image of the player's position on the pitch: in that way the uniform number would not really matter: if at a certain position there is an opponent player that would be recognized and generalized.

2.1.11 Multilevel Modeling with Hypernetworks

For what concerns the theoretical framework of hypernetworks, [Johnson 2001] showed that it is possible to model the interactions between soccer (and Robot Soccer) players through

hypernetworks, describing multilevel structures. Those interactions and multilevel structures may coincide with the concept of tactics. This is discussed in detail in paragraph 2.2 .

2.1.12 Comparison of the Methods of Analysis

At this point it is useful to compare the most relevant methodologies found in literature, using three tables with different criteria.

The tables below show that many methods do not generalize well: those methods based on symbolic pattern matching, TRIE analysis, decision trees and k-nearest neighbours. They are not fit for this research since generalization is important to recognize the same kind of phenomenon happening in different circumstances.

Other methods are limited to only some aspect of the game such as methods that examine only team formation, ball dynamics, kicks distributions. Those methods are not solving the problem of capturing the emergence of a team behaviour. Then there are methods based on neural networks that look promising but need further research since they have important critical issues: in particular they may lack generalization when facing different opponent teams and in case of players changing uniform number with respect to training data.

Papers with the asterisk * are discussed in next section 2.2 .

Literature Comparison Table 2.1: **SPACE INTERPRETATION**

REFERENCE	VORONOI DIAGRAM	PITCH SUBDIVISION	SINGLE PLAYERS POSITIONS
Abreu et al. 2010	No	No	Quantitative
Adachi et al. 2017	No	No	Quantitative
Almeida et al. 2009	No	No	Quantitative
Almeida et. al 2013	No	No	Quantitative
Beetz et al. 2005	No	No	Quantitative
Cintia et al. 2015	No	Yes	Approximated to pitch subdivision
Copete et al. 2015	No	No	Quantitative
Dutt-Mazumder et al. 2011	No	No	Quantitative
Erdogan & Veloso 2011	No	No	Quantitative
Fathzadeh et al. 2006	No	No	Qualitative
Floyd et al. 2008	No	No	Quantitative
Frencken et al. 2011	No	No	Quantitative
Fukushima et al. 2019b	No	No	Quantitative
Grund 2012	No	No	Quantitative
Grunz et Al. 2012	No	No	Quantitative
Huang et al. 2003	No	Yes	Qualitative
Iglesias et al. 2008	No	No	Qualitative
Johnson 2001	No	No	Qualitative and quantitative
Johnson & Iravani 2007*	No	No	Qualitative and quantitative
Johnson & Rossi 2018,2020*	Yes	Yes	Qualitative and quantitative
Kaminka et al. 2003	No	No	Qualitative
Karimi and Ahmazadeh 2014	No	No	Quantitative
Kuhlmann et al. 2005	No	No	Quantitative
Larik & Haider 2015	No	Yes	Quantitative
Lattner et al. 2006	No	No	Qualitative/Coarse
Laviers et al. 2009	No	No	Quantitative
Matsubara et al. 1999	Yes	No	Quantitative
Michael et al. 2018	No	No	Quantitative
Nakanishi et al. 2008, 2010	Yes, with velocity and acceleration	No	Quantitative
Nakashima et al. 2015	No	No	Quantitative

Perl 2002, 2004	No	No	Quantitative
Perl & Dauscher 2006	No	No	Quantitative
Raines et al. 2000	No	No	Quantitative
Ramos & Ayanegui 2008	No	No	N/A
Ramos et al. 2018	No	No	Quantitative
Shahri 2008	Yes	No	Quantitative
Silva et al. 2014	No	No	Quantitative
Sprado & Gottfried 2009	No	No	Qualitative
Taki and Hasegawa 1998	Yes, with velocity and acceleration	No	Quantitative
Visser et al. 2001	No	Yes	Approximated to pitch subdivision
Voelz et al. 1999	No	No	Quantitative
Warncke & Uhrmacher 2016	No	No	Qualitative
Wünstel et al. 2001	No	No	Quantitative
Zare et al. 2018	No	No	Quantitative
Ze-Kai et al. 2013	No	Yes	Approximated to pitch subdivision

Literature Comparison Table 2.2: **DYNAMIC ANALYSIS**

REFERENCE	GRAPHS	ANALYSIS METHOD	GENERALIZATION
Abreu et al. 2010	No	Sequence reconstruction	Low
Adachi et al. 2017	No	Clustering actions similarities	Medium
Almeida et al. 2009	No	Support Vector Machines	High
Almeida et. al 2013	No	Defined conditions	Low
Beetz et al. 2005	No	Decision tree, Gaussian distribution	Low
Cintia et al. 2015	Networks	Frequent Network Paths	High
Copete et al. 2015	No	Neural Networks	High, but low with new teams or uniform change
Dutt-Mazumder et al. 2011	No	Dynamically Controlled Networks	High
Erdogan & Veloso 2011	No	Clustering coordinates sequences	Low
Fathzadeh et al. 2006	No	Pattern mining	Low
Floyd et al. 2008	No	K-Nearest Neighbour	Low
Frencken et al. 2011	No	Convex Hull, Centroid	Medium
Fukushima et al. 2019b	No	Distribution distances of kicks (Wasserstein, L^2, Jensen-Shannon)	Medium
Grund 2012	Networks	Network Analysis of passes	Medium
Grunz et Al. 2012	No	Dynamically Controlled Networks	High
Huang et al. 2003	TRIE	TRIE search with statistical test	Low
Iglesias et al. 2008	TRIE	TRIE search with statistical test	Low
Johnson 2001	Hypernets	Q-Analysis	High
Johnson & Iravani 2007*	Hypernets	Q-Analysis	High
Johnson & Rossi 2018,2020*	Hypernets	Neighbourhood structures	High
Kaminka et al. 2003	TRIE	TRIE search with statistical test	Low
Karimi and Ahmazadeh 2014	No	Decision Tree	Medium
Kuhlmann et al. 2005	No	Decision Tree	Low
Larik & Haider 2015	No	Neural nets/Naïve Bayes,Clustering	Medium
Lattner et al. 2006	No	Sequential Pattern Mining	Low
Laviers et al. 2009	No	Support Vector Machines	High
Matsubara et al. 1999	No	N/A	Medium
Michael et al. 2018	No	Recurrent Neural Networks, Clustering	Medium
Nakanishi et al. 2008, 2010	No	N/A	Medium

Nakashima et al. 2015	No	Clustering distributions of kicks	Medium
Perl 2002, 2004	No	Dynamically Controlled Networks	High
Perl & Dauscher 2006	No	Dynamically Controlled Networks	High
Raines et al. 2000	No	Decision Tree	Low
Ramos & Ayanegui 2008	TRIE	TRIE search	Low
Ramos et al. 2018	Bipartite nets	Network analysis	Medium
Shahri 2008	No	N/A	Medium
Silva et al. 2014	No	Pitch occupancy, elliptical ranges	Medium
Sprado & Gottfried 2009	No	(unknown) pattern matching	Supposedly low
Taki and Hasegawa 1998	No	N/A	Medium
Visser et al. 2001	No	Neural Networks	High
Voelz et al. 1999	No	Defined conditions	Low
Warncke & Uhrmacher 2016	Networks	Subgraph Isomorphism	Low
Wünstel et al. 2001	No	Kohonen's Self-Organizing Map	High
Zare et al. 2018	No	Neural Networks	High, but low with new teams or uniform change
Ze-Kai et al. 2013	No	A priori all	Low

Literature Comparison Table 2.3*:* **CRITICAL ISSUES**

REFERENCE	CRITICAL ISSUES FOR OUR AIMS
Abreu et al. 2010	Only finds sequences of actions, does not generalize.
Adachi et al. 2017	Clustering does not imply recognizing situations or tactics
Almeida et al. 2009	It analyses only formations.
Almeida et. al 2013	Designed for players repositionig situations
Beetz et al. 2005	Doesn't analyse whole team behaviour. Low generalization.
Cintia et al. 2015	Limited to passages between regions, not to complete sequences of actions.
Copete et al. 2015	Predicts positions, but it ignores whole team properties. Unreliable if switching uniforms or playing against an unknown opponent.
Dutt-Mazumder et al. 2011	Only proposed, not experimented.
Erdogan & Veloso 2011	Does not identify tactics or strategy. Does not generalize.
Fathzadeh et al. 2006	Not much informative. Does not generalize well.
Floyd et al. 2008	Does not generalize well.
Frencken et al. 2011	Does not give much information about situation or tactics
Fukushima et al. 2019b	It does not take in account full team position. Weak assumption that same distribution of kicks implies same game style.
Grund 2012	Analyzing passes among player is only one of the many aspects of game
Grunz et Al. 2012	Sliding windows of fixed length may be problematic for time generalization. It's not completely automated and needs human intervention.
Huang et al. 2003	Does not generalize well.
Iglesias et al. 2008	Does not generalize well. Only qualitative.
Johnson 2001	Method conceived but not implemented.
Johnson & Iravani 2007*	Method conceived but not implemented.
Johnson & Rossi 2018,2020*	Not completely automated: it requires final step of human examination.
Kaminka et al. 2003	Does not generalize well. Only qualitative.
Karimi and Ahmazadeh 2014	Only applicable to predict micro-level actions
Kuhlmann et al. 2005	Does not generalize well.
Larik & Haider 2015	Only finds supposed similarities between teams or assesses the goodness of the team.
Lattner et al. 2006	Does not generalize well. Only qualitative/coarse.
Laviers et al. 2009	Proven to work only with begin of game configuration of formation.
Matsubara et al. 1999	Does not look for patterns, only for Voronoi diagram's properties.
Michael et al. 2018	Fragile to player order change
Nakanishi et al. 2008, 2010	It does not look for patterns, only for dominant region's properties

Nakashima et al. 2015	It ignores full team position. Weak assumption that same kicks distribution implies same game style.
Perl 2002, 2004	Only proposed, not experimented in RoboCup or soccer.
Perl & Dauscher 2006	Applied to human squash not RoboCup, only single player trajectory.
Raines et al. 2000	Focused on success/failure prediction. Does not generalize well.
Ramos & Ayanegui 2008	Dedicated just to ball dynamics.
Ramos et al. 2018	Method proposed but not experimented. Implementation details missing.
Shahri 2008	Does not look for patterns, only for Voronoi diagram's properties
Silva et al. 2014	Does not convey a lot of information
Sprado & Gottfried 2009	Only qualitative. Authors did not provide the actual algorithm.
Taki and Hasegawa 1998	Does not look for patterns, only for dominant region's properties.
Visser et al. 2001	It analyses only formations.
Voelz et al. 1999	Only finds high-level actions, descriptive
Warncke & Uhrmacher 2016	NP-Hard. It only finds known patterns that we defined in advance. Qualitative.
Wünstel et al. 2001	Analyses only single player trajectory.
Zare et Al. 2018	Only forecasts pass target. Unreliable if switching uniforms or novel opponent.
Ze-Kai et al. 2013	Does not generalize well.

2.1.13 Insights from Literature Review

The literature review exposed many criticalities of existing methods for analysing RoboCup 2D Soccer, bringing evidence about the limited applicability of symbolic processing and qualitative or coarse variables with respect to this topic, hence a more promising path seems to be a machine learning approach. Among many possible machine learning techniques, neural networks are considered objects with great ability to generalize, even if at the cost of not being easy to train.

Moreover, deep learning (the term to describe neural networks with multiple layers and deep probabilistic graphical models) in general is known to be able to discover and encode properties of the input features into middle neuronal layers in a distributed representation, obtaining what is called "learned representations" or "learned features" [Le et. al 2011] that are interpretable as high-level concepts. This seems to fit well into tactics analysis, where each discovered concept may refer to some intrinsic aspect of a tactic. To give a concrete example, this could mean learning information about a prototypical geometrical shape of the team, its location or its scaling, its temporal dynamics. For instance, once trained an

hypothetical neural network able to recognize some tactical aspect, if we process a new input configuration with the current game data, one middle layer neuron may turn on in case a certain group of players occupy the leftmost part of the field, another may turn on if the goalkeeper is outside the defence area, another may turn on if 4 attackers assumed a diamond shape, and so on.
While deep learning methods have that generalization potential, it is not guarantee that it alone would be enough: the kind of data used as input plays a fundamental role. As already written in the literature section, the studies of [Copete et al. 2015] and [Zare et Al. 2018] are based on deep neural networks but they used input configurations that were limiting the generalization and making the forecasting very brittle. Since each input neuron was devoted to contain the position data of a fixed player with a certain number on the uniform (in an ordered way, where the order is given by the number of the uniform), if in a game a player was still playing in the same role/zone but using a different uniform number, all the inference would be wrong because it would be referred to another player (to be clear, here the meaning of the word “uniform” is of “shirt”).

We can name this issue “**fixed order input limitation**”. It is a problem related to the way we represent the data into the computer memory. This could be avoided only if the input data were such that the neural network algorithm didn’t need to process the players information in a fixed order, but it could extract that information no matter the uniform number. One example of such a data format could be using an image as input, where the image is a discrete 2D subdivision of the pitch (each pixel representing a little squared subdivision of the pitch area), on which each subdivision element is painted with a certain colour if there is a player of the first team, with a different colour if there is a player of the second team, and with a third colour if there is the ball. In fact, instead of using 3 different colours, it may be easier to use the three independent RGB channels of a picture for the three information types. We will come back to this later, for the moment it is enough to specify that for some experiment it may be useful to first start with a simpler analysis using the fixed ordered input, and in case it gives good results, it may be improved in the next step using an image-like input, since that is more generalizable when switching teams or players order.

2.2 Hypernetworks and Multilevel Systems

Robot soccer is a complex system in which players dynamically build and adapt their behaviour at multiple levels: single player, close collaborating players, full team tactics, and others (including opponents-dependent structures and temporal levels). A way to describe multilevel complex systems is through Hypernetworks, hence they may be proficiently used to discuss about Robot Soccer analysis. Following the work of [Johnson 2013], Hypernetworks are entities that describe relationships between two or more objects. They may be described as Hypergraphs augmented by simplicial edges and an explicit relationship, so we will begin by describing hypergraphs and simplices.

2.2.1 Hypergraphs

A Hypergraph is a generalization of a graph that admits any number of vertices involved in an edge, i.e. relationships between vertices are n-ary instead of just binary like in graphs. Formally, a hypergraph is a set V of vertices and a set E of subsets of V (of any cardinality) called hypergraph edges.

This allows us to describe relationships constituted by any number of objects, while in common graphs only 2 objects (vertices) are involved in a relationship at once.

For instance, a recipe for a cake may be depicted by an edge including all the ingredients: without one of the ingredients the relationships would be incomplete and the relationship would not exist anymore (think about keeping out the flour or the baking powder from the recipe).

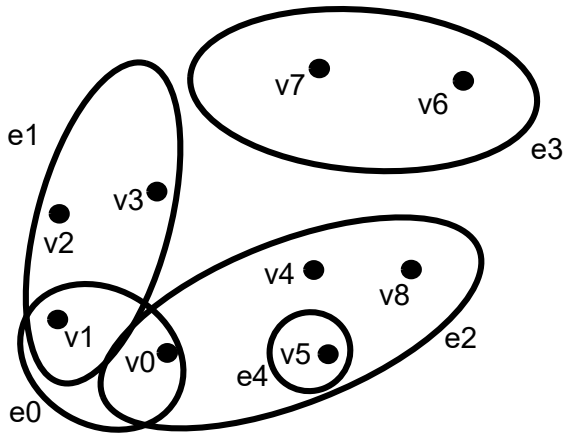


Figure 2.6 : An example of Hypergraph with Edges: e0={v0, v1}, e1={v1, v2, v3}, e2={v0, v4, v5, v8}, e3={v6, v7} , e4={v5}

Every hypergraph has a dual hypergraph where the edges of the original graph are the vertex of the dual graph, and for each vertex of the original graph there is an edge in the dual graph containing all the original graph edges (now vertices) that contained the original vertex.

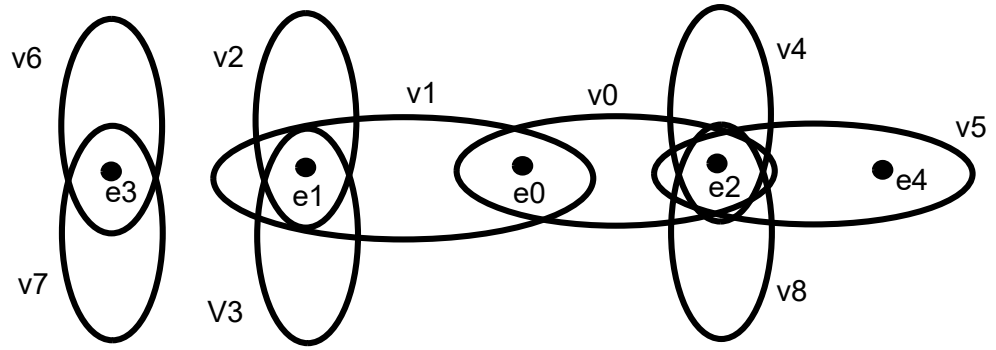


Figure 2.7 : The dual Hypergraph of the graph in figure 5.1

	e0	e1	e2	e3	E4
v0	1	0	1	0	0
v1	1	1	0	0	0
v2	0	1	0	0	0
v3	0	1	0	0	0
v4	0	0	1	0	0
v5	0	0	1	0	1
v6	0	0	0	1	0
v7	0	0	0	1	0
v8	0	0	1	0	0

Table 2.4 : The same Hypergraph of figure 5.1 , but in tabular/matrix form

2.2.2 Simplices and Hypersimplices

Hypernetworks are an extension of hypergraphs where the order of the vertices in an edge matters, and where each edge may describe a different type of relationship. The need to have an ordered set of vertices leads to the use of simplices instead of sets to define edges, and the need to have an explicit relationship leads to augment the simplices with it, obtaining hypersimplices, that are used as edges in hypernetworks.

Attention must be paid to not confuse this meaning of the term “hypernetwork” with the one that originated later in the Deep Learning community by the work of [Ha et al. 2016] , which is

a completely different meaning and signifies a neural network that defines the synaptic weights of another neural network.

From a geometrical standpoint, a simplex is a generalization of the concept of triangle or tetrahedron to any dimension. A simplex with k vertices is called a k-simplex. It may be defined as a k-dimensional polytope, determined by the convex hull of its $k+1$ vertices. We are not interested in the geometric properties of a Simplex, but rather in the fact that it identifies a relationship between its k vertices, that would not exist, not even partially, if a vertex was cancelled, and in the fact that the order of vertices matter (even if it is often disregarded in some geometrical operation, the order of vertices in a simplex must be taken in account in some task such as when dealing with homology, see [Hatcher 2001], p.103). So, we are interested in a simplex as an "ordered set of vertices".

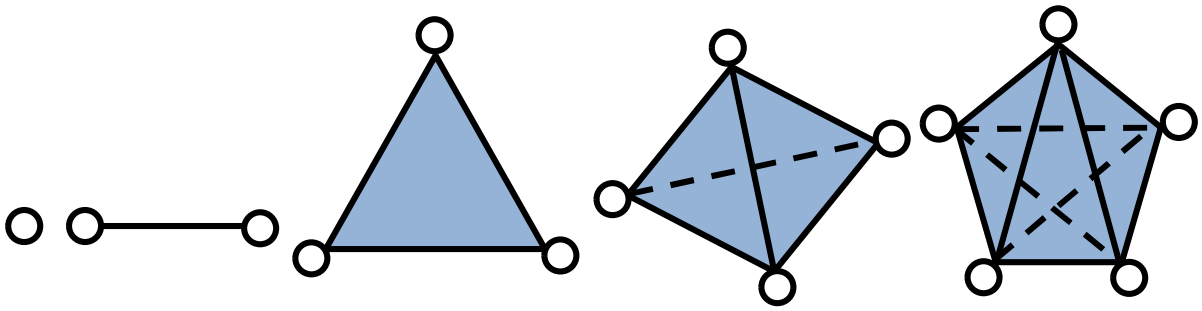


Figure 2.8: From left to right: a 0-simplex (point), a 1-simplex (line), a 2-simplex (triangle), a 3-simplex (tetrahedron), a 4-simplex

With a simplex it is possible to describe an n-ary relationship, and the order of its vertices may incorporate useful information.

The simplex $A = \langle v_0, v_1, \dots, v_q \rangle$ is a q-dimensional face, or q-face, of the simplex $B = \langle v_0, v_1, \dots, v_p \rangle$ if every vertex of A is also a vertex of B (to be noted that q is equal to the total number of vertices of A, minus 1). Two simplices are said "q-near" if they share a q-dimensional face (that is composed of $q+1$ vertices). If two simplices are connected by a q-chain of q-near pairs of simplices, they are said to be "q-connected" [Atkin 1972a]. Doing a "Q-analysis" means finding all the separated q-connected components, for different values of q.

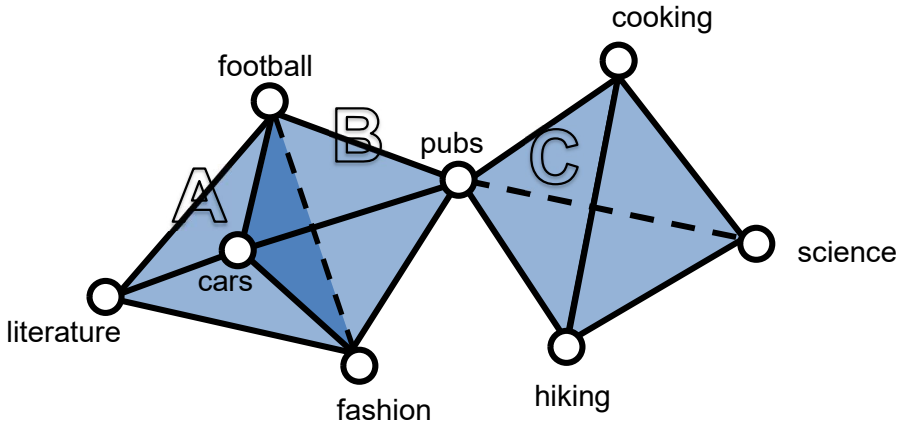


Figure 2.9: Simplices representing the interests of Alice, Bob and Cindy, where each interest is a vertex. Alice's interests are <literature, cars, fashion, football>. Bob's interests are <cars, fashion, football, pubs>. Cindy's interests are <pubs, cooking, hiking, science> . Those simplices in the pictures are respectively indicated by A, B, and C. Alice and Bob are q-connected by 3 interests: <cars, fashion, football> (this simplicial face is in darker color in the picture because is shared). So Alice and Bob are said to be 2-connected. Bob and Cindy instead are q-connected by one interest, <pubs>, so they are 0-connected. Together, Alice, Bob and Cindy form a 0-connected chain. In this example the order of vertices has not been taken in account, but it is easy to think of an example where it is: it is enough to imagine that the interests in each simplex are ranked in order of importance, and that the q-connection is about having the same interest, and in case of more than one, they must match in preference order.

Simplicial vertices may be of numerical continuous type or of categorical type (including Boolean). When they are categorical, the written representation may account for it, representing each possible categorical value occurrence as a different vertex. For a simple example let's think of having the simplest categorical type: the Boolean type.

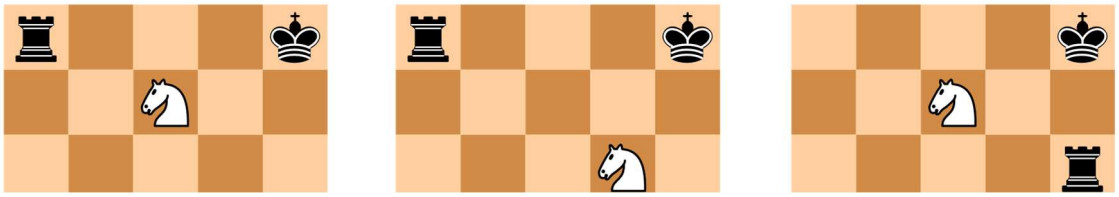
Then a simplex $C = \langle v_0 : true,\ v_1 : true \rangle$ can be written differently from the simplex $D = \langle v_0 : false,\ v_1 : true \rangle$. The former could be written $C = \langle v_0, v_1 \rangle$ and the latter $D = \langle \sim v_0, v_1 \rangle$. This explicitly makes v_0 and $\sim v_0$ as 2 different vertices. Apart from reading clarity, the advantages are that in case of a missing value we don't have neither v_0 nor $\sim v_0$, and so missing a vertex it's not mistakenly evaluated as that vertex being present with a negated value.

Vertices created as negations of other vertices are properly called "antivertices". A set of simplices is called "simplicial family" and it is the generalization of a hypergraph augmented by the order of the vertices in each simplex.

Simplices are often used to express a relationship between their vertices, and the type of relationship is implicit (the same happens when we express a relationship with edges in hypergraphs) but if we want to have a more complete specification we should insert the type

of relationship together with the vertices. A hypersimplex is a simplex augmented by an explicit relationship (about the elements of the simplex). This makes possible also to have different type of relationships described explicitly in the same hypernetwork.

Since a hypersimplex is a simplex augmented by a relationship, it can be written adding the relationship at the end of the simplex, separated by a semicolon, for instance as $H = \langle v_0, v_1, \dots, v_q \,; R_1 \rangle$ where R_1 is the relationship.



(a) <b rook, w knight, b king; F1>

(b) <b rook, w knight, b king; C>

(c) <b rook, w knight, b king; F2>

Figure 2.10: in the game of chess, a configuration where the knight threatens the opponent rook and at the same time puts the opponent king in check is called "knight fork". In case (a) we have a knight fork, but in case (b) the rook is not threatened so it is not a knight fork. In case (c) there is knight fork, but with a different disposition of the pieces, so it is not the same fork as in case (a). These three different configurations can be described by three different hypersimplices: <black rook, white knight, black king; Fork1> for case (a), <black rook, white knight, black king; Check> for case (b), <black rook, white knight, black king; Fork2> for case (c). From [Johnson 2013]

2.2.3 Hypernetworks

Informally we already described hypernetworks as entities that describe n-ary ordered relationships between objects, *i.e.* as generalization of networks with edges involving n vertices at once, and explicit ordered relationships. Now that hypersimplices have been defined, we can formally define a *hypernetwork* as a set of hypersimplices.

One important characteristic of hypernetworks is their possibility to model hierarchies and multilevel dynamics. This is possible in two different ways: the former is by studying the q-dimensional faces of a hypernetwork, looking for entities described by faces that are included in other faces with a bigger dimension (lower level element as a simplicial face of an upper level entity simplex), and the latter is by identifying upper level's entities that are themself hypernetworks whose vertices are current level hypernetworks (lower level element as a vertex of an upper level entity simplex). In other words, hierarchies may be defined in hypernetworks in two ways: (1) describing an upper-level entity as a hypersimplex whose

faces are lower-level entities, or (2) describing an upper-level entity as an hypersimplex whose vertices are lower-level entities. A variety of different tangled hierarchies may be expressed using both at the same time.

An example may be given by a hypothetical modelling of the society from an economical point of view: families may be composed of people, with each family being an hypersimplex whose vertices are people and the relationship is “being a family”. Companies are composed of people and capital, so companies are other types of hypersimplices with both people and capital as vertices, and “business relationship” between them. But inside companies, many levels of hierarchies may exist, both horizontally (different functions of the company) and vertically (different levels inside a unit, different aggregations for each company’s department or unit) each of which may be described by a dedicated hypersimplex, and each hypersimplex may become a vertex or a face for the upper-level hypersimplex. The region or the state will have a political or administrative structure, composed of other hypersimplices and other hierarchies, that interact both with companies and with families.

A vertex may take part in more than one hypersimplex of different type (a person is both a member of a family and may work in a company) or of same type (someone may take part to the boards of directors in more than a company) and of same or different level (the hypersimplex of a business transaction may include a person-customer or company-customer). All this richness of multi-level structure and interaction may be modelled using hypernetworks.

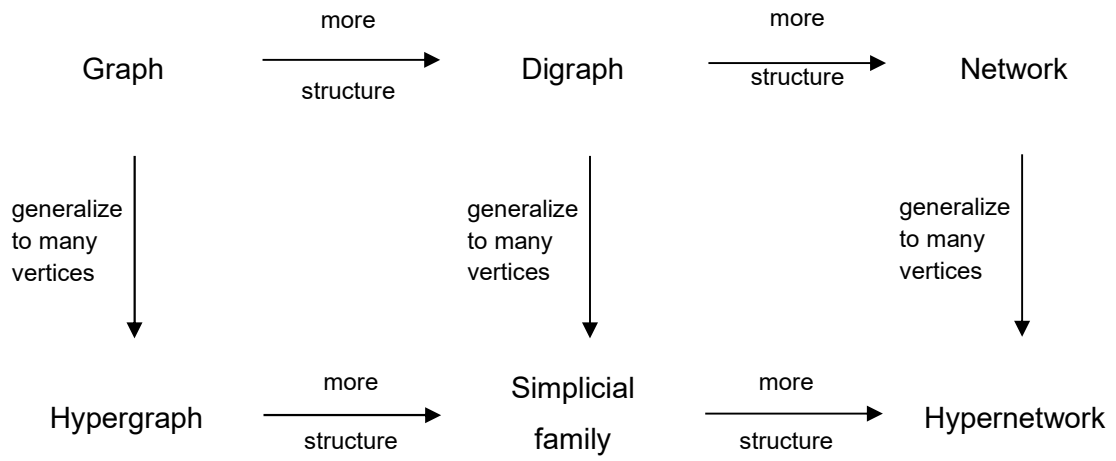


Figure 2.11: a resuming scheme that shows how hypernetworks generalize all common network structures. From [Johnson 2013]

Hypernetworks are formal structures that describe connections between objects, where the connection is not limited to two objects at the same time but may involve more [Johnson 2013]. They can be thought as generalizations of networks where the links/edges are not only dual but instead define a relationship between any number of vertices at once, and where the relationships among vertices may depend on the order of the vertices (hence simplices are used to describe n-ary edges/links).

2.2.4 Hypernetworks and Robot Soccer

As already written, Hypernetworks can be employed to model positions and relationships inside the RoboCup 2D game, as well as in real human football. The position of the players and the ball together with other related variables representing n-ary relations may constitute a Hypernetwork. [Johnson and Iravani 2007] bring as an example a hypothetical situation in which two teams, "Blacks" and "Whites" are playing, and two Whites attackers w_1 (possessing the ball) and w_2 are very close to the goal. Player w_2 is at the left and slightly forward with respect to w_1, and they are facing a Blacks defender (b_1) and the Blacks goalkeeper (b_0). The defender robot b_1 may try to tackle directly w_1, but in doing so he will risk an easy pass from w_1 to w_2, which in that case would have an easy shot at goal. On the other hand, b_1 may want to mark w_2, but in that case w_1 will advance and have an easy shot at goal himself. This is named "the defender's dilemma" (the counterpart of this dilemma, from the standpoint of the attacker, has been described in [Kostiadis and Hu 1999]).

With an hypernetwork formalism, we may think of w_1, w_2, b_0, b_1, R_1 as the composing elements (vertices), and then a certain spatial configuration may represent the relationship R_1 of a hypersimplex, for instance the hypersimplex$h_1 = \langle w_1, w_2, b_0, b_1;\ R_1 \rangle$ may be the initial situation in which w_1 has the ball, w_2 is at his left, and they are facing b_0 and b_1. Then the situation may evolve either in the configuration in which b_1 tackles w_1, having the hypersimplex $h_2 = \langle w_1, w_2, b_0, b_1;\ R_2 \rangle$ or in the configuration in which b_1 marks w_2, having hypersimplex $h_3 = \langle w_1, w_2, b_0, b_1;\ R_3 \rangle$ Hypersimplices h_1 and h_2 in turn, may form an hypothetical upper level hypersimplex $H_a = \langle h_1, h_2 \rangle$ that describes a possible evolution of a situation, while hypersimplices h_1 and h_3 may form the upper level $H_b = \langle h_1, h_3 \rangle$ that describes the other case.

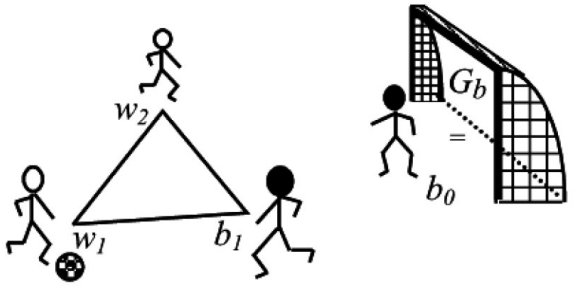


Figure 2.12: The defender dilemma, as a hypernetwork <w_1, w_2, b_1; Rdd>: what would be the best move for defending player b_1 against attackers w_1 and w_2 ? From [Johnson & Iravani 2007]

This is an example of how actions may generate upper-level structures. When we can identify patterns in these upper-level structures, the discovered regularity reveals the emergence of a certain behaviour. This means that, as already anticipated in the introduction, the emergent organization of players may form a "hyper-agent" [Johnson & Iravani 2007], that is an entity with a well-defined behaviour. This behaviour is what we are willing to analyse, and coincides with the actuation of the tactics of the team (even if it may not coincide with the tactics planned by the coach!). Other ways of employing hypernetworks in RoboCup analysis is to build one from the sequence of passages, paying attention to the ones that are successful (as the ones that lead to scoring goals). For instance, if an action that led the Blacks to score a goal was started by player 9 that passed to player 6, who in turn passed to player 5, that passed to player 7, who passed to player 10 who scored a goal, the corresponding hypersimplex could be $h_{goal} = \langle b_9, b_6, b_5, b_7, b_{10}\,; R_{GOA}\,\rangle$. Also the marking relationship may constitute an hypernetwork: if Blacks player 9 was marked by Whites player 1, Blacks player 6 was marked by Whites player 6, Blacks 5 by Whites 7, Blacks 7 by Whites 5, and Blacks 10 by Whites 3, we have the following hypersimplices $h_1 = \langle b_9, w_1;\, R_{Mark}\rangle$, $h_2 = \langle b_6, w_6;\, R_{Mark}\rangle$, $h_3 = \langle b_5, w_7;\, R_{Mark}\rangle$, $h_4 = \langle b_7, w_5;\, R_{Mark}\rangle$, $h_5 = \langle b_{10}, w_3;\, R_{Mark}\rangle$. Then it's possible to put in relation h_{goal} with h_1, h_2, h_3, h_4, h_5 .

Another example about how to analyse a RoboCup game structure with hypernetworks can be found in our previous work [Johnson & Rossi 2018]. In that paper hypersimplices were used to model basic game structures in a multilevel hierarchy. For instance the ball possession by a player may be represented by a hypersimplex containing the coordinates of the player and the ball and a flag identifying if the player was in possession of the ball. This hypersimplex was named "*possession point*". Many possession points of the same player, of subsequent time stamps may be composed together to create a higher level simplex that represents a player dribbling the ball. Two possession points of players of different teams may be composed together to represent the vertices of a *tackle* hypersimplex, and two possession

points of players of the same team may be composed together to represent the vertices of a *pass* hypersimplex.

Another kind of hypersimplex may be created between two players of same or different teams to create a “closest player” hypersimplex, and when two or more such hypersimplices share one vertex (one player) they together form a connected component that may be considered an higher-level hypersimplex . Those hypersimplices may be then composed to create vertices for an even upper level hypersimplex, whose variation in time describes collective moves, and so on: elements existing at level N may be used to model elements at level $N+1$. With such high-level relationships identified, it is possible to notice a change in the game leading to a score when the multilevel structure changes.

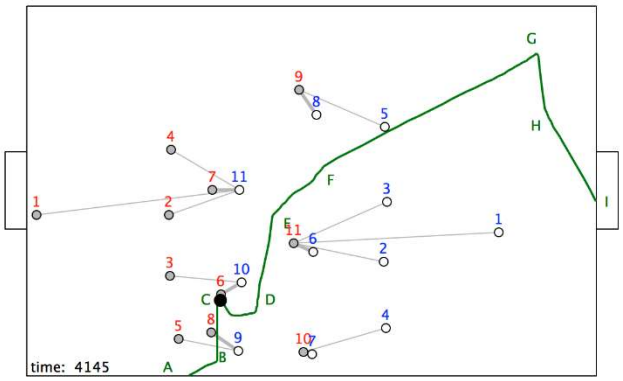


Figure 2.13: hypersimplices of the relationship “closest player” are dual relationships among players, whose representation in the figure is a grey line. Such relationship gives rise to connected components that are clusters of players reachable by a path of grey lines, isolated from players of other clusters. Players of each cluster then are vertices of a higher-level simplex representing the connected component. (The green line is the trajectory of the ball during a timespan, while the player positions are referred to a time tick at t=4145) [Johnson & Rossi 2018]

In our other study [Johnson & Rossi 2020] we used again hypersimplices to conduct an analysis of Robocup 2d games, this time we also used the pitch area occupation as hypersimplices vertices. The pitch was divided in small squares and each square assigned to the closest player. In this way each player would “own” a certain area (this is equivalent to a discretized version of a Voronoi tessellation where players are the generator points). Then each area may be a vertex of a hypersimplex in which the relationship is “be adiacent” to a companion area. In this way, hypersimplices’ connected component would identify portions of pitch under control of the same team. Analyzing the configuration in which these high-level structures form is useful to understand the evolution of the game. In turn, the relationship between different controlled areas forms a further higher-level hypersimplex.

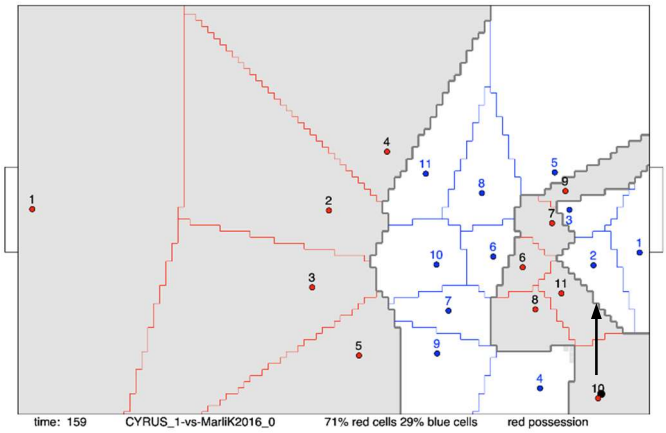


Figure 2.14: controlled areas in a Robocup game. There are 4 major islands, with the red team owning the bigger portion of the pitch, attacking, and owning the ball possession (the ball is the black spot close to red player 10). Notably the red team breaks the opponent team in two parts, surrounding the opponent goal. This high-level configuration, that happens at tick 159, is good for reds (left team) and bad for blues (right team), and in fact this leads to having the red team scoring a goal at tick 180. [Johnson & Rossi 2020]

In both studies information processing and hypersimplices extraction was automated by computation, but a final step of human interpretation was required to understand what configuration was successful and why, so the problem of automatically learn/forecast opponent tactics or automatically learn good or bad configurations is not solved yet.

Similarly, hypernetworks have been used also to analyse a real human soccer game, as in the exploratory study of [Ramos et al. 2017], where the authors used the data of the players positions of an English Premier League team during 5 games. They identified hypersimplices based on the proximity of players of the same team and of the opponent team, finding local configurations such as 1 vs. 1, 2 vs. 1, 2 vs 2, and so on, and calculated spatial statistics about those hypersimplices. Then they observed how those measures and structures were changing during time and after relevant game events. They also identified higher level hypersimplices formed by those low level simplices, with the purpose of studying aspects of the team's emergent collective behaviour. That research showed that a soccer game may be analyzed with hypernetworks in an exploratory way, but still a part of the analysis was not automated and the relevant patterns discovery was arbitrary. A similar approach to use hypernetworks in soccer analysis has been proposed also by [Ribeiro et al. 2019].

2.2.5 Multilevel Hypernetwork Trajectories in Robot Soccer

Robot soccer is a tangled complex system with many levels of structure. At the microlevel are interactions between individual players, as illustrated by the defender's dilemma hypersimplex. A football game can be seen as a sequence of such hypersimplices through

time. This is another higher level hypersimplex whose vertices are the bottom level hypersimplices. In soccer coaches and managers gives sequences of structure like this name such ‘play-37’, as in the figure below.

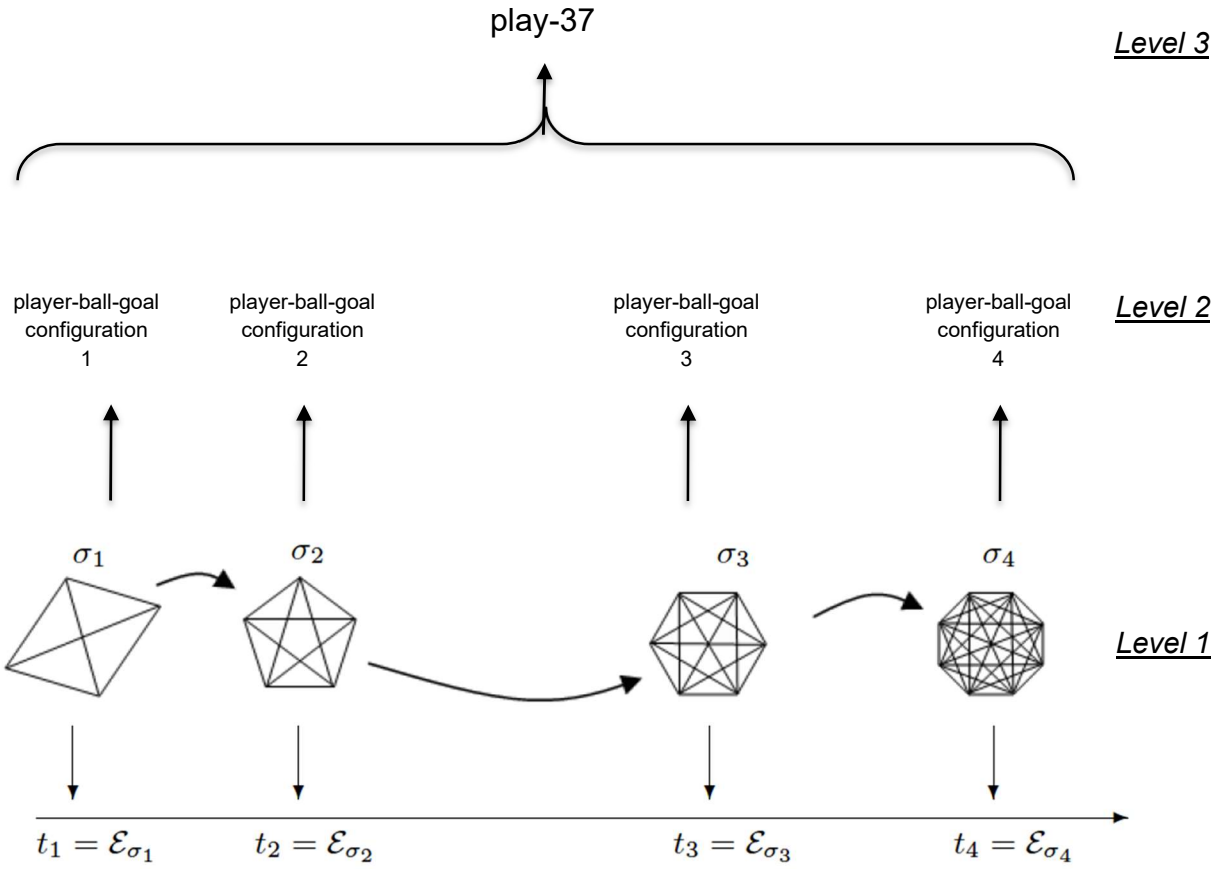


Figure 2.15: A 3-level trajectory in a soccer hypernetwork

Hypersimplices are mathematical entities, and to be recognized in a soccer game, it is necessary to describe algorithmically the relationship part. In fact, while we defined previously the hypersimplex as a simplex augmented by a relationship, that relationship is computationally vague and ambiguous if described by words (natural language) and should be better defined by an algorithm, defining all cases and details, and translating human concepts to computable instructions (the algorithm may also be probabilistic). Gregory Chaitin wrote once "*you understand something only if you can program it*" [Chaitin 2004] and in this case it could not be more appropriate. Once the algorithm is defined, implementing it allows finding hypersimplices in data. One of the challenges of such an approach is to define an algorithm that has a feasible computational complexity. In Chapter 7 we use an algorithm to identify the Defender’s Dilemma hypersimplex.

If we do not know in advance what are the types of hypersimplices that form the structure of the hypernetwork, and so we cannot define them, we may want to use methods that capture relationships in data, such as neural networks, whose internal representation we claim are in fact a form of hypersimplices. We discuss further this idea in the following paragraph, and in Chapter 5.

This thesis is the one of the first contributions to the literature that uses neural networks to compute the relational structure, R, of a hypersimplex < v_0, v_1, …, v_p; R > and to learn classes of hypersimplices from data. [Rucco et al. 2015] used machine learning in a 'neural hypernetwork approach for pulmonary embolism diagnosis', but there the approach was different to that developed in this thesis.

2.2.6 Hypernetworks and Artificial Neural Networks

Hypernetworks (HN) are a theoretical conceptual framework while artificial neural networks (ANN) are not only a theoretical model but also a family of concrete algorithmic methodologies. Anyway, HN and ANN share some common aspect, as suggested in [Johnson 2013] neural networks may be considered as a special kind of hypernetworks.

From a hypernetwork perspective, we could superpose the input configuration of a neural network to the vertices contained in the hypersimplices of a hypernetwork. Then the synaptic weights values may be the equivalent of the relationship part of hypersimplex: for each neuron in the hidden levels that expresses a prototypical property (a "learned feature") we have a certain set of weights that describes how inputs should be in order to turn the property on (in hypernetwork terms that means to confirm the presence of the "relationship"). Then, a second neuronal layer may be regarded as the equivalent of an upper level hypernetwork whose vertices are lower level spatial hypernetworks.

This similarity between HNs and ANNs is evident if we look at the architectural aspect. The hypersimplices constituting a hypernetwork may be transformed to generate a bipartite graph in which the vertices of the hypersimplex are the vertices of one side of the graph, and the hypersimplices themself are the vertices of the other side of the graph, with edges connecting each hypersimplex to its vertices. This resembles the two layers of a neural network. For instance, let's think about the hypersimplex $A = \langle v_0, v_1, v_2 \,; R_1 \rangle$ and the hypersimplex $B = \langle v_1, v_3, v_4 \,; R_2 \rangle$. The generated bipartite graph would be like in Figure 4.11:

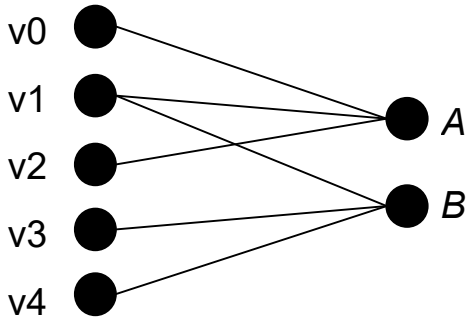


Figure 2.16: bipartite graph derived from two hypersimplices A=<v0, v1, v2, R1> and B=<v1, v3, v4, R2>

With that kind of representation, it's even more clear how ANN's input neurons are comparable to HN's vertices, ANN's synaptic weights comparable to HN's relationships definitions, and the ANN's learned features neurons comparable to the existence of HN's hypersimplices. In fact in a HN the input data is represented by vertices, and relationships describe a logic for the existence of the hypersimplices. Parallelly in an ANN the input data fits into the input neurons and the synaptic weights and the connectivity describe an algorithm that reveals the existence of a certain property. There is not a perfect 1:1 overlap anyway: for some kind of HN's relationships, as when some vertices are required to not be included in the relationship, the equivalent logic would entail applying "xor" operations in the corresponding neural network, for that reason the equivalent neural network would need an additional middle layer since just two layers are not enough to process xor operations (because solutions to xor operations are not linearly separable, see [Mitchell 1997]).

Given this similarity between hypernetworks and neural networks an interesting question is if computationally or representationally one may be equivalent to the other, at least for some type of HN an NN. In other terms the question could be "does using a deep learning model entail that there exists an implicit correspondent hypernetwork" ? As we will see in the chapter about neural networks, it seems realistic to think that neural networks are learning an internal representation that may be thought as equivalent to a hypernetwork.

2.2.7 A Note on Hypernetworks Terminology and Definitions

We are aware that in Network Science and Graph Theory, a variety of different definitions exist for some concepts, sometimes with contrasting or confusing meaning. Part of this is because Network Science and Graph Theory researchers are separate communities with different

purposes. Simplifying, in graph theory the aim is to find rigorous mathematical proofs on graphs' properties, in network science the aim is to find mathematical ways to deal with real word data that shows relationships expressible by connections. There is little crossover between the two scientific communities, and they use different terminology even if the underlying mathematical objects may be the same [Iñiguez et al 2020] (there is another subtle difference in the subject: the graph is supposed to be the mathematical concept, the network is supposed to be any real-world instance of the graph concept. But thinking more carefully, the network itself is a concept, when you see an "instance" of a graph it is in fact also an "instance" of a network, while the real thing is something else, you call it "network" because you are inspired by the concept).

One first example of the difference in terminologies: graphs are composed of vertices and edges, whose equivalents in networks are nodes and links, even if sometimes the terms vertices and edges are used also in Network Science studies, with the same meaning. Both in Graph Theory and Network Science community, it is possible to find the word "arc" to mean a directed edge/link. This words correspondence is used very frequently, but some use instead a different meaning for "network": they define a network as a graph with values associated to edges, following (partially) the definition of the seminal work of [Harary 1969] (p.52): "A network N may be regarded as a graph or directed graph together with a function which assigns a positive real number to each line.".

But, in contrast, graphs with values associated to the edges are also commonly called "weighted graphs".

[De Nooy et al. 2018] have a similar definition but are more general on the nature of supplementary information to have a network from a graph:" A network consists of a graph and additional information on the vertices or the lines of the graph." (p.8).
Some define a network as a directed graph with values associated to vertices and edges, such in [Johnson 2013] (p. 151).

In [Berge 1973] (p.3), graphs are composed of vertices and "arcs" (or "arrows"), where arcs have a direction, so to Berge graphs are what is currently usually called instead directed graphs or digraphs. Berge calls instead "multigraphs" or "undirected graphs" what are currently commonly called graphs (with no directions on the edges) (p.5).
Unexpectedly, when Berge defines hypergraphs (p. 389), he does not use the word "arc" but "edge", and he does not specify any kind of direction for edges, so despite considering directed graphs as "default graphs", he considers undirected hypergraphs as "default hypergraphs".

In fact, while with a dual relationship it is easy to think of a direction as a something that goes from a node/vertex to the other, when the relationship involves 3 or more nodes it is not obvious what a “direction” means. For [Johnson 2013] the direction is the order of the vertices in the simplicial edge (the same meaning that we intended in our chapter).
[Gallo et al. 1993] instead gave a different and peculiar interpretation of what “direction” is in “directed hypergraphs”, creating a definition of “hyperarc” that is a big change with respect to the concept of edge in undirected hypergraphs: a hyperarch would identify a set of vertices called “Tail” from which the path departs, and a set of vertices called “Head” to which the path arrives. In other words, this concept of “hyperarch” is equivalent to a directed subgraph, in which relationships are dyadic (not n-ary, not “hyper”) and all nodes in the tail are connected to all nodes in the head with a dyadic, directed, edge (this is evident when in the paper the authors define the path in the hypergraph and it turns out that a path may pass along even just one of the tail nodes and just one of the head nodes). This is very different from the idea, in hypergraphs, of n-ary relationships, because the background idea of hypergraph would be to have relationships involving more than 2 nodes, a kind of relationships different from a set of relationships involving only 2 nodes. All directed hypergraphs following the definition of [Gallo et al. 1993] can be transformed in directed graphs (not “hyper”) substituting each hyperarc with the corresponding set of dyadic edges, while undirected hypergraphs cannot be transformed in undirected graphs without changing the meaning or losing information.

For this reason we do not find useful using the definition of directed hypergraph by [Gallo et al. 1993] and we use instead the one based on simplices and the order of their vertices.
Nonetheless, the definition by [Gallo et al. 1993] is frequently found in studies by other authors, and can still be considered a generalization of directed graph, since any such “directed hypergraph” where all tails and all heads of the arcs have cardinality 1 is also a directed graph.
A similar concept, where the “Head” was constituted by only one node, was in the proposal of directed hypergraph by [Ausiello et al. 1986], again closer to a concept of dyadic subgraph than to a true n-ary relationship.

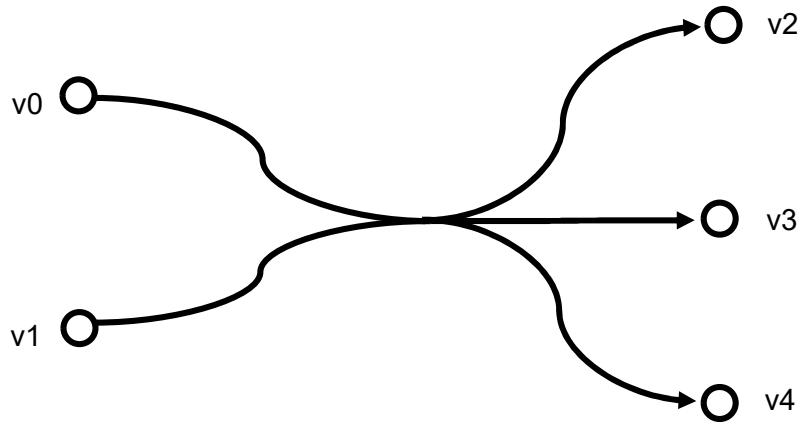


Figure 2.17 : A hyperarc in the definition of [Gallo et al. 1993], with tail vertices {v0, v1} and head vertices {v2, v3, v4}

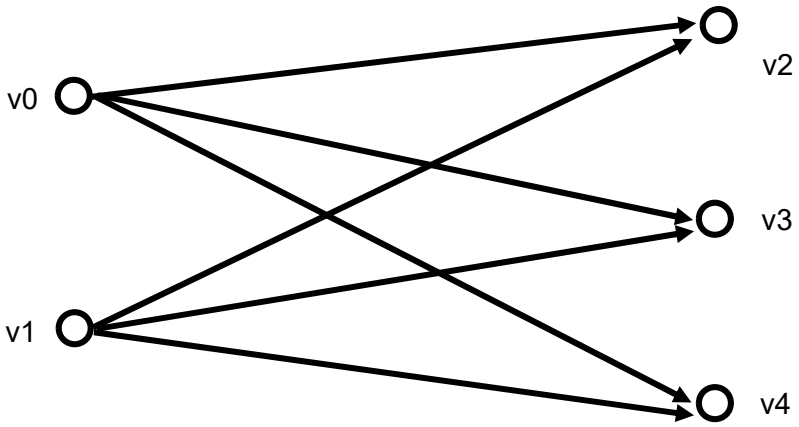


Figure 2.18: The hyperarc, as in the definition of [Gallo et al. 1993], of previous figure, transformed into the equivalent directed subgraph

Also the term “Hypernetwork” has been used by different authors with different meanings, in different fields, such as in [Volpentesta 2008] and [Pretolani 2013] where a hypernetwork would be a subcomponent of a hypergraph with certain defined characteristics (a concept very different from ours), or in [Ha et al. 2016] where a hypernetwork would be a neural network that generates synaptic weights for another neural network (again, nothing to do with our definition), or in [Sorrentino 2012] where the concept is related to the coupling of dynamical systems (definitely unrelated to our definition).

Since there is all this variability in the definition of terms depending to each author, we specify that in our research we are following the definitions of [Johnson 2013], as we presented in the chapter.

2.3 Deep Learning

As the literature review evidenced, the methods that are able to capture game patterns with a good level of generalization are the ones based on Machine Learning, more specifically using Artificial Neural Networks. Let us briefly introduce the part of those topics that we use in our research.

2.3.1 Supervised learning

Consider having data points of structure $\langle x: y \rangle$ where x is a part we name "features" and is given for each element, while y, named "label", is a part that is given for training data but is missing in operative data, and is what we want to guess or forecast, given x. The problem of supervised learning is to find a function $f(x)$ that can guess y , with as little error as possible.

In mathematical language, modifying and simplifying the formalization of [Vapnik 2000], we may define the problem as follows.

Let's assume that each $\langle x: y \rangle$ data point is an i.i.d. sample distributed by probability density function $P(x, y)$, this entails that it exists a pdf $P(x)$ and that a hypothetical conditional distribution with pdf $P(y|x)$, named "supervisor", exists.

We use $f(x)$ to define our supervised learning function, the one that tries to guess/forecast y. Then we may define the error on the prediction of a sample $\langle x: y \rangle$ as a "Loss function" $L(y, f(x))$, sometimes also known as "cost function".

It is possible to define many different forms of loss functions, depending on the type of problem (classification or regression) and on the type of algorithm intended to be used. For instance, in regression problems it is often used the squared error:

$$L = (y - f(x))^2$$

Equation 2.1

So, minimizing the forecast error L, for samples distributed with pdf $P(x, y)$, means minimizing the Risk functional, that is the expected forecast error:

$$R = \int_{x,y} L(y, f(x)) * P(x, y)\, dx\, dy$$

Equation 2.2

2.3.2. Artificial Neural Networks

There are many different Supervised Learning algorithms, each with its strengths and weaknesses. Guided by the literature review findings, in our work we use Artificial Neural Networks, whose general description can be found in [Goodfellow et al. 2016].

In a Feedforward neural network, also named Multilayer Perceptrons (MLP), neurons are numerical variables organized in layers, with directional connections going from a layer to the one immediately above. In some model there are also skip-connections, that are connections that do not direct towards the layer immediately above but to some upper layer (still, the direction is “forward”).

The first layer of the network is composed of input neurons, that are just memory cells that receive the data from samples of the dataset, what we called x . The last layer of the network is composed of output neurons that will contain the values considered to be the response of the network. Internal layers of the network, those that are not input nor output layers, are called “hidden layers”. In the Supervised Learning formalization written some paragraph above, we would have one final output neuron whose value would be the value of $f(x)$, that we may indicate also with the letter $\hat{y}$, that is the forecast of y, in other words the approximation of true y.

Each neuron receives a signal from each of its afferent neurons (the ones in the layer below), multiplies each signal by a certain weight (different for each receiving neuron and for each afferent neuron) and all those multiplications are then summed together. To that number it’s also summed a special number named “bias”, different for each receiving neuron, that in the biological metaphor simulates the adaptation of the firing threshold of biological neuron.

The resulting number is passed as a parameter to an activation function (that may be possibly be just an identity function), whose result will be the final value of the receiving neuron, named “activation value” or just “activation”.

To formalize it mathematically we will call the value (activation) of a neuron as $a_{k,n}$ where k indicates the layer and n indicates the number of the neuron within that layer. The synaptic weight will be $w_{n,m}^{k}$ where k indicates the layer of the emitting neuron (hence the receiving neuron will be at level $k+1$), n is the number of the emitting neuron among those in its layer, and m is the number of receiving neuron among those in its (upper) layer. The bias will be

$b_{k,m}$ with k standing for the layer of sending neurons and m the number of receiving neuron in the $k+1$ layer) . We may use capital sigma Σ for summation of terms, the lower-case sigma σ for the activation function (that may be a sigmoid function such as logistic function or a hyperbolic tangent, but may be also something else, as we will see later with Rectified Linear Units). So:

$$a_{k+1,m} = \sigma(a_{k,0} * w^k_{0,m} + a_{k,1} * w^k_{1,m} + \cdots + a_{k,n} * w^k_{n,m} + b_{k,m})$$

Equation 2.3

To have a very simple example, let's make a graphic representation of the first neuron (the number 0) at layer 1 processing signals from 3 neurons and bias at layer 0.

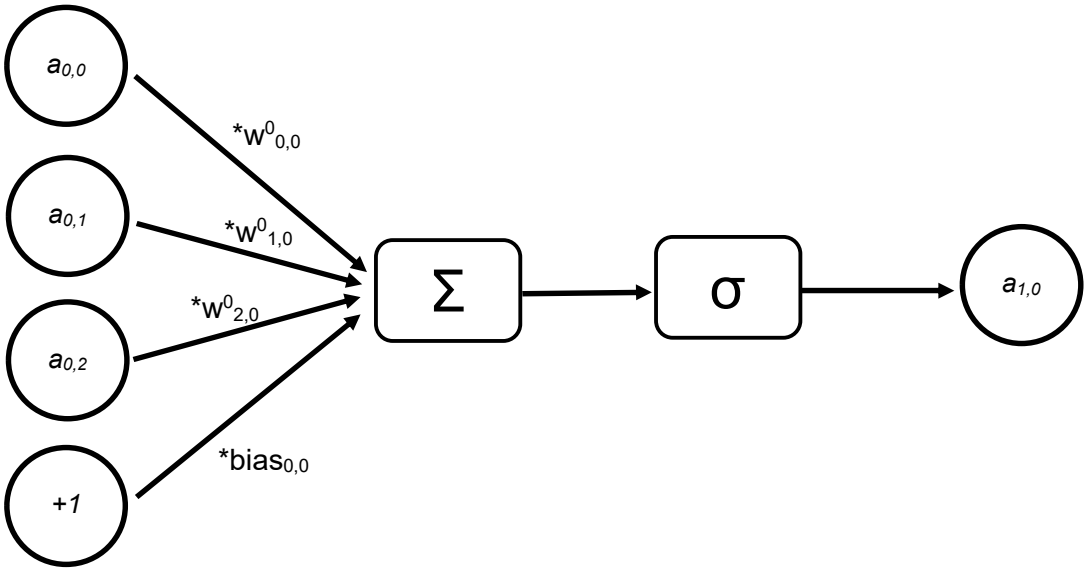


Figure 2.19 A scheme of 3 neurons and the bias sending their signals to the upper neuron

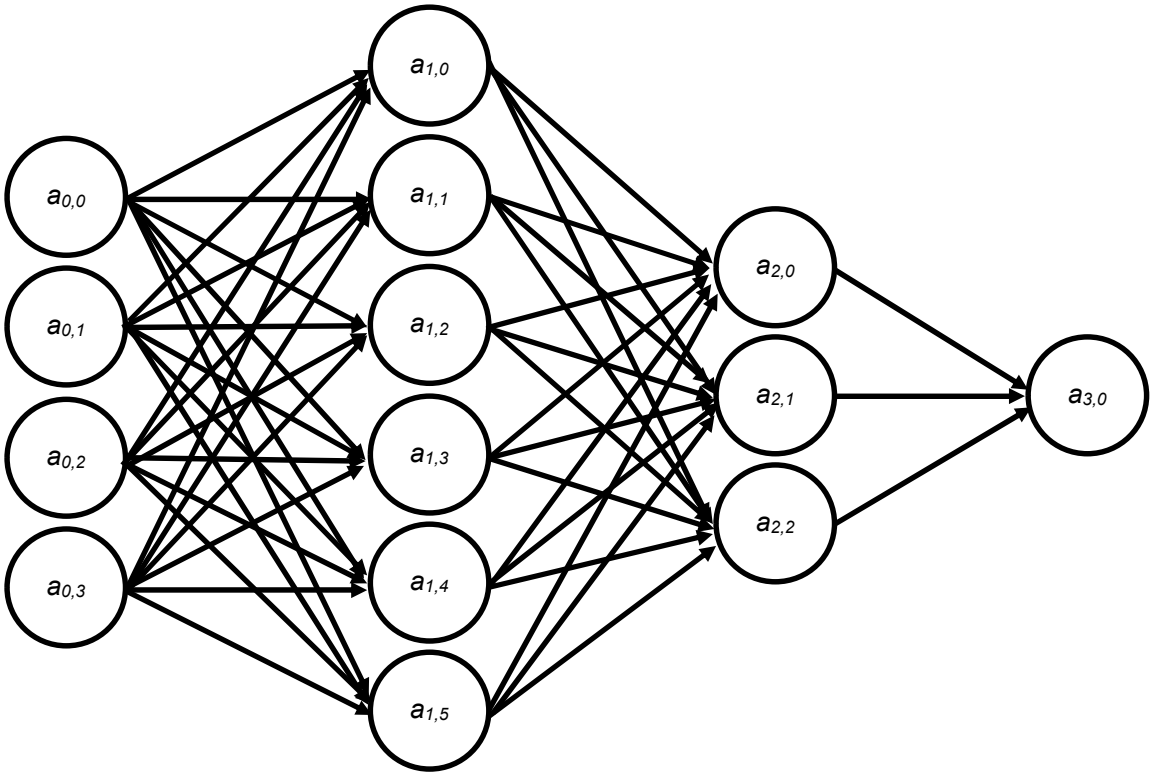


Figure 2.20 A scheme of a 4-layers neural network (bias and operations' boxes are omitted for clarity) : one input layer with 4 neurons, one hidden layer with 6 neurons, another hidden layer with 3 neurons, one output layer with one

Those operations may be also written in matrix notation. Let's imagine having all the activations of a layer in a row vector a_k (with an additional element equal to 1 appended as last element of the vector, to act as bias multiplier) and weights (including bias) for connections from layer k to layer $k+1$ will form the matrix W^k. Activation function may be different at each layer so we may indicate it with σ_k where k is the layer, and we apply it elementwise.

$$a_{k+1} = \sigma_k(a_k * W^k)$$

Equation 2.4

The activation function σ is particularly important because it introduces non-linearities in the processing allowing the neural network to express complicated functions (see [Picton 1994] Ch.3). A peculiar activation function to introduce non-linearities is the Rectified Linear Unit (RELU) [Glorot et al. 2011]. It consists just in a transformation that, calling z its input, returns z if z is positive, otherwise it returns zero.

What we have seen until now is an exemplification of Feedforward networks, but architectures may vary and include additional operations, such as “Drop-out” [Srivastava et al. 2014] that consists in avoiding to use a certain quantity of neurons during training to make the network more robust to noise, and “Batch-normalization” [Ioffe and Szegedy 2015] that is used to improve learning performance through normalization and rescaling of the input of a layer during training.

Additional operations may also be added after the last layer of a neural network, as in the case of Softmax [Hastie et al. 2009, Ch.11] that is used to turn the activation values of more than a neuron, in a same number of probabilities, to be used in classification. If we have C terminal neurons (because we have C classes), if each neuron i has value z_i (called “logit”), applying the Softmax means calculating the following value p_i for each neuron i:

$$p_i = \frac{e^{z_i}}{\sum_{c=1}^{C} e^{z_c}}$$

Equation 2.5

Sometimes in literature it is possible to find that formula also called Boltzmann distribution or Gibbs distribution, from the usage in physics.

In case the classes are only two, or the label is a binary value like true/false, it is possible to have only one final neuron, whose value we may call z . The Softmax will then be simplified into a logistic function, whose output expresses the probability of one class (or “true”):

$$p(z) = \frac{1}{1 + e^{-z}} = \frac{e^{z}}{e^{z} + 1}$$

Equation 2.6

2.3.3 Training

Learning means finding the best weights such that the network can make the smallest possible error in forecasting the Y values. It’s the problem of minimizing the Risk functional R as described above in equation 2.2 . Since we don’t know the exact distribution of data $\langle x : y \rangle$ we can’t minimize the Risk (the expected Loss), but since we know that the data that we are using are a sample of that distribution, we can minimize the Empirical Risk (whose expectation tends

to the Risk), that is the average of the Loss function [Vapnik 2000]. It's an optimization problem in which the parameters are the network weights.

Mathematically, we may define our neural network as a function

$$\hat{y} = f(x)$$

Equation 2.7

we may also want to include the weights $w_{n,m}^{k}$ in the equation: each layer has a matrix of weights, and stacking all matrices forms a three-dimensional tensor that we may call W:

$$\hat{y} = f(x, W)$$

Equation 2.8

We define a Loss function as a function of the label y, the input x , the weights W, and implicitly depending on the architecture of $f(x)$. We may write it in different ways relatively to the form we want to emphasize.

$$L(y, x, W) = L\big(y, f(x, W)\big) = L(y, \hat{y})$$

Equation 2.9

Among the many optimization methods, the most used one, that seems to be the best performer in Feedforward networks (as well as in other ANNs architectures), is the Gradient Descent. It consists of increasing parameters (weights) in little steps towards the direction opposite to the gradient of the Loss.

We start by initializing the neural network with some small random weights to differentiate the role of each neuron among the others (more principled initialization procedures may be found in [Glorot and Bengio 2010] or [Saxe et al. 2013]).
Then we process our samples through the neural network, that means that we put each sample x as value for the input neurons, we do the feedforward propagation, and so we obtain an output value $\hat{y}$ that is the forecast or guess.
Then we may calculate the gradient of the Loss function for that sample with respect to each weight. If we infinitesimally increase each weight in the opposite direction of the gradient, we will end up with a neural network with a smaller Loss than before.

Hence at each calculation step we will add to each weight a small increase proportional to the negative of the gradient of the Loss. To decide "how much small" we will introduce a constant named "learning rate" that will multiply the negative of the gradient, using the symbol η .
So if the weight of the connection from neuron n of layer k to neuron m of layer $k+1$ at time t is indicated with $w_{n,m}^{k,t}$ and the gradient of the Loss function with respect to it is indicated with $\nabla L(w_{n,m}^{k,t})$, the update rule of a weight will be:

$$w_{n,m}^{k,t+1} = w_{n,m}^{k,t} - \eta * \nabla L(w_{n,m}^{k,t})$$

Equation 2.10

We cyclically process all samples through the network, modifying a little all weights after each sample (stochastic gradient descent) or after small batches of samples (batch gradient descent). When all samples have been fed to the network, an "epoch" passed.

There are some algorithms that may be added to the learning mechanics to dynamically adapt the learning rates of weights, with an individual learning rate for each weight, such as AdaGrad [Duchi et al. 2011] and RMSProp [Tieleman and Hinton 2012].

Other algorithms don't just adapt the learning rate but use adaptive moments to modify the weights: in the Adam algorithm [Kingma and Ba 2014] the quantity subtracted to the weight is not directly the gradient of the Loss multiplied by the learning rate, but instead it is a momentum depending on the gradient and past gradients. This makes the learning more stable without abrupt direction changes of the values to be subtracted/added to the weights. A version with regularized weights is AdamW [Loshchilov & Hutter 2019].

2.3.4 Loss Functions

For what concerns Loss functions, if the neural network is outputting a single real number, the Loss functions is commonly the squared error, as in equation 2.1 . If instead the neural network is a classifier with a number of C classes with a final Softmax, that outputs probabilities p_i , one for each class i , the correct Loss function is the so-called "Cross-Entropy Loss" [Bishop 2006 Ch.4]. Calling the labels t_i , with $t_i = 1$ if it i is the correct class for the corresponding sample, otherwise $t_i = 0$,:

$$L = -\sum_{i}^{C} t_i * \log(p_i)$$

Equation 2.11

When the classes are only two, or the label is a binary value, as we saw above in equation 2.6 the Softmax will be simplified into a logistic function. In that case the output of the neural network will be $\hat{y} = p(z)$ and the true value of the label will be a binary value y . The Cross-Entropy Loss then simplifies in the following way:

$$L = -y * \log(\hat{y}) - (1 - y) * \log(1 - \hat{y})$$

Equation 2.12

The gradient of the Loss function is computed using calculus: to compute the gradient of the Loss function with respect to a term of any function it's enough to use the chain rule of calculus:

If

$$y = g(x) \quad and \quad z = f\big(g(x)\big) = f(y)$$

then

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

Equation 2.13

In case x and y are not scalars, that is if $x \in \mathbb{R}^n$, $y \in \mathbb{R}^m$, $z \in \mathbb{R}$:

$$\frac{\partial z}{\partial x_i} = \sum_{j} \frac{\partial z}{\partial y_j}\frac{\partial y_j}{\partial x_i}$$

Equation 2.14

So, applying the chain rule of calculus through all the layers of the network it is possible to compute the gradient of the Loss function with respect to each parameter, i.e. with respect to each network weight.

One algorithm that does it in a fast way is the Backpropagation [Rumelhart et al. 1986], where during the forward propagation phase a graph is built with all the variables and the relative

operations, from the bottom of the network until the end. Then it is possible to calculate the Loss gradient with respect to the last variables, and following the graph backwards it is possible to compute the gradient with respect to other variables/parameters (weights) at previous levels, going back down to the bottom level, reusing the parts of computations done until then. The name is due to the fact that the gradient is “propagated back”. Current neural networks libraries can compute the backpropagation thanks to automatic differentiation functionalities.

2.3.7 Deep Learning

The term Deep Learning is currently used to call the research field and the methodology of ANNs, inspired by the fact that ANNs leveraged great performance and successful task solving when the architectures became deeper, stacking a big number of neuronal layers to process information: it has been shown that deep networks performed better than shallow networks with the same number of parameters [Bengio et al. 2007].

That was possible by the increase of cheap parallel computing power given by modern GPUs. So Deep Learning is in fact a synonym for Artificial Neural Networks, or if we want to be more precise, it describes the modern ANNs architectures and algorithms.

Neural networks find meaningful relationships in data and encode them in weight vectors, [Browne et al. 2003]. When a trained network processes data it transforms input values into a meaningful internal state of neural activations, basing on those learnt relationships stored in the weights. This internal state in literature is known as “internal representations”, “distributed patterns of activity”, “distributed representations”, “hidden units features” (to not be confused with the input data structure that is also known as “features”) [Rumelhart et al. 1986], “vector representations” [Mikolov et al. 2013], “learned distributed feature vector” [Bengio et al. 2003] , “learned high-level feature representations” [Salakhutdinov 2015], “embedding” [Goodfellow et. al 2016 Ch.14], or just “representation” [Goodfellow et. al 2016 Ch.15].

What is peculiar in deep network architectures is the emergence of the capability to combine features of a layer to create higher-level features in the layer above.

This allows the network to recognize patterns at different scales and to capture complex “concepts”.

One example of internal representation that is able to capture concepts is given by the work of [Mikolov et al. 2013]. They implemented specialized self-supervised models of neural networks (informally named “word2vec”) that were able to attribute to each word of a vocabulary (composed of 1 million words) an internal representation that is consistent in many ways. For instance not only the internal representation (that is a vector of neural activations) of words that are semantically similar is also close from a vectorial point of view, but it is possible to do math operations on those vectors and obtain consistent results, as is the case of the operation that subtracts the internal representation of the word “France” to the internal representation of the word “Paris” and then sums the internal representation of the word “Italy” to obtain a vectorial representation that may be reconducted to the word “Rome”.

Another example of compositionality of internal representation is the one in Convolutional Networks [LeCun et al. 2010]. Convolutional networks are models that take advantage of the structure of data and use neurons in a similar way to a convolutional filter. In 2D Convolutional Networks, the input data are 2D images and the purpose of the network is to classify the image into a particular class, or to find a particular class of object inside the image (other purposes are possible, those are just the more used). The peculiar 2D structure of images is favorable to the possibility of processing the whole picture doing piecewise analysis in small squared areas of the picture. In convolutional layers each neuron of the layer sending the signal is not connected to all neurons of the upper layer receiving the signal, but it is connected only to some, that geometrically are considered close in horizontal and vertical coordinates. In other words, every upper layer neuron activation is computed only taking in account the activation of a square of underlying neurons under it, without being affected by the activations of neurons outside that square (the square is commonly 3, 5 or 7 neurons width). The weights of the connections are shared: each receiving neuron multiplies the activation of the neurons under him with the same weights (weights are relative to the position of the input neuron within the square). In this way, weights are like a squared convolutional filter that is applied piecewise to the whole 2D layer of neurons, moving it horizontally and vertically, whose output creates another 2D layer. Sharing the set of weights among each receiving neuron makes sense because it corresponds to apply the same type of analysis everywhere, and it drastically reduces the number of parameters to learn, that causes faster and improved learning.
Each filter, through the training, specializes in recognizing a certain pattern, that is called “feature”. For this reason, the layer of neurons created by such a filter are called “feature maps”, and usually there are many parallel maps for each network level.

Then there are other types of layers such as “pooling layers” where the data resolution is decreased (so to be able to process the relationships inside a bigger area) and relevant

information retrieved in previous layers is retained. Other types of layers may be present inside a Convolutional Network, such as non-linear operators, fully connected layers and so on, but we will not present a complete description of the architecture and its mechanics, leaving the reader to the literature. What we want to focus on is the way in which 2D Convolutional Networks create internal representations. It turns out that Convolutional Networks are able to recognize certain basic shapes and features in the lowest level, and use them to recognize more complex shapes at the next level, and generally at each upper level the features detected in the underlying level are combined to detect a more complex pattern, corresponding to a more specific object recognition: [LeCun et. al 2010] explicitly call them “multi-level hierarchies of features”. An illustrative example is the research of [Zeiler and Fergus 2014], who trained a convolutional network with the Imagenet 2012 data (a dataset containing 1.3 million images, containing objects of 1000 different classes). They devised a method to discover what input pattern in the image caused a given activation in the feature maps. Thanks to that method, they showed that filters at the lowest level capture very basic features such as small horizontal, vertical, diagonal segments and color gradients. Then at the next level those low-level features are combined to recognize more articulated features such as curved lines, circles, corners, squares, eye-like silhouettes, interleaved segments patterns. At third level, the activations of the second level features make possible to detect even more complex objects like geometrically shaped grids, car wheels, human faces and chests, intersecting segments, printed characters. At fourth level the detected features include concentric circles, dog faces, animal paws, cars’ details, edges of cylindrical objects. At fifth level again dog faces, human faces, labels and printed characters, flowers, bicycle wheels, car wheels, animal eyes (it seems that upper layers sometimes continue to detect some graphical shape already detected in previous layers but in the upper layer the feature has more invariance, such rotation invariance or scale invariance).

These examples show how neural networks’ internal representations have the characteristic of encoding knowledge in a way that is consistent, composable and hierarchical.

Once a neural network has been trained to serve a certain task, it can be partially reused for other tasks. This is possible because of the internal representation: the lower levels of the network transform the input data in a consistent internal representation that discover the features in data that are useful for the final purpose. But those features may be useful also for other purposes so it would make sense to reuse the feature-discovering levels also in a different network with different aims. A simple example would be to train a convolutional network to classify dog races, and then reuse the weights of the first three levels as starting weights for the first three layers of another convolutional network whose aim is to classify cat

races, since the first three levels would have the capabilities to discover segments, circles, eyes, legs shapes and so on. The other upper levels, those who discover the most composited features and discriminate the cat races instead would be initialized with new (small) random weights and trained, keeping fixed the weights of the reused lower levels. Then after the network has a good ability in the new task, also the lower levels of the network may be unlocked and fine-tuned with all the other weights in the final training phases. This procedure is called “transfer learning” and allows the knowledge gained from training in one task to be reused into a new one.

2.3.8 The Manifold Hypothesis

The internal representation of a neural network is condensing relevant information of the input data inside the neuron’s activations. This sometimes may be thought as a compression of the original information. When an internal layer of a network has less units than the input layer, the information contained in that layer is necessarily compressed with respect to the input data. A particular class of unsupervised and self-supervised models, the autoencoders, have the purpose of outputting a signal as similar as possible as the input signal, while restricting the information bandwidth in the process. That means that autoencoders have at least an internal layer with less units than input layer, and then the reconstructed signal will have the same size as the original input. Usually there are many layers that progressively reduce the size as in a funnel, and then other layers that progressively increase the size back to original. In autoencoders, the layer with the smallest size contains a “compressed” version of the data, that may be decompressed by next layers to transform it back to original size. If the encoding/decoding of the original data works well, i.e., the reconstructed signals are sufficiently similar to the originals, it means that the compressed internal representation is able to capture the important characteristics of data and filter out unimportant ones. It may seem that inside the greater dimensionality of the original data the meaningful information resides in a “space” of smaller dimensionality, and that neural networks are able to identify it.

Autoencoders makes this clear because they always have an internal layer smaller in size than the input layer, but the same happens in Feedforward Neural networks and in other models of neural networks.

Among Deep Learning scientific community this inspired the “Manifold Hypothesis”. In a first, informal and intuitive (and not mathematically correct) definition, a Manifold may be thought

as a “possibly curved space” that locally has a lower dimensionality than the space in which it lays. If you write the letter “C” on a paper, the paper is a 2-dimensional space, but the letter C internally is a one-dimensional manifold: you may position at one end of the curved segment, and moving only in one direction reach the other end, and internally you would move only forward or backward to reach a point inside the letter, so it locally has only one dimension. Hence, noticing that neural networks can compress the input data into a smaller dimensional representation through non-linear transformations, the parallel with a Manifold is evident.
A more correct and general definition of Manifold, still informal and mathematically incomplete, is the one from Ch. 5 of [Goodfellow et al. 2016]: “*A manifold is a connected region. Mathematically, it is a set of points associated with a neighborhood around each point. From any given point, the manifold locally appears to be a Euclidean space*.”

Hence, compressing the input data is not the only hint that suggests Manifold learning: studying the properties of the internal representation revealed that it may behave as a connected region in which points have a neighborhood belonging to the same region. For instance, let’s look at the generative network named Variational Autoencoder [Kingma and Welling 2014]. That architecture can generate data that is similar to the training samples: it generates synthetic data belonging to the same domain as input data. It learns to represent internally the relevant features of sample data, called “latent variables”. Then, the generation of a synthetic data point starts with sampling the latent variables from a distribution and subsequently transform them by next layers into the final synthetic output. Modifying slightly the latent variables values usually creates a slightly different output, similar but with noticeable difference, still belonging to the same data domain. This suggests that the internal representation of the latent variables has “neighborhoods” and behaves as a connected region. An example is the generation of human faces images by a Variational Autoencoder trained on the dataset “Frey Faces”: changing the value of the latent variables will generate different expressions and pose of the generated face.

Figure 5.3 . [Kingma and Welling 2014]: the faces generated by a Variational Autoencoder trained on the "Frey Faces" dataset. Each face is generated from a slightly different set of latent variables. Two latent variables have been progressively modified to generate different faces: horizontally the change in the first latent variable generates different poses of the face, vertically the change of the second latent variable generates the mouth expression, from contracted to smiling.

A small dimensional set of latent variables contains the information for the meaningful output in a much greater dimension (the pixels of the image), and a small change in a point in latent space will result in a neighbor point in output space. This is the kind of behavior that we would expect if the internal representation would capture a manifold of the sample data space.
Two more formal definitions of manifold may be found in Topology and Algebraic Topology textbooks:

"*An m-manifold is a Hausdorff space X with countable basis such that each point x of X has a neighborhood that is homeomorphic with an open subset of* $\mathbb{R}^m$ " [Munkres 1999]
"*A manifold of dimension n , or more concisely an n-manifold, is Hausdorff space M in which each point has an open neighborhood homeomorphic to* $\mathbb{R}^n$ " [Hatcher 2001]

Despite this kind of empirical support for the manifold hypothesis, definitive conclusions cannot be drawn. On the opposite, the research from [Balestriero et al. 2021] seems to go in a different direction. They found that when the dimension of input data is high (approximately >100), the probability that a new data point falls inside the convex hull defined by training data is very little, regardless of the dimension of the underlying manifold. This means that if a neural network processes a new data point, it is extrapolating instead of interpolating. But if it is extrapolating, then it cannot have learnt a complete manifold, it has learnt only some relationship among the data that is valid inside the manifold (in the training data) but also outside the manifold (in real operational data, or in test data). So, while in theory neural networks can learn manifolds if fed with a sufficient number of samples (that should be exponential with respect to the number of dimensions of data), in practice that never happens with high dimensional input. This implies that it is not correct to think that neural networks in general are learning manifolds, it may be better to think of them as learning meaningful composable relationships in data, or in other words it may be better to think that neural networks are learning hypernetworks structures within data.

2.4 Deep Reinforcement Learning

Reinforcement Learning is a method for an artificial agent to do automated learning using the outcomes caused by its actions. A definition from [Sutton & Barto 2018] reports: “Reinforcement learning is learning what to do - how to map situations to actions - so as to maximize a numerical reward signal. The learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them”. The fact that the agent is not following any prior plan is the reason why we chose to use RL to answer the third research question “Is it possible to make behavioural patterns emerge in the game without specifying the behavioural rules in detail ?” .

The theoretical exposition of RL in this paragraph (2.4), when a different source is not referenced, is taken from [Sutton & Barto 2018] and OpenAI [web reference 16], with possible change of variables names to have a consistent description. There is no theoretical contribution of the author of this thesis to the RL formalization in this paragraph, and we want to make it clear to avoid repeating continuously the two previous references .

2.4.1 Reinforcement Learning Formalization

A formalization of the Reinforcement Learning problem may be done expressing it as a Markov Decision Process (MDP). A MDP is a way to describe how an agent passes from one state to

another and possibly obtains a reward as a consequence of its actions. One requirement is that states must have the "Markov Property": the probabilities to move from one state to another and to obtain rewards depend only on the knowledge of the current state and the chosen action, and not on any previously accessed state. In other words, a state description has the Markov property if it contains all the necessary information from the past and from the present to determine the transition probabilities to other states and the rewards.

This means that theoretically, non-Markov states can be turned in Markov states if all the history of past states and interactions are added to them (but this way to describe states as long sequences of the past would complicate the algorithms).

Formally, following OpenAI definition [web reference 16], a MDP is a 5-tuple $\langle S, A, R, P, \rho_0 \rangle$:

- S is the set of all states
- A is the set of actions
- $R : S \times A \times S \rightarrow \mathbb{R}$ is the reward function, with $r_t = R(s_t, a_t, s_{t+1})$
- $P : S \times A \rightarrow \mathcal{P}(S)$ is the transition function, with $P(s'|s, a)$ being the probability of transitioning from state s to state s' after taking action a
- ρ_0 is the distribution of initial state

The policy that decides which action to take depending on the state, is a stochastic function π . such that if at time t the state is s_t , it will return a probability distribution over the actions, from which it may be sampled the action to take a_t:

$$a_t \sim \pi(\cdot\,|s_t)$$

Equation 2.15

A sequence of states and actions is called "trajectory", identified by the symbol τ

$$\tau = (\, s_0, a_0, s_1, a_1, \dots)$$

Equation 2.16

The return $G(\tau)$ of a trajectory is the (potentially infinite) sum of all rewards of the trajectory, with $\gamma \in [0,1]$, and may be time-discounted if $\gamma < 1$:

$$G(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t$$

Equation 2.17

Given a policy π , the probability of following any trajectory τ made of T steps is:

$$P(\tau|\pi) = \rho_0(s_0) \prod_{t=0}^{T-1} P(s_{t+1}|s_t, a_t)\pi(a_t|s_t)$$

Equation 2.18

And the expected return $J(\pi)$ is :

$$J(\pi) = \int_{\tau} P(\tau|\pi)G(\tau) = E_{\tau\sim\pi}[G(\tau)]$$

Equation 2.19

The problem of Reinforcement Learning is to find the policy that maximizes that expectation. Hence:

$$\pi^* = arg\max_{\pi} J(\pi)$$

Equation 2.20

Where π^* is the optimal policy.

The Value Function for a given policy π is:

$$V^{\pi}(s) = E_{\tau\sim\pi}[G(\tau)|s_0 = s]$$

Equation 2.21

The Action-Value function is:

$$Q^{\pi}(s, a) = E_{\tau\sim\pi}[G(\tau)|s_0 = s, a_0 = a]$$

Equation 2.22

It has to be noted that

$$V^{\pi}(s) = E_{a\sim\pi}[Q^{\pi}(s,a)]$$

Equation 2.23

The value function may be expressed in a recursive way because the value of a state is dependent on the value of the next state, recursively. The Bellman Equations express this relationship:

$$V^{\pi}(s) = E_{\substack{a\sim\pi \\ s\prime\sim P}}[R(s,a,s') + \gamma V^{\pi}(s')]$$

Equation 2.24

$$Q^{\pi}(s,a) = E_{s\prime\sim P}\left[R(s,a,s') + \gamma E_{a\prime\sim\pi}[Q^{\pi}(s',a')]\right]$$

Equation 2.25

The “Advantage Function” indicates how much it is better or worse to take a certain action (and after that follow the policy) instead of taking the action suggested by the policy, i.e. it indicates the relative advantage of taking a certain action with respect to the policy. The advantage function is the following:

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s)$$

Equation 2.26

Among RL methods, a first distinction may be between on-policy and off-policy methods: on-policy methods improve the same policy that is followed by the agent during training, while in off-policy methods training is done following a policy that is different from the one that is evaluated or improved.

A second distinction may be made between value based methods and policy based methods. In value based methods the algorithm aims at estimating a value function or an action-value function and consequently modify the policy to prefer actions that lead to better values. The “*Policy Improvement Theorem”* [Sutton & Barto 2018] guarantees that modifying the policy for a state in a way that obtains a value improvement in that state strictly improves the expected return $J(\pi)$ and hence improves the overall policy.

In policy based methods instead, the action is directly chosen by a policy function (that outputs a probability distribution over actions) and value or action-value functions may not be present, or may be used only to improve learning.

2.4.2 Policy Based Methods

Policy based methods, also called “Policy Gradient methods” and “Policy Optimization methods” (those two expressions have slightly diverse meaning), differently from the value based methods, may or may not use value or action-value functions, and are characterized instead by having a parametrized policy function that, given the state as input, returns a probability for each action, without explicitly assigning expected returns values.

In other terms, the policy function $\pi(\cdot\,|s_t)$ is a function that is explicitly parametrized by θ (for instance θ may be the weights of a neural network), such that if at time t the state is s_t , it will return the probability distribution over which action to take a_t :

$$a_t \sim \pi_\theta(\cdot\,|s_t)$$

Equation 2.27

That is like equation 2.15 with the addition of parameters θ.

One way in which a neural network can output a probability distribution over a set of discrete actions is to have a last layer with as many neurons as the number of actions, and on top of that operate a Softmax. If instead the actions are continuous then the last layer of the neural network will have as many neurons as continuous parameters in the action, and each neuron will be considered the mean of the distribution of that action parameter (the distribution type may be arbitrary, for instance a gaussian usually works well).

The problem of reinforcement learning is still the same, we want to maximize the expected return $J(\pi)$, (detailed equation 2.19), and we would like to optimize directly the policy parameters θ. One way to do that would be to do gradient ascent using the gradient of the expected return $J(\pi)$ over the parameters θ:

$$\theta_{k+1} \leftarrow \theta_k + \eta\, \nabla_\theta J(\pi_{\theta_k})$$

Equation 2.28

With η learning rate.
So it is necessary to compute the gradient of the expected return $J(\pi_{\theta_k})$.

2.4.2.1 Policy Gradient

A basic algorithm that does that is the one called “Vanilla Policy Gradient” (for demonstration and details see OpenAI, web reference [16]) or “REINFORCE” [Williams 1992], and consists in doing gradient ascent on $J(\pi_{\theta_k})$, where the gradient is computed as follows :

$$\nabla_\theta J(\pi_\theta) = E_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t)\, G(\tau)\right]$$

Equation 2.29

We obtained the gradient of the expected return in form of an expectation. This means that if we can sample what is contained inside the expectation we can use it to estimate the gradient. It turns out that we can do that: the expectation is with respect to the trajectory τ , whose probability distribution is determined by the environment and by our policy π_θ , so by making the agent follow the policy we can collect the returns $G(\tau)$ and compute the estimate of the expectation. Since the agent must follow the policy π_θ the algorithm is on-policy .

If we run N different episodes or trajectories $\{\tau_0, \tau_1, .., \tau_{N-1}\}$, with respective trajectory lengths $\{T_0, T_1, .., T_{N-1}\}$ we can do batch gradient ascent. If we call the actions and states of trajectory i at time t respectively with $a_{i,t}$ and $s_{i,t}$ the following is the mean gradient:

$$\Delta\theta = \frac{1}{N}\sum_{i=0}^{N}\sum_{t=0}^{T_i-1} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t})\, G(\tau_i)$$

Equation 2.30

Then we can update the weights of the policy neural network:

$$\theta \leftarrow \theta + \Delta\theta$$

Equation 2.31

We can decompose $G(\tau)$ in 2 sums, one of the rewards before the action and one after the action happened at time t (named “rewards-to-go”).
We use $G(\tau_{0:t-1})$ for the past rewards:

$$G(\tau_{0:t-1}) = \sum_{k=0}^{t-1} \gamma^k r_k$$

Equation 2.32

And we use $G(\tau_t)$ for the “rewards-to-go”.

$$G(\tau_t) = \sum_{k=t}^{T-1} \gamma^k r_k$$

Equation 2.33

$$G(\tau) = G(\tau_{0:t-1}) + G(\tau_t)$$

Equation 2.34

It can be proven [web reference 16] that a formula that is equivalent in expectation but with lesser variance is possible if we use only the rewards-to-go instead of complete return in the gradient computation:

$$\nabla_\theta J(\pi_\theta) = E_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \sum_{l=t}^{T-1} \gamma^l r_l\right]$$

Equation 2.35

This allows us to use, for each action, only the rewards obtained after it (the rewards-to-go) to calculate the gradient of the expected return. Using the complete returns is not wrong, because it is equal in expectation, but it would introduce much more variance in the gradient and that would slow down learning. Using $\gamma = 1$ would assure that each action has the same importance, it can be used in finite-horizon (episodic learning). In infinite time horizon it is not possible to collect the whole return (the trajectory never ends), so in that case it is necessary to bootstrap $G(\tau_t)$ using only the reward for the first step (or for some steps) and then summing a discounted state value.

For instance, in infinite-horizon, an k-step bootstrap would be:

$$\widehat{G_k}(\tau_t) = \sum_{l=t}^{t+k-1} \gamma^l r_l + \gamma^{t+k} V(s_{t+k})$$

Equation 2.36

$\widehat{G_n}(\tau_t)$ is equal in expectation to $G(\tau_t)$, so it is theoretically correct to use it, but since in any algorithm we are not using the real state value but an approximation, it will introduce bias (but it will allow to work with infinite-horizon cases).

In infinite horizon case moreover, since we don't use the whole infinite trajectory at once but only n-step of it a time, we have to consider the infinite trajectory as equivalent to an infinite set of n-step episodes. So, every n-step we will reset t to 0, that is also necessary to avoid that the discount makes progressively less relevant the subsequent rewards. Or, if we don't want to reset t to 0, just change the equation a little, subtracting t to the exponent:

$$\widehat{\widehat{G_k}}(\tau_t) = \sum_{l=t}^{t+k-1} \gamma^{l-t} r_l + \gamma^k V(s_{t+k})$$

Equation 2.37

In this way the finite and infinite horizon are included in the same framework.
Also $Q^{\pi_\theta}(s_t, a_t)$ is theoretically correct to be used instead of $G(\tau_t)$, because since inside the expectation the actions are distributed following the policy π_θ , it implies that $Q^{\pi_\theta}(s_t, a_t)$ is equal in expectation to $G(\tau_t)$ (another formal proof by OpenAI may be found in web reference [18]).

In fact, it is theoretically correct also to use, instead of $G(\tau_t)$, any other expression that is equal in expectation. Every expression that starts with $G(\tau_t)$ or $\widehat{G_n}(\tau_t)$ and then adds any other expression that does not depend on the current action (but may depend on state), is equal in expectation.
So, more generally:

$$\nabla_\theta J(\pi_\theta) = E_{\tau \sim \pi_\theta}\left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t)\, \Phi_t\right]$$

Equation 2.38

With Φ_t that may be one of:

$$G(\tau_t)$$

$$\widehat{G_k}(\tau_t)$$

$$\widehat{\widehat{G_k}}(\tau_t)$$

$$Q^{\pi_\theta}(s_t, a_t)$$

$$G(\tau_t) + b(s_t)$$

$$\widehat{G_k}(\tau_t) + b(s_t)$$

$$\widehat{\widehat{G_k}}(\tau_t) + b(s_t)$$

$$Q^{\pi_\theta}(s_t, a_t) + b(s_t)$$

Equation 2.39

Where $b(s_t)$, usually called “baseline”, is an additional expression that may depend on state and may be negative. For example, it may be $V^{\pi_\theta}(s_t)$.
So many different Φ_t are possible. For instance if we start from $Q^{\pi_\theta}(s_t, a_t)$ and we add $b(s_t)$ = $-V^{\pi_\theta}(s_t)$, we end up with the Advantage function $\Phi_t = A^{\pi_\theta}(s_t, a_t) = Q^{\pi_\theta}(s_t, a_t) - V^{\pi_\theta}(s_t)$, see 2.26 .

The advantage function makes sense because the policy gradient update is intended to increase the probability of actions that perform better than the policy and to decrease the probability of actions that perform worse than the policy, if we use the Advantage function as Φ_t we are sure to have positive updates when the action is better than current policy choice, and negative updates when it is worse. This will have the effect of reducing variance in the gradient updates and hence speed up learning.

Policy gradient methods that use a value function to bootstrap the estimate of the total return such in 2.34 are called “actor-critic”, where actor is referred to the policy function and critic to the value function [Sutton and Barto 2018]. Some authors call “actor-critic” every policy method that uses a value function as component of Φ_t, but Sutton and Barto specify that the term should be exclusive for those that use value function to bootstrap the estimate of the total return.

2.4.2.2 Generalized Advantage Estimation

[Schulman et al. 2016] introduced a particular version of the Advantage Function, the "Generalized Advantage Estimation", with proven variance reduction property. The setting is the policy gradient family of algorithms, using undiscounted rewards. So there is not a time-dependent parameter γ to discount the rewards, but there is another parameter named γ with a different meaning but same mathematical behaviour: it is meant to downweight rewards corresponding to delayed effects. Thus the meaning is different from the "discount", it does not mean that delayed effects are less "useful" at current time (that would be the meaning of the classical time-discount), but it still discounts the delayed effects, and doing so it introduces bias, because in this way the total return will be smaller in absolute value. On the other hand it will decrease variance, for the same reason: smaller absolute value means smaller variance among different trajectories. So the meaning of this parameter $\gamma \in [0,1]$ here is to be a switch for smoothly parametrize a trade-off between bias and variance, with bias increasing and variance decreasing when γ decreases, and bias decreasing and variance increasing when γ tends to 1.

Then this Generalized Advantage Estimation has another characteristic that decreases variance but introduces bias: it computes the estimate of the returns as an exponentially weighted average of k different n-step bootstrap estimates, where n goes from 1 to k. To be clearer, if we define:

$$\delta_t^V = r_t + \gamma V(s_{t+1}) - V(s_t)$$

Equation 2.40

Imagine of having a trajectory of length at least k, then we could build a 1-step bootstrap estimate of the advantage function at time t :

$$\hat{A}_t^{(1)} = r_t + \gamma V(s_{t+1}) - V(s_t) = \delta_t^V$$

Equation 2.41

But we could also build a 2-step bootstrap estimate, or a 3- step estimate, or a k-step estimate etc. :

$$\hat{A}_t^{(k)} = \sum_{l=t}^{t+k-1} \gamma^{l-t} r_l + \gamma^k V(s_{t+k}) - V(s_t) = \sum_{l=t}^{t+k-1} \gamma^{l-t} \delta_t^V = \widehat{\widehat{G_k}}(\tau_t) - V(s_t)$$

Equation 2.42

In infinite horizon, k may tend to ∞. In that case the $\gamma^k V(s_{t+k})$ at each time will be canceled by the discounted $-V(s_t)$ of the next time and we would have:

$$\hat{A}_t^{(\infty)} = \sum_{l=t}^{\infty} \gamma^{l-t} r_l \; - V(s_t) = \sum_{l=t}^{\infty} \gamma^{l-t} \delta_l^V$$

Equation 2.43

Now, a 1-step estimate has low variance but high bias, while a 5-step estimate has bigger variance and lower bias, the more steps we include in the estimate the bigger the variance and the lower the bias. It is possible to compute an exponentially weighted average of all n-steps estimators, parametrized by λ such that when $\lambda = 0$ the weighted average coincides with $A_t^{(1)}$ and when $\lambda = 1$ it coincides with $A_t^{(\infty)}$, so that λ is another parameter that may be used to tune the bias/variance trade-off.

The “Generalized Advantage Function” is:

$$\hat{A}_t^{GAE(\gamma,\lambda)} = (1-\lambda) \sum_{k=1}^{\infty} \lambda^{k-1} \hat{A}_t^{(k)}$$

Equation 2.44

And a more practical formulation may be:

$$\hat{A}_t^{GAE(\gamma,\lambda)} = \sum_{l=t}^{\infty} (\lambda\gamma)^{l-t} \delta_t^V$$

Equation 2.45

This GAE hence can be used instead of the Advantage Function and put in place of Φ_t in the policy gradient (see equation 2.36) . This allows tuning variance and bias with two parameters: γ controls how far in time to consider the rewards relevant, and λ that decides how much the return component of the advantage has to be similar to a 1-step bootstrap (if λ=0) or progressively to a Monte Carlo return (if λ=1).

2.4.2.3 Proximal Policy Optimization

In the Proximal Policy Optimization an alternative optimization problem is posed, such that allows to do more than one policy gradient ascent iteration in each optimization round (using the same set of trajectories).

The Reinforcement Learning problem here is framed in a different (but equivalent) formulation. In the classical formulation, we aim at maximizing $J(\pi_\theta)$, that for ease of reading we may write $J(\theta)$. But equivalently, we may maximize the performance of a policy (the one to be maximized) with respect to a fixed policy (the one used until now to generate trajectories). If we call π_θ, the optimized policy (parametrized by the new parameters θ') and π_θ the fixed policy, we may want to maximize $J(\theta') - J(\theta)$.

Following [Schulman et al. 2015], it is possible to show that

$$J(\theta') - J(\theta) = E_{\tau \sim \rho_{\theta'}(\tau)} \left[\sum_t \gamma^t A^{\pi_\theta}(s_t, a_t) \right]$$

Equation 2.44

That means that the difference between the expected return of the policy parametrized by θ' and the expected return of the policy parametrized by θ is equal to the expectation of the advantage function $A^{\pi_\theta}(s_t, a_t)$ over trajectories distributed by the policy parametrized by θ' (since the expectation of the trajectories are distributed by θ' it means that the actions are distributed by π_θ, ,so $A^{\pi_\theta}(s_t, a_t)$ it is the advantage function of the actions selected by the new policy θ' with respect to the fixed policy θ) .

We may develop the equation above into:

$$\sum_t E_{s_t \sim \rho_{\theta'}(s_t)} \left[E_{a_t \sim \pi_\theta(a_t|s_t)} \left[\frac{\pi_{\theta'}(a_t|s_t)}{\pi_\theta(a_t|s_t)} \gamma^t A^{\pi_\theta}(s_t, a_t) \right] \right]$$

Equation 2.45

The most external expectation, over the states distribution, is depending on the optimizing policy θ'. This means that if we use the samples generated from the fixed policy π_θ we would ignore the difference in states distribution consequent to using a different policy. We could decide to do it on the condition that the distributions are not too different: with a bounded

Kullbach-Leibler divergence between $\pi_{\theta'}$ and π_θ we could suppose that $\rho_{\theta'}(s_t)$ is approximated sufficiently well by $\rho_\theta(s_t)$ and we could use the samples obtained by π_θ.

In that case, a penalty may be imposed on the Kullbach-Leibler divergence of the two distributions, and the objective to be maximized would be:

$$\sum_t E_{\tau\sim\rho_\theta(\tau)}\left[\frac{\pi_{\theta'}(a_t|s_t)}{\pi_\theta(a_t|s_t)}\gamma^t A^{\pi_\theta}(s_t,a_t) - \beta D_{KL}(\pi_\theta||\pi_{\theta'})\right]$$

Equation 2.46

For some coefficient β.

But it is hard to find the right β, so [Schulman et al. 2017] in their Proximal Policy optimization suggest using clipping as a way to bound the difference in distribution. For ease of reading, let us call the ratio between the two probabilities $d_t(\theta) = \frac{\pi_{\theta'}(a_t|s_t)}{\pi_\theta(a_t|s_t)}$. A new surrogate objective function that uses clipping, called L^{CLIP} may be devised:

$$L^{CLIP}(\theta) = E_{\tau\sim\rho_\theta(\tau)}[\gamma^t \min(\, d_t(\theta)A^{\pi_\theta}(s_t,a_t), clip(\, d_t(\theta), 1-\epsilon, 1+\epsilon\,)A^{\pi_\theta}(s_t,a_t)\,) \quad]$$

Equation 2.47

To practically use it, we have to do gradient descent with respect to θ' on 2.46 or 2.47 , using an automatic differentiation library that supports clipping.

A further improvement to the PPO with ratio clipping is the early stopping based on the Kullbach-Leibler divergence: at each policy gradient ascent iteration (in the algorithm above the iteration of m from 0 to M), an estimate of the Kullbach-Leibler divergence between the old and the new policy is computed using the action probabilities, and if that divergence is bigger than a certain threshold, the policy gradient ascent iteration cycle is stopped for this learning round (a new policy gradient ascent iteration cycle may be executed later in a new learning round with newly sampled trajectories). This avoids to use samples generated from a policy that has a distribution too different from the one that gets optimized.

2.4.3 Multi-agent Reinforcement Learning

When an agent is not playing against an opponent, that is when we are not in a competitive game, if the environment is immutable the reinforcement learning process has just to learn how the environment works or reacts to our agent actions.

Things get more complicated when there are more than one agent: the success of an agent is dependent on the policy of all agents: a change in the policy of an agent may lead to a general improvement (or at least to no worsening) only if all other agents are not changing their policies. But other agents may change their policies as well: collaborative agents may change their policies with the purpose of improving the outcome of the team. They may be successful, but that is not guaranteed, since the change in their personal policy that is believed to improve the expected rewards, sometimes may be effective only if all other agents are not changing. Since collaborative agents may have an imperfect communication and synchronization, they may change behaviour in a way that is counter-productive since other companions may have changed their behaviour in turn! So, in multi-agent systems such as in robotic soccer better performances can be obtained if there is some form of implicit or explicit coordination among agents. In addition to this, the assignation of reward for a certain action is not easy to be learnt by an agent because the goodness of the outcome may not depend actually on what the agent did but rather on what another agent did.

The complexity grows even more if there are competitive agents (opponents): they may change their policies to counter effect the policy of our agent.

Those difficulties are well known in multi-agent literature, where one common way to model collaborative-competitive multi-agent games is to describe them as Markov Games [Vlassis 2007] or as Extensive Form Games [Zhang et al. 2021] and consider for each state all the possible actions of all the agents. Trying to solve a Markov Game, such as finding Nash equilibria or finding a sufficiently good strategy may often not be feasible (this may be particularly true in the field of robotic soccer, where even only modelling the opponent strategy for each player depending on the state may not be practical because the search space may be too large).

When the total utility may be decomposed in a sum of individual utilities approximated for each agent, and when those utilities depend on local variables, i.e. only some interaction with some other agents influence the local utility, it is possible to build a graph of dependence between agents. That graph will indicate which agent is involved in which utility function. Then each

agent action will be chosen individually to optimize only the utilities who are affected by it. [Guestrin et al. 2002]. This will decrease the combinatorial complexity, but it may still be unfeasible because each utility may still include actions from many agents, and it will be anyway an approximated solution. Unfortunately, in robotic soccer we cannot make such a subdivision of local utilities and this method is not applicable.

A way to deal with the problems introduced by multi-agent settings may be to consider teammates and opponents just as part of the environment and use single agent algorithms. That simplifications make feasible to apply single agent reinforcement learning algorithms, but they are not theoretically sound in multiagent context and do not guarantee good results (other agents can't be rightfully considered as an immutable environment), even if in practice they have been used successfully in a certain number of cases [Busoniu et al. 2008]. Considering the other agents as part of the environment reduces the performance of the reinforcement learning system and the accuracy of the Value Function. Anyway, when using massive computational resources, big deep learning models as function approximators and a lot of training time, even single agent reinforcement learning system may achieve great performance in multiagent environments. One example of this is OpenAI Five, the system trained to win the multiplayer game Dota 2 [OpenAI 2019].

2.4.4 Reinforcement Learning in Robotic Soccer

There have been many attempts to apply reinforcement learning to players in 2D simulations, even if there is not, up to now, a complete end-to-end successful experiment in which players are either not using pre-built skills or not using a simplified version of the task.

2.4.4.1 RL in Simulations with Simplified Tasks

[Stone and Veloso 1999] used Reinforcement Learning to train an agent in RoboCup 2D to make him decide where to kick the ball once he possesses the ball, with 8 possible actions corresponding to 8 fixed position in the pitch. The agent was not learning what to do in other situations (i.e. when not possessing the ball), that cases were managed with other algorithms. An approach called "layered behaviour" was used: complex task are subdivided into easier tasks, then methods to solve the easier tasks are developed, and used to solve also the complete task. A first layer of learning, not using reinforcement learning, was constituted by a neural network trained (presumably with supervised learning) to learn how to intercept the ball, and that functionality was used to separately train a decision tree (again presumably with

supervised learning) to learn when a pass had a good probability to reach the companion. The decision tree then was applied to the state and the forecast used to enrich the input state of the RL algorithm. The Reinforcement Learning algorithm used was invented for that scope and named TPOT-RL. It was deciding one action at time (only for the player who was the ball owner), so it was not actually a multi-agent algorithm (even if in the formalization, the sequential chain of actions may be possibly from different agents, so the name “multi”, but it was not concurrently).

In the RoboCup 2D simulator environment, the “Karlsruhe brainstormers” team has been using Reinforcement Learning to learn which hand-written high-level sequence of actions to take [Merke and Riedmiller 2001]. Their approach was founded on single agent reinforcement learning algorithms and assumed that all team agents are collaborating. Since it was based on human-made sequences of actions it can’t be considered “full reinforcement learning” from the grassroot, but anyhow it is an example of using RL to learn to solve some multiagent problem using only a single agent algorithm. In a later iteration of the team [Riedmiller and Gabel 2006], they managed to use reinforcement learning to learn some sequence of actions to do some low-level task such as kicking the ball to a certain direction with a certain strength, intercepting the ball, going to a certain position and so on.

Since the full RoboCup 2D simulation is a very complex environment in which doing RL, some experiments have been made to reduce the complexity of the task.

[Kostiadis and Hu 1999] used the Reinforcement Learning algorithm Q-Learning to train an agent in the “attacker’s dilemma”: an attacker starts with the ball in his possession in the proximity of opponent’s goal, and also a companion is in a position from which it is possible to score, while an opponent player is placed randomly in a position either blocking the shot of the ball owner or blocking the pass to the attacker’s companion. The learning agent (the ball owner) learns in each situation which of the 2 actions is better: shooting to goal or passing to companion.

Another reduction of the complexity of the task was the so-called “Keepaway” subtask [Stone and Sutton 2002] . In Keepaway there is the team of “keepers” that owns ball possession and whose goal is to keep it, and the “takers” who are the opposing team trying to steal the ball. To simplify things KeepAway is played in Robosoc2D environment but players have complete noise-free 360 degrees sight, and they have some built-in hand-written skill such as the ability of kicking the ball towards a certain player, going to an opponent-free position, reach the ball, hold the ball away from opponents, block the pass between keepers. The application of single

agent reinforcement learning with the algorithm Sarsa(λ) [Stone and Sutton 2001] lead the agents to learn better "keeping" strategies than a benchmark hand-coded strategy, while having better performance in the "taking" task than a benchmark hand-coded simplified strategy and equal performance than another more refined hand-coded strategy. In a subsequent study [Stone et al. 2005] the Sarsa(λ) was applied successful also in a noisy, limited sight setting.

Also [Kostiadis and Hu 2001] applied Reinforcement Learning to the Keepaway task, with 3 attackers ("keepers") and 2 defenders ("takers"). They used Kanerva coding to generate features: 5000 prototypes generated the same number of features when confronted for similarity with the current state (including symmetric transformations of the state). Those features were fully connected to 3 outputs indicating the 3 possible actions: "pass to companion 1", "pass to companion 2", "hold the ball". The action corresponding to the greater output was selected, connection weights were updated through residual Q-Learning. Each of the 3 attacking agents underwent individual training so each may possibly learn a different policy. The experiment showed that agents trained with this reinforcement learning method outperformed all benchmark policies.

[Bai et al. 2018] applied Hierarchical Reinforcement Learning to the Keepaway problem, following the methodology of Hierarchies of Abstract Machines, again using high-level actions.

Reinforcement Learning was applied to the problem of Keepaway also by [Ollino et al. 2018], but they used the simulation of RoboCup Small Size League, with GrSim. They used Q-Learning in a centralized artificial intelligence instead of having independent agents. Agents had hand-written skills such as passing the ball to some other player or intercepting the ball.

An extension of Keepaway is "Half Field Offense", in which the game is limited to one half pitch with one team trying to score and the other trying to defend. It is still a limitation of the whole game, but more similar to it than Keepaway. [Kalyanakrishnan et. al 2007] applied Reinforcement Learning to this "Half Field Offense" task in the RoboCup 2D simulator, using agents with built-in hand-written skills such as passing to a companion, kicking to the best promising part of the goal, dribbling the ball avoiding defenders, getting the ball, going to attack position. Moreover, the strategy is built-in for each player when not in possession of ball, only when they have the ball the Reinforcement Learning algorithm works. They used the algorithm Sarsa(0) using tile coding CMAC to discretize the state, gave rewards to each action that ends as intended (for instance a pass that reaches the correct companion), and enabled communication between players to share information about states, actions and rewards.

Results were promising. They tried the same algorithm but with a neural network as value approximator instead of tile coded states-value tables, and they obtained a worse performance.

Also [Hausknecht and Stone 2016] tried to face the Half Field Offense task with Reinforcement learning. Even if Half Field Offense environment supports hand-written high-level skills, they reportedly trained one agent to use the basic actions. They used Deep Deterministic Policy Gradient (DDPG) with neural networks as function approximators, and a set of pre-processed input features. They used the ADAM method for gradient descent, with the inverted gradient technique for actions own parameters learning: to avoid certain types of parameters such as “angle degrees” (in directional actions) exiting the acceptable interval and grow unlimited, each time the parameters exceeded the interval, the gradient was inverted in sign, as a sort of balancing feedback. They noticed that only with this mechanism the algorithm was capable of learning.

Similar to “Half Field Offense” was the task of “Breakaway” by [Torrey et al. 2005]. Inside the RoboCup 2D simulation they set a task in which attacking players tried to score a goal against defending players. They trained the agents with Reinforcement Learning, with state encoded in tile coding to have discrete states, on the Keepaway task, using the SARSA algorithm with support vector regression. Then transferred the knowledge to the new Breakaway task using “advice”: a human supervisor indicating if a transferred behaviour is good or not. Agents were endowed with hand-written skills such as in Breakaway, so they did not have to learn how to pass the ball, reach it, hold it, and so on.

[Scardua et al. 2000] applied Reinforcement Learning in the RoboCup 2D environment, and simplified the problem using high-level hand-written actions, and disputing a game of 1 vs 1 player in which the first player was an agent trained with reinforcement learning and the second was choosing randomly his actions. A neural network was used as a function approximator to decide which of the 3 high-level actions to take: get the ball, kick towards adversary goal, turn the ball. They did not specify exactly which RL algorithm has been used, but it was based on action-value.

Again in RoboCup 2D [Tuyls et al. 2002] were able to use Q-Learning and Bayesian network to make an agent alone learn how to run to the ball and to run with the ball.
[Gabel et al. 2021] used the DQN Reinforcement Learning algorithm in the Robocup 2D simulator with full observability to train an agent to run to a target position, without using hand-written skills.

[Celiberto et al. 2007] in the RoboCup 2D simulator used reinforcement learning to train a defender and a goalkeeper in the task of defending against two opponents. The task was simplified by the existence of hand-coded skills such as intercepting the ball, dribbling with the ball, passing the ball to the companion and so on. Agents learnt how to combine those built-in behaviours. The state space was simplified dividing the pitch in squares. It was applied a derived version of Q-Learning where the action-value function was not determined only by the q-value but also by a hand-written heuristic that simplified learning (Heuristically Accelerated Q–Learning).

[Ahumada et al. 2013] trained a goalkeeper in RoboCup Small Size League simulated environment GrSim. They hand-coded skills to kick the ball, intercept the ball, going into defense position, and applied an algorithm derived from Q-Learning on top of those basic skills, using a Kalman filter to predict ball position. They compared their algorithm with the Heuristically Accelerated Q–Learning and found an equivalent efficiency.

[Masson et al. 2015] applied Reinforcement Learning in RoboCup 2D to teach one agent to score into an empty goal. The experiment was a test for their Q-PAMDP algorithm that is a special version of Q-Learning in which it is possible to learn actions with continuous parameters.

[Watkinson and Cam 2019] applied RL, more precisely Deep Deterministic Policy Gradient (DDPG) to train a Robocup 2D agent to score both in an empty goal and in a goal defended by a basic opponent. They used “curriculum learning” (they name it instead “transfer learning”, that is a more general term) to train the agent at first with easier positions of agent and ball, and then increasing the distance between agent and goal, and agent and ball.

[Ma and Cameron 2008] used reinforcement learning to make agents in RoboCup 2D soccer choose which of the built-in hand written plans to use depending on the situation.

Another usage of RL in simulated soccer robotics has been made by [Wiering et al. 1999] in their own 2D soccer simulator, in which many simplifications were adopted, such as the fact that there could be just 1 o 3 players per team, players could automatically move the ball with them, kicks had always the same initial strength/velocity of the ball, players could turn only either to ball or to goal. This allowed both to have a very small action space to choose on, and to exempt the agents to learn how to reach the ball, how to dribble, how to go towards the goal, since those behaviour were automatically built into the simulator. In this extremely

simplified environment the researchers were able to apply value-base reinforcement learning algorithms.

[Kalyviotis and Hu 2002] used a neural network to decide which hand-written plan to use in RoboCup 2D soccer agents. It was supervised learning and not reinforcement learning, but it was solving the task of taking a decision depending on the context so it was similar in purpose, and it used the neural network as an approximator for the expected outcome, that is a similar concept to the Valued Function of RL systems.

[Schwab et al. 2019] used Deep Reinforcement Learning (Deep Deterministic Policy Gradient) to train Small Size League robots in the CMDragons simulation in some basic skills like “go to ball” or “aim and shoot”. They update their research in [Zhu et al. 2019], where they also tested the system on real robots finding that it is slightly less efficient than in the simulation, but still successful.

[Yoon et al. 2016] used Temporal Difference Reinforcement Learning, with a feedforward neural network as value function approximator, to train a simulated Small Size League robot agent, to the task of shooting and score, both with and without goalkeeper.

[de Medeiros et al. 2020] applied the RL algorithm Proximal Policy Optimization to agents in a simulated environment of the IEEE Very Small Size Soccer robots to the task of intercepting the ball and pushing it to the opponent’s side. The action space was represented by the position that the agent chooses to reach, using a built-in higher-level skill to go there.

[Pena et al. 2020] trained agents with RL in the FIRASim simulator (derived from GrSim) for IEEE Very Small Size Soccer, in the 3 vs 3 competition. They used Double Deep Q-Network (DDQN) and Deep Deterministic Policy Gradient (DDPG) in separate experiments, to choose among strategies to apply, with agents using built-in high-level actions such as reach the ball, try to score, follow a target. They obtained better outcomes with DDPG.

[Fukushima et al. 2019a] used imitation learning to copy the team behaviour of a strong team in RoboCup 2D. It is not actually Reinforcement Learning because they used neural networks with a supervised learning algorithm, but it tries to solve the same problem of RL: to associate the best action to a state. In fact they did not do a “direct” imitation learning because they did not directly imitate actions, rather they trained an evaluation function that was evaluating the goodness of actions, so to use it when planning sequences of actions. The algorithm did not evaluate basic actions, but higher-level behaviour such as pass, dribble and so on.

[Bai et al. 2015] applied Markov Decision Process online planning with hierarchical decomposition, in the RoboCup 2D simulator, inside their agent “WrightEagle” that has been several times world champion and runner-up. Their algorithm, named “MAXQ-OP” it is not, strictly speaking, a “learning” algorithm, it is rather a planning algorithm, but we included a mention because in Reinforcement Learning textbooks, planning algorithms are usually presented as the first examples of MDP-solving algorithms.

2.4.4.2 RL in 3D Humanoid Simulated Robots

Some other application of Reinforcement Learning to robotic soccer are in a different domain with respect to our research, outside the 2D simulation, either in real robots or in 3D simulations of real robots. When dealing with real robots or 3D simulations, there are additional problems that need to be solved (and may be addressed by Reinforcement Learning) such as controlling movements, transforming camera observations into useful information, etc. that in the 2D simulations usually are not existing. Even if those works are not directly regarding our research, we list them below for completeness.

For instance [Abreu 2019] used Proximal Policy Gradient Optimization (an actor-critic RL algorithm) to train a humanoid robot in the RoboCup 3D simulation to run as fast as possible: coordinating movements in order to run is not a problem that has to be solved in the 2D simulation.

[Ocana et al 2019] used reinforcement learning (DDPG algorithm) in a 2 vs 2 free kicks task in the simulated 3D environment with Nao humanoid robots. Agents were endowed of pre-existing skills to do macro actions, and defending players followed a hand-written policy. They tested both independent learning, in which each agent is independent, and joint-action learning, in which the action choice is centralized and takes in account every possible combination of action for all agents. Independent learning in multiagent systems has the problem, as we wrote above, of being suboptimal because it does not coordinate the actions of all players. On the other hands joint learning may be intractable with increasing number of agents (or may be not possible with lack of communication/centralization). They found out that in this setting joint learning was giving better results.

[Fahami et al. 2017] applied RL in RoboCup 3D soccer with the simulated Nao humanoid robot. Their scope was to teach the robot how to kick the ball towards a certain target.

[Muzio et al. 2020] used deep RL to train the Nao humanoid robot in the Robocup 3D simulation to dribble the ball against one opponent. They tried 3 algorithms: Proximal Policy Optimization (PPO), Trust Region Policy Optimization (TRPO) and Deep Deterministic Policy Gradient (DDPG), and obtained the best performance with PPO.

In the simulated Robocop 3D league, [MacAlpine et al. 2015] trained a humanoid robot with "Layered Learning", that is not properly a reinforcement learning method, but may be used as providing features and "composed actions" to a reinforcement learning system (this was not the case). The tasks to solve where those basic of 3D humanoid robots, such as getting up, learning to walk fast, going to a target, rather than typical 2D-simulation tasks (learning to dribble the opponent, free yourself from opponent, try to score a goal, etc.).

[Asik et al.2019] trained a simulated version of the Nao humanoid robot in the 3D simulation of RoboCup Standard Platform League (SimRobot) using imitation learning. It is not properly Reinforcement Learning but it tries to solve the same type of problem: to associate actions to a situation. The task to solve was to score in an empty goal. They used as inputs the signals of the two (simulated) robot cameras and processed them with convolutional neural networks to do end-to-end learning. They tried 3 different networks architectures: with CNNS, with Hierarchical CNNs, and with CNNs plus a Long Short-Term Memory layer. A teacher was providing step by step the examples to imitate.

[Lobos-Tsunekawa et al. 2018] applied decentralized reinforcement learning using finite support basis functions for state representation, to train the humanoid robot Nao to execute the "in-walk kick", in the SimRobot simulator.

2.4.4.3 RL in Physical Environment

[Uchibe et al. 1996] used Q-Learning to train a 4-wheeled robot with camera to score a goal in both an empty goal and a goal with a moving keeper robot. To add the ability of avoiding an obstacle they tried three methods: in the first two methods they used a second Q-value function to evaluate the situation with respect to the task of avoiding the obstacle and either summed it to the original Q-value (first method) or used it to substitute the original Q-value only when an obstacle was very close (second method). The third method consisted in learn the new behaviour (obstacle avoidance) using the same original Q-value function, and this revealed to give the best performance.

[Enokida et al. 2001] used an algorithm derived from Q-Learning to train a goalkeeper robot to reach the ball, using camera as input and using RL to choose between 5 different directions to go.

[Riedmiller et al. 2009] used neural network based RL to train a Robocup Middle Size robot (a real physical robot) to the task of dribbling the ball (in the same paper they also discussed a separated experiment in which neural network based RL was used in Robocup 2D simulator to train agents to disturb opponents who own ball).

Also [Leottau et al. 2015] applied reinforcement learning to teach a humanoid robot Nao to dribble the ball, training in the SimRobot simulator and validating also on real robots.

[Latzke et al. 2007] used Q-Learning to teach a humanoid Robosapien to dribble the ball to an empty goal, using “imitation”: they provided experience data from a teacher and not from exploration. It worked successfully in the simulator, then once transferred on the real robot it worked successfully. They managed also to successfully learn the behaviour directly from the real robot.

2.5 Summary

In this chapter we reported previous works on robot soccer analysis that evidenced the possibility of proficiently using Deep Learning to capture patterns and hidden structure. Then we described how hypernetworks, that are generalization of graphs with more than 2 vertices per relationship, are a useful language to describe multilevel structures in robotic soccer, and how they share similarities with neural networks, to the point that we could conjecture that for each neural network there may be a corresponding hypernetwork. We also reviewed the literature about the neural networks algorithms that we are going to use in our experiments, and about the reinforcement learning algorithms that we are going to use in answering the third research question, as well as the previous application of reinforcement learning to robot soccer.

Chapter 3. Defender's Dilemma Approach

3.1 The Defender's Dilemma

One of the most basic building blocks of the game of soccer is the creation of situations in which the defending team is in numerical minority in the proximity of the ball, such that the defender player has to choose whether trying to tackle the opponent that owns the ball (with the risk that the opponent may eventually pass the ball to a companion that could advance alone to the goal), or try to stay in a position that avoids the pass to the opponent without ball (with the risk that the one that owns the ball may advance to the goal). This has been called the "Defender's dilemma" by [Johnson and Iravani 2007] (we already presented it in paragraph 2.2.4) and the parallel dilemma from the point of view of the attacker was presented by [Kostiadis and Hu 1999]. The players' configuration may be described by a hypersimplex:
$\langle attacker_1,\ attacker_2,\ defender,\ ball\ ;\ R_{defender's\ dilemma} \rangle$

This basic situation may arise from a certain number of lower-level actions and decision, so it is already an emergent phenomenon, and in turn it may produce other higher-level situations that we may identify as prototypical. Computer simulations may help in studying the creation of defender's dilemma and the subsequent effects in terms of higher-level causation. A first question of interest may be: are winning teams creating more defender's dilemmas ? If so, and if we are able to detect Defender's Dilemmas, we would be able to answer our first research question: "Is it possible to identify patterns of play, that lead a team to obtain an advantage ?".

We applied the Defender's Dilemma analysis to 1020 games played in official RoboCup 2D competitions in the period 2011-2017, downloaded from the official archive website.

Then we applied the Defender's Dilemma also to 3015 games played in our computer by two different instances of Helios2018 (the winner of 2018 RoboCup competition) playing repeatedly against each other, both with basic default configuration (those are the same games that we analysed in Chapter 4 to extract a Value Function).

The Helios2018 vs itself series of games has the characteristic of being based on teams playing against an exactly equivalent opponent, and may serve as a comparison baseline for the real RoboCup 2011-2017 games, where instead the opponents are diverse.

In this analysis we do not consider anything that happens after tick number 8000, because games have a duration of 6000 ticks, after which in the case of a draw there may be an additional 2000 ticks of play, and if even after that the game is a draw there may be a penalty competition, in which we are not interested. That means that we analysed the normal game until tick 6000, and in case of a draw in the first 6000 ticks we analyse also the additional times up to tick 8000, but in the case the teams are still tied at tick 8000 we do not analyse what happens during subsequent penalties. This entails that in case at tick 8000 the two teams scored the same amount of goals we consider the game terminating as a draw (despite the fact that maybe the real game ended up with a penalty victory for one team or the other).

3.2 Defender's Dilemma Algorithm Description

In the previous paragraph we described the Defender's Dilemma in the form of an Hypersimplex, now we want to find the algorithmic equivalent. In the following paragraph we define an algorithm that may discover if a certain game configuration is such to represent a Defender's Dilemma. In this way we instantiate a Hypersimplex definition in a concrete program.

In a Defender's Dilemma there are three players involved: an attacker who is leading the action possessing the ball, we call him "ball owner"; another attacker that may possibly receive the pass, we call him "pass receiver", and a defender that tries his best to not make them score.
The first thing checked by the algorithm is if there is any player who "owns" the ball, that is if its distance to the ball is less or equal than a certain fixed value (a parameter named BALL_OWNER_THRESHOLD).

Among all players within that distance to the ball, the closest one is considered to be the "ball owner".

Then the algorithm checks if the ball owner is in an attack phase, that is if the ball is in the opponent half field, and if the distance of the ball owner to the opponent goal is greater than the distance of the ball to the opponent goal (otherwise the ball would be behind him with respect to the goal).

If those conditions are satisfied then we may call the team of the ball owner as the "attacking team" and the other team as the "defending team".

The algorithm at this points considers all the defending team's players who are closer than the ball owner to their own goal as the possible defenders involved in an hypothetical defender's dilemma.

If the Boolean parameter ONLY_ONE_DILEMMA is set to "True", only one Defender's Dilemma is considered to be possible in each game situation, that means that only the defender that is closest to the ball owner is considered to be part of the dilemma.

If otherwise the Boolean parameter ONLY_ONE_DILEMMA is set to "False", more than one dilemmas are possible, and all the possible defenders found at the previous step are considered in the next steps to check if they are part of their own dilemmas.

If the parameter ONLY_LAST_DEFENDER is set to "True", only the defender who is closest to the goal (without counting the goalkeeper) and the goalkeeper himself may be the defender part of the dilemma. The algorithm checks this condition and excludes the defenders that do not satisfy it.

Then the algorithm checks if there is any attacker team's player that can receive a pass: it has to be closer to the goal than the ball owner (otherwise it would not be really a dilemma for the defender), and it has to be within a certain distance to the companion, as parametrized by the parameter COMPANION_OWNER_THRESHOLD.

The "pass receiver" should also be within a certain distance to the defender, as parametrized by the parameter DEFENDER_RECEIVER_THRESHOLD.

It must be noted that those last two parameters should be big enough to allow the capture of many different dilemmas. Without those thresholds the dilemma would always be possible no matter how far the pass receiver is from the ball owner or from the defender. Those two parameters make possible to experiment with different values of threshold and consequently find a greater or lesser number of dilemma's occurrences.

In addition to this, the trajectory from the current ball position to the pass receiver's position should be clear of obstacles, namely opponent players including their radius (and their leg extension, if any) should not be in the way. The algorithm does something similar to ray-casting to check this.

If all those conditions are satisfied, at the end the algorithm will return one (or more, depending on parameters) Defender's Dilemma, that is a triplet of Defender, Pass Receiver, and Ball Owner (of course the dilemma would also include the ball position, but that is implicit).

Algorithm 3.1 **Defender's dilemma recognition**

```
require: BALL_OWNER_THRESHOLD, real number
require: COMPANION_OWNER_THRESOLD, real number
require: DEFENDER_RECEIVER_THRESHOLD, real number
require: ONLY_ONE_DILEMMA, boolean
require: ONLY_LAST_DEFENDER, boolean

closeToBallPlayers ← All players with distance(player , ball) < ALL_OWNER_THRESHOLD
if (closeToBallPlayers is empty)
        return <Dilemma not found>
end of if
ballOwner ← player in closeToBallPlayers with minor distance(player , ball)
if (ball is in the defending pitch of ballOwner)
        return <Dilemma not found>
end of if
defendingGoal  ←  goal of the same half pitch where the ball is
if ( distance(ballOwner, opponentGoal) > distance(ball, defendingGoal) )
        return <Dilemma not found>
end of if
attackingTeam ← all companions of ballOwner, except ballOwner
defendingTeam ← all opponents of ballOwner
possibleDefenders ← each player in defendingTeam with
        distance(player, defendingGoal) < distance(ballOwner, defendingGoal)
if (ONLY_ONE_DILEMMA)
        possibleDefenders ← the player in possibleDefenders with minor distance(player,ball)
end of if
if (ONLY_LAST_DEFENDER )
        possibleDefenders ← only the player in possibleDefenders that are either the
                defender closest to goal (not counting goalkeeper) or the goalkeeper
end of if
if (possibleDefenders is empty)
        return <Dilemma not found>
end of if
possibleReceivers ← each player in attackingTeam with
        distance(player, ballOwner) < COMPANION_OWNER_THRESOLD
        and
        segment(player, ball) not intersecting any other player
if (possibleReceivers is empty)
        return <Dilemma not found>
end of if
defendersDilemmas  ← empty list
for defender in possibleDefenders
        for receiver in possibleReceiver
                defendersDilemmas.append( <ballOwner, defender, receiver, ball>  )
        end of for
end of for
return defendersDilemmas
```

3.3 Preparatory Analysis

As anticipated, we analysed 1020 games from the official RoboCup archive from the period 2011-2017, and 3015 games from simulations run on our computer using the Helios2018 program against itself. Below we present a table resuming the number of games that ended with or without a winner, and the average number of goals scored .

	2011-2017 RoboCup official archive	**Helios2018 vs Helios2018**
Number of games	1020	3015
Whose draws	167	297
Whose not draws	853	2718
Average goals for winners	4.123	2.543
Average goals for losers	0.447	0.935
Average goals for draws	0.724	1.676

Table 6.1. Games analysed for “Defender’s dilemma” search

We can spot some differences in those descriptive statistics: in the Helios2018 vs Helios2018 games the difference between the average goals scored by winners and the average goals scored by losers and team that have a draw is much smaller than the same average differences in case of 2011-2017 real games. In the Helios2018 games the winners score on average 1.6 goals more than the losers, while in 2011-2017 games the winners score on average 3.7 goals more than the losers. This is something that we could expect, since in the Helios2018 games the two teams are the same, hence they are balanced. The hypothesis that the Helios2018 vs itself games have a lower average goals difference between winner and loser than the RoboCup 2011-2017 games has been statistically tested as explained below in evidence that the two distributions differ.

To check if it is correct to state that the Helios2018 vs itself games have a lower average goals difference between winner and loser than the RoboCup 2011-2017 games, we can do a statistical test, using as samples the 853 goals differences from the RoboCup games and the 2718 goals differences from the Helios games. A “goal difference” is just, for each game that did not end as a draw, the number of goals scored by the winner minus the number of goals scored by the loser.

To choose which statistical test to run, we must check if those 2 samples have a normal distribution and if they have same variance, since some statistical tests require to have sample normality and same variance among samples. We set the significance level at 0.05 .

Running the Shapiro-Wilk test [Shapiro and Wilk 1965] to check normality reveals a P-Value approximately close to 0 for Robocup2011-2017, and P-Value $=7.62*10^{-41}$ for Helios2018. This means that we have strong evidence to refute the hypothesis that both samples are normally distributed, even considering Bonferroni correction [Wasserman 2004 ch.10.7] for 2 tests.

Executing the Levene's test to assess homogeneity of variance [Levene 1960], setting the significance at 0.05, we obtain a p-value of $3.04*10^{-84}$, that is a strong evidence to refute the hypothesis of same variance.

Those findings suggest that to test for same mean we should not use the student t-test, because even if it may be robust to non-normality of samples, we have both a strong evidence of non-normality and also a strong evidence of different variance (that is not something the t-test is robust against).

So we will use Mann-Whitney U test [Hettmansperger and McKean 2011 Ch.2] (known also as Wilcoxon-Mann-Whitney) to assess the null hypothesis that “it is equally likely that a randomly selected value from one population (goals difference in RoboCup 2011-2017) will be less than or greater than a randomly selected value from a second population (goals difference in Helios games)”. Again we set the significance level at 0.05 .

The Mann-Whitney U test results in a P-Value of $6.24*10^{-88}$, giving a very strong evidence against the null hypothesis, and this implies that it is correct to empirically affirm that the goals difference between Winners and Losers in the RoboCup 2011-2017 Games is bigger than the goals difference in Helios2018 against itself games.

Another notable thing, even if still not directly related to the defender’s dilemma, is that despite being less balanced on the average difference of goals between winners and losers, the RoboCup 2011-2017 games ended as a draw in the 16,37 % of cases, while the Helios2018 against himself games ended as a draw only in the 9,85 % of cases. Testing the two percentages as two binomial proportions to check the null hypothesis of same proportion, under a significance of 0.05, a double tailed test gives a p-value of $2.52*10^{-7}$ that is strong evidence to refute the null hypothesis, that means that we instead accept the hypothesis that the results of Helios2018 against itself are distributed differently than the RoboCup 2011-2017 games.

That is notable because we could expect that a team playing against itself is more likely to draw since the two teams are equally strong and play the same set of tactics. Instead, we see here that they have a smaller percentage of draws than the heterogenous games. An explanation for this may be that Helios2018 is an exceptionally strong team (it's the 2018 champion) and so it's exceptionally dangerous during the attack phase: this makes it easier to unbalance the result even in the case of same ability, or in other words, when two equally strong teams play against each other, the equilibrium seems to be brittle if they are strong teams with very effective attacking strategies.

To summarize, this preliminary analysis showed that, confirmed by statistical tests, the games of Helios 2018 against itself have a different distribution both in outcomes and in goals than the games from the official RoboCup 2011-2017 games, and so we may expect differences in the defender's dilemma analysis too.

3.4 Defender's Dilemmas Analysis.

Now let's discuss the findings in the Defender's Dilemmas analysis.
The parameters used for these Defender's Dilemma algorithm runs are chosen to have reasonable sizes with respect to the simulation distance field size. They are the following:

Parameter of the Algorithm	Value
BALL_OWNER_THRESHOLD	2.4
DEFENDER_OWNER_THRESHOLD	16
COMPANION_OWNER_THRESOLD	20
ONLY_ONE_DILEMMA	search both with TRUE and with FALSE
ONLY_LAST_DEFENDER	search both with TRUE and with FALSE

Table 3.2 : Parameters for the Defender's Dilemma search algorithm

The algorithm is run with the parametrization described in the above table 3.2. The algorithm is run 4 times for each dataset, to account for each possible combination of the parameters ONLY_ONE_DILEMMA and ONLY_LAST_DEFENDER, that can be either True or False.
The results are presented in the table below.

	2011 2017	2011 2017	2011 2017	2011 2017	Helios 2018	Helios 2018	Helios 2018	Helios 2018
ONLY_ONE	F	T	F	T	F	T	F	T
ONLY_LAST	F	F	T	T	F	F	T	T
Avg dilemma time Winner	261.750	71.007	48.689	6.032	118.159	30.641	23.399	3.141
Std	142.206	58.583	38.141	8.604	41.065	14.198	12.529	4.087
Avg dilemma time Loser	170.958	39.243	23.196	1.944	110.065	28.474	21.056	2.789
Std	116.890	37.606	24.745	4.438	39.044	13.677	11.861	3.891
Avg dilemma time Draw	237.174	57.829	32.389	2.404	152.557	38.983	28.774	3.737
Std	144.719	51.344	29.417	4.441	44.509	16.021	13.665	4.556
Avg dt/goal Winner	63.484	17.222	11.809	1.463	46.464	12.049	9.201	1.235
Avg dt/goal Loser	381.746	87.628	51.796	4.340	117.686	30.445	22.514	2.982
Avg dt/goal Draw	327.339	79.814	44.702	3.318	90.983	23.249	17.161	2.229
% of games with more dt for winners	68.230	71.512	73.271	60.023	56.291	53.311	53.790	44.849

Table 3.3. Comparative descriptive statistics. “Std” means “standard deviation”. The parameters ONLY_ONE_DILEMMA and ONLY_LAST_DEFENDER vary according to the column.

For each game the software analyses each tick (tick ="discrete instant") and if a team creates one or more defender's dilemmas against the opponent during that tick, a dilemma counter for that team is incremented by one. The statistics “Average dilemma time” is the average number of ticks per game in which a team has created at least a Defender’s Dilemma against the opponent team. It’s calculated separately for winners, for losers, and for teams who tied. The statistics “Average dilemma time per goal” calculates the ratio “Average dilemma

time"/"Average number of goals", separately for winners, for losers, and for teams who tied. This describes somehow a possible "efficacy" of the defender's dilemma: supposing that goals are related to defender's dilemma, a smaller ratio would show that less dilemmas are needed to score a goal, meaning that the team is more able to bring a defender dilemma to success.

Anyway we should be cautious with that meaning: that is based on the assumption that scoring is related to defender's dilemma, that is just an hypothesis that will be tested in the next paragraph. The last statistics is the percentage of games in which the created dilemma time is greater for winning teams than for losing teams. This may give a hint to understand if a bigger dilemma time is associated to a better outcome or not.

To statistically test that the dilemma time for winners, losers, and team who draw, has a different distribution we conducted a series of statistical tests. Since to choose the kind of test we needed to know in advance if the sample is normally distributed, at first we visually examined the histogram of the data (the dilemma time in each game for winners, losers, team who draw) and noticed that they didn't seem normally distributed (see Section 3.5 for the histograms). So we decided to more rigorously test the sample normality with the Shapiro-Wilk test for normality. We grouped all the normality tests under the same "family" and executed a family-wise error correction using Benjamini-Hochberg procedure [Benjamini and Hochberg 1995] [Wasserman 2004 ch.10.7], with a False Discovery Rate of 0.05. The results of the normality test are the following:

	2011 2017	2011 2017	2011 2017	2011 2017	Helios 2018	Helios 2018	Helios 2018	Helios 2018
ONLY_ONE	F	T	F	T	F	T	F	T
ONLY_LAST	F	F	T	T	F	F	T	T
dilemma time Winner	5.73e-24	8.37e-37	1.21e-27	2.25e-36	1.36e-20	1.61e-25	5.01e-29	0.0
dilemma time Loser	2.06e-21	6.32e-33	1.49e-33	1.54e-44	7.89e-22	7.71e-26	6.14e-29	0.0
dilemma time Draw	3.31e-12	4.13e-20	3.80e-18	1.73e-27	3.18e-05	1.42e-09	1.11e-08	7.05e-27

Table 3.4: Shapiro-Wilk test containing the p-value for each normality test

The Benjamini-Hochberg procedure found all those values to be significant, so there is a strong evidence in rejecting the null hypothesis of sample normal distribution for all samples.

We also needed to know if every two samples to compare had homogeneity of variance, and we used the Levene's test, using Benjamini-Hochberg procedure for all Levene's test, with FDR=0.05. The result of the Levene's test as corrected by Benjamini-Hochberg are the following (S=Significative, N=Not significative):

	2011 2017	2011 2017	2011 2017	2011 2017	Helios 2018	Helios 2018	Helios 2018	Helios 2018
ONLY_ONE	F	T	F	T	F	T	F	T
ONLY_LAST	F	F	T	T	F	F	T	T
winner and loser	S	N	S	S	S	N	S	N
winner and draw	N	N	S	S	S	S	S	S
draw and loser	S	N	S	N	S	S	S	S

Table 3.5 : Levene's test for homogeneity of variance

Where the procedure found the value to be "significative" it means that we have to reject the null hypothesis, that means that we have evidence to consider the two samples with different variances.

Now, knowing for each sample that it is not normally distributed, and knowing for each couple of samples if they have the same variance or not, we can conduct the test to assess if they have the same distribution. Where two samples have homogeneity of variance we chose to execute a student t-test: it is robust against violation of normality, but not robust to both violations of normality and homogeneity of variance. We used the paired version when we had to assess the same distribution between dilemma time of winners and losers (since in those games you have a winner and a correspondent loser) and the independent version otherwise (to test "winner dilemma time vs draw dilemma time" and "draw dilemma time vs loser dilemma time").

When we had not homogeneity of variance we used either the Wilcoxon signed-rank test [Mann 2013, Ch.15] to assess if differences are symmetric around zero (in case of dependent samples) or Mann-Whitney U test [Hettmansperger and McKean 2011 Ch.2] (in case of independent samples) that tests if, having two samples X and Y, $P(X > Y) = P(Y > X)$.

All these tests have been grouped in a family to apply the Benjamini-Hochberg correction, with False Discovery Rate = 0.05.

	2011 2017	2011 2017	2011 2017	2011 2017	Helios 2018	Helios 2018	Helios 2018	Helios 2018
ONLY_ONE	F	T	F	T	F	T	F	T
ONLY_LAST	F	F	T	T	F	F	T	T
winner and loser	2.16e-34	2.40e-31	3.39e-51	2.77e-39	4.18e-13	4.79e-09	6.26e-11	0.00058
winner and draw	0.00788	4.01e-09	3.09e-16	4.37e-13	5.61e-65	3.36e-32	3.14e-20	0.00440
draw and loser	1.13e-15	1.05e-12	6.35e-08	0.00113	5.98e-95	2.42e-50	4.65e-39	1.74e-07

Table 3.6 : p-values for same distribution test

The Benjamini Hochberg procedure assessed all the tests as significative, that is to say that we have strong evidence that the dilemma time for winner has a different distribution than the dilemma time for loser and than the dilemma time for teams who draw, and that also the dilemma time for losers has a different distribution than the dilemma time for teams who draw. This is tested valid for every type of defender's dilemma algorithm (that is with every combination of the parameters ONLY_ONE_DILEMMA and ONLY_LAST_DEFENDER) and both for RoboCup 2011-2017 and Helios2018 games.

3.5 Interpretation and Discussion of the Results

Starting from the last row we can see that in real games (RoboCup 2011-2017) in non-draw games, the winner team has on average created more defender's dilemmas than the loser team. It must be taken in account that this fact alone is not indicating that the defender dilemma is the unique factor for victory. Moreover, depending on the way we calculate the defender's dilemma, the teams who created more defender's dilemma ticks were at maximum 73% of times the winners, and at minimum 60% of times the winners. This means that the cases in which the loser teams were creating more defender's dilemma ticks were a minority, but substantial in size (from a minimum of 27% of cases to a maximum of 40% of cases).

If we consider the average dilemma time for RoboCup 2011-2017 games, we notice that Winners created more dilemma time than losers and teams who tied, and teams who tied created more dilemma time than losers, and this is valid in every variation of the defender's dilemma algorithm (i.e. with every combination of the parameters ONLY_ONE_DILEMMA and ONLY_LAST_DEFENDER). The statistical tests refuted the hypothesis that the defender's dilemma time samples created by winners belong to the same distribution of the dilemma time samples created by losers and by team who draw, and refuted the hypothesis that the dilemma time samples created in tie games belong to the same distribution of the ones created by losers. From this we can refute the hypothesis that the association of defender's dilemma with victory is due to chance. Hence we can assert that there is a strong empirical evidence that associates the emergence of Defender's Dilemmas with the success of the attacking team.
The same statistics from the Helios2018 vs itself games are more balanced: there is a smaller difference between winners and losers in the percentage of who created more defender's dilemma ticks. This make sense, because the two equal teams were playing the same strategy and you can expect more or less the same number of defender's dilemmas. Moreover, in Helios2018 vs itself games the average dilemma time created in the draw matches is bigger than the average dilemma time for winners, while in the RoboCup 2011-2017 games the winners on average have always bigger dilemma time of both losers and teams who draw. In Helios2018 vs itself games, in one case, when the parameter ONLY_ONE_DILEMMA is set to True, and ONLY_LAST_DEFENDER is set to True, the average dilemma time for loser teams becomes very similar to the average dilemma time for winning teams, and the percentage of games in which the winner team created more dilemma ticks than the loser team becomes 44.8 %, that means that we have more cases in which the losing team created more dilemmas in a game than the winning team ! This is probably because the numbers of dilemmas ticks are so little with all those parameters restrictions and that the play is so balanced because of the two teams being the same, that a small oscillation by chance may end up having a team doing just some dilemma tick more than the other, and this is enough to mark it as the "team doing more dilemma ticks".

In general, we can see that in the games from RoboCup 2011-2017 there have been more defender's dilemma's ticks than in Helios2018 against itself games, a confirmation that the latter series of games were more balanced. One thing to notice is that the more we restrict the parameters (ONLY_ONE_DILEMMA=True, ONLY_LAST_DEFENDER=True), the lesser are the number of found dilemmas, and this is quite obvious since the conditions become stricter. If we use the maximum restriction, that is both parameters = True, on average each game has very few dilemmas: in the real RoboCup Games 2011-2017 we have an average of 6 dilemmas

tick created by winners and an average of 2 dilemmas tick created by losers, that are very little number in a game of at least 6000 ticks.

The histograms below are the empirical distributions of defender's dilemma time for winners, losers, teams who draw, and the difference of dilemma time for winners minus dilemma time for losers, with the 4 different parametrizations of ONLY_ONE_DILEMMA and ONLY_LAST_DEFENDER.

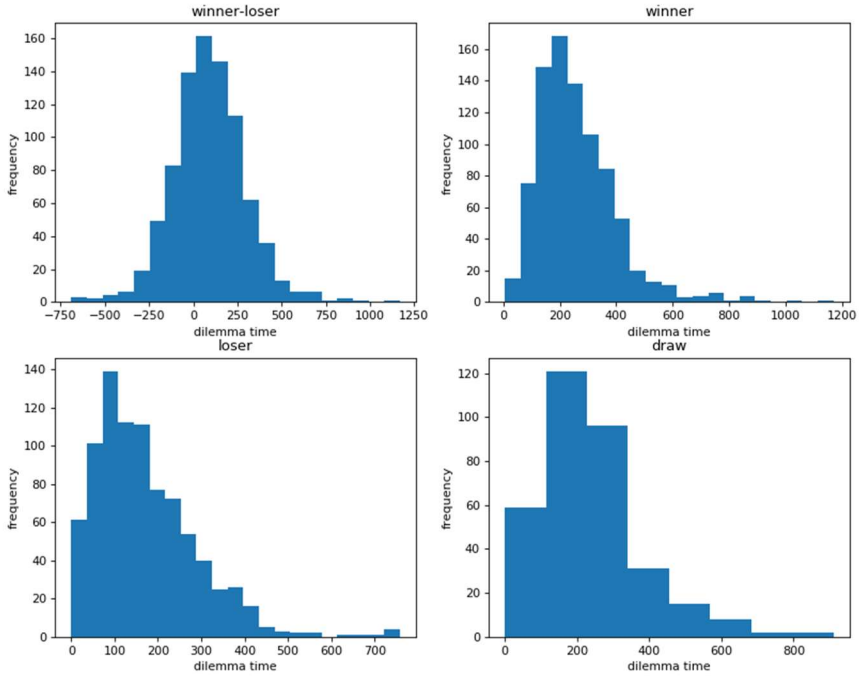


Fig. 3.1 : Dilemma time distribution for RoboCup Games 2011-2017 with ONLY_ONE_DILEMMA=False and ONLY_ONE_DEFENDER=False

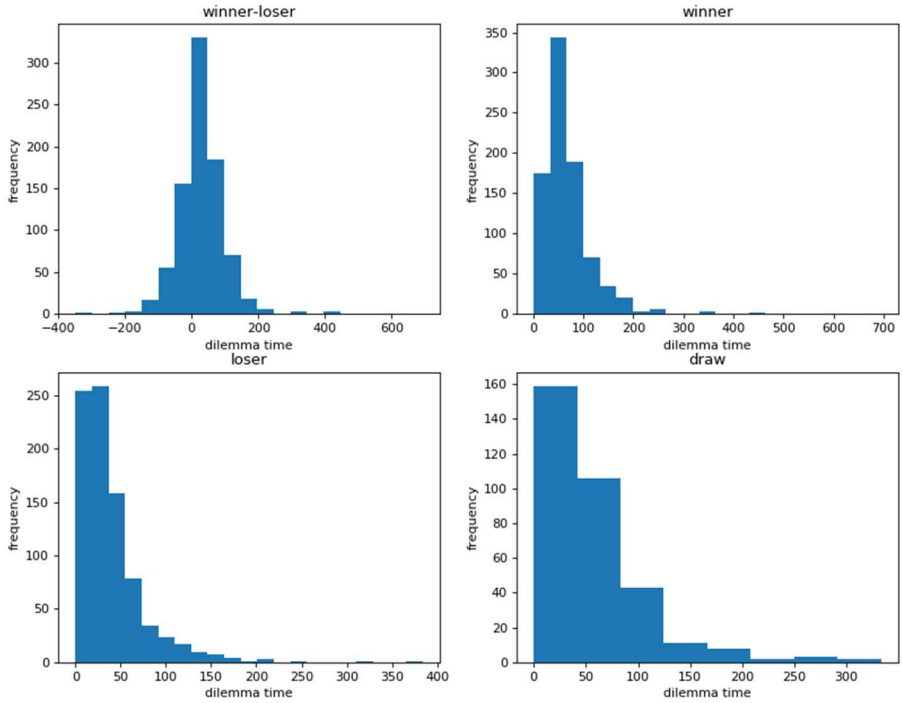


Fig. 3.2: Dilemma time distribution for RoboCup Games 2011-2017 with ONLY_ONE_DILEMMA=True and ONLY_ONE_DEFENDER=False

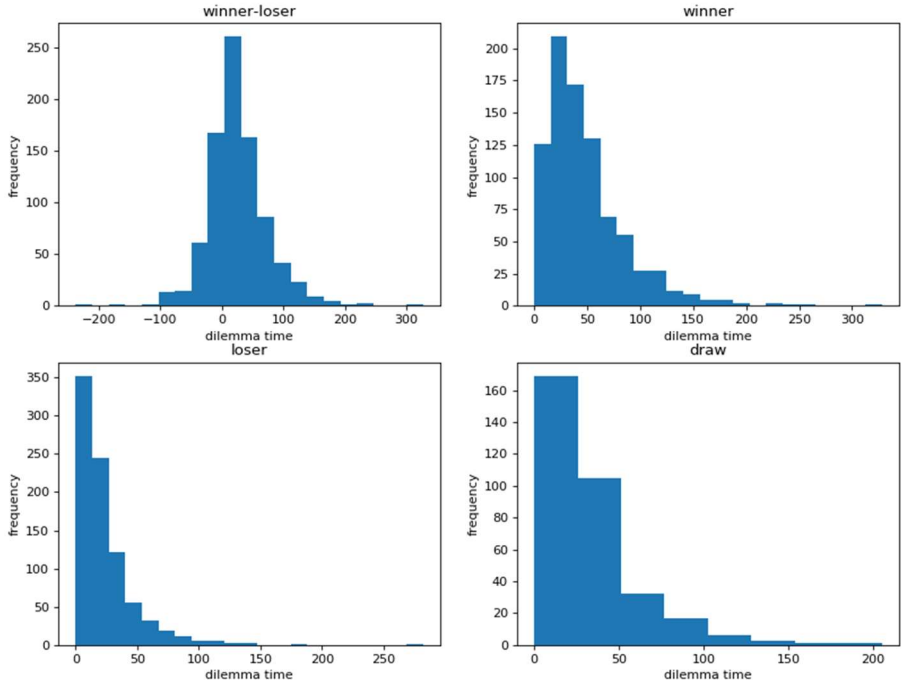


Fig. 3.3 Dilemma time distribution for RoboCup Games 2011-2017 with ONLY_ONE_DILEMMA=False and ONLY_ONE_DEFENDER=True

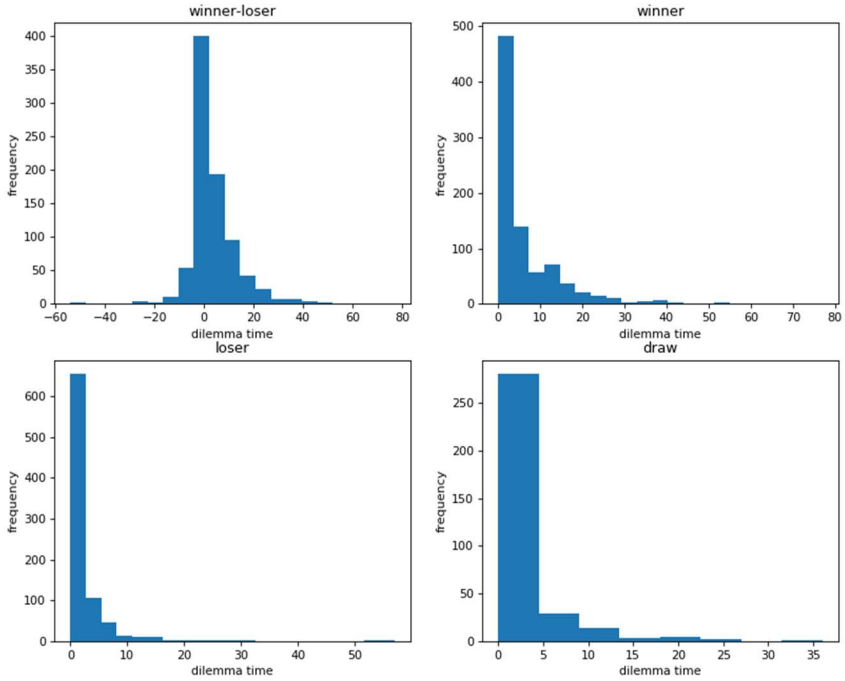


Fig. 3.4 Dilemma time distribution for RoboCup Games 2011-2017 with ONLY_ONE_DILEMMA=True and ONLY_ONE_DEFENDER=True

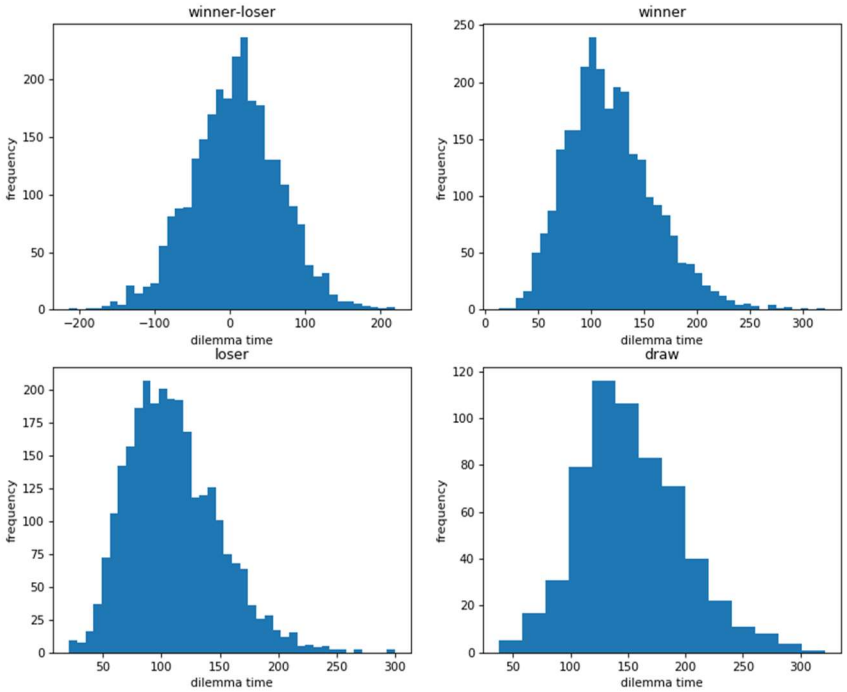


Fig. 3.5 Dilemma time distribution for Helios2018 vs Helios2018 with ONLY_ONE_DILEMMA=False and ONLY_ONE_DEFENDER=False

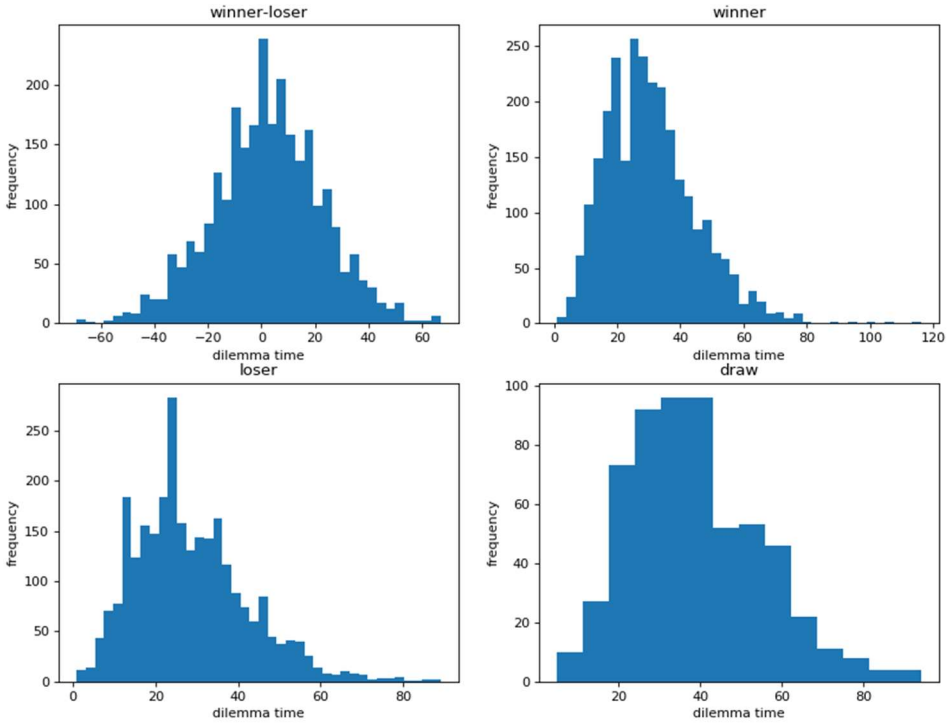


Fig. 3.6 Dilemma time distribution for Helios2018 vs Helios2018 with ONLY_ONE_DILEMMA=True and ONLY_ONE_DEFENDER=False

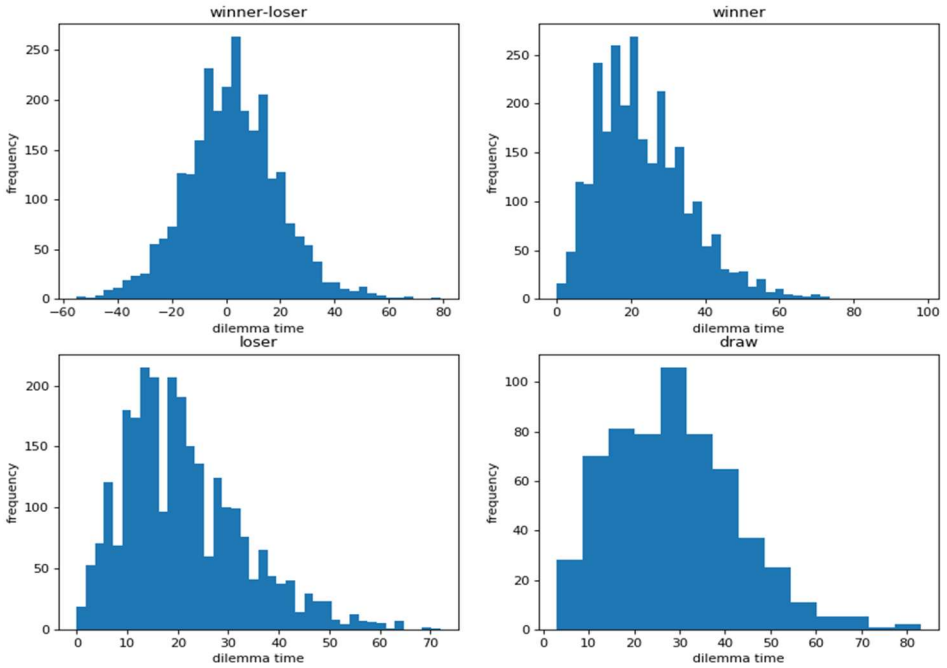


Fig. 3.7 Dilemma time distribution for Helios2018 vs Helios2018 with ONLY_ONE_DILEMMA=False and ONLY_ONE_DEFENDER=True

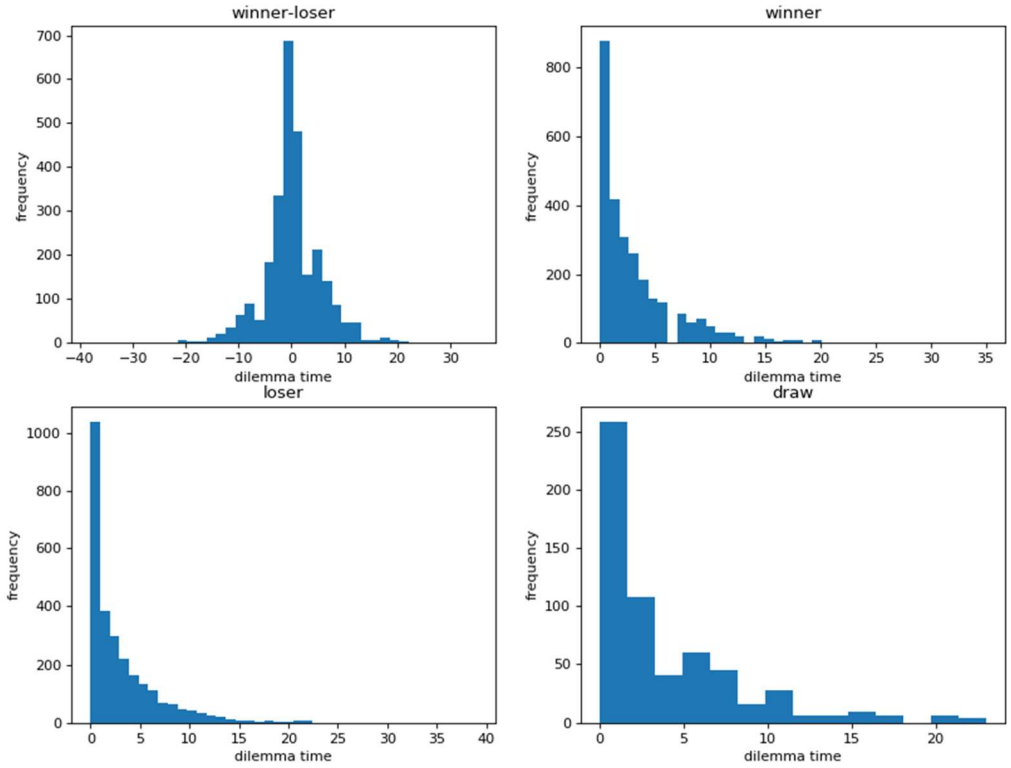


Fig. 3.8 Dilemma time distribution for Helios2018 vs Helios2018 with ONLY_ONE_DILEMMA=True and ONLY_ONE_DEFENDER=True

3.6 Summary

This experiment shows a strong association between the emergence of the Defender's Dilemma and the success of a team: the team that is able to create more Defender's Dilemmas against the opponent's defence has more chances of winning. This is confirmed by statistical tests. Nonetheless, the association with success is not total, since in a considerable minority of games the loser was creating more dilemma time. This means that also other factors are influencing the success of a team. In conclusion, our contribution with this experiment is threefold: we statistically tested the association of the Defender's Dilemma with a positive outcome, we developed a fast and efficient algorithm to detect Defender's Dilemma occurrences, that can be used in real-time, and we showed how an emergent dynamic structure that can be described by hypernetworks relationships may be actually defined by an algorithm.

Chapter 4. Value Function Perspective

4.1 Introduction

The second research question that we defined on paragraph 1.3 asks if “Is it possible to forecast with a good degree of accuracy if a certain game action or sequence of game actions is going to be successful, before it has been completed ?”. In this chapter we will try to find an answer to that, using data from RoboCup 2D games and neural networks.

4.2 Two Ways to Analyse a Soccer Game

One way to analyse a soccer game may be to try to understand the typical way in which the opponent is playing and so the focus would be on identifying the repeated regularities in the opponent actions, that is to say it would be a pattern recognition/pattern learning task, or using the Hypernetworks terminology, it would be a “Hypersimplices learning task”. But another way could be to just try to understand “how much” the current game configuration is “good or bad”.

This second approach could then be used by the team as an advisor when taking decision: just avoid going into situations where the game can be “bad”, and take actions in order to go into situations where the game can be “good”. One way to evaluate the goodness of a situation is to use Machine Learning methods. As we wrote in Chapter 3, some of those methods, such as the Deep Learning ones, are known to be able to automatically learn relevant relationships inside the data in a way that is useful for the final task. More in detail, Deep Learning methods can do “representation learning” (also known as “feature learning”) [see Goodfellow et al. 2016, chap. 15] that means that during the training the intermediate neural layers of the network will self-organize in a way that makes them able to capture “concepts” that are of higher-level with respect to raw data, if those higher-level concepts are useful to solve the wanted task (again, in Hypernetworks terminology that would mean learning the Hypersimplices that contain the relevant relationships).

This means that if we train a neural network with the intent of recognizing if a game configuration is good or bad, hypothetically it may happen that the neural network in its internal layers will be able to convey information useful for that purpose, such as recognizing different tactics of the opponent team. A network that learns to do so, then can partially be reused to switch task, such as directly recognize tactics or patterns or relevant game information, instead of just telling if a situation is good or bad. Using a trained network to do a different task, taking

advantage of its ability to represent relevant features, and then finetune the network for the new task is called “transfer learning” in Deep Learning literature.

To sum up, there can be two ways to analyse a soccer game: the first is to directly look for patterns, the second is to evaluate the goodness of a game configuration. If we use deep learning methods in this second way to evaluate the goodness of a situation, then potentially we could also do “transfer learning” and reuse part of the neural network to analyse the game in the first way, that is to find tactical patterns.

It turns out that evaluating if a situation is good or bad through a scoring is very similar to what is known as “Value Function” in Reinforcement Learning literature. Hence, looking at our research topic through a reinforcement learning perspective may be fruitful, even if this does not automatically mean to really apply any Reinforcement Learning algorithm. In the following paragraph I will give a quick explanation about what contribution a Value Function can bring to our research.

4.2 Value Function Analytical Perspective

As seen in detail in paragraph 2.4, in some Reinforcement Learning algorithms, an indicator of "how much a situation is good or bad" is the Value Function: it’s a function, whose approximation is to be learned by the agent through “trial and error”, that takes the description of a state as input and returns a numerical value that describes how much that state is good or bad. In our case a value function could evaluate a numeric answer to the question "are these players positions in the field or recent movements likely to give a good outcome or a bad one ?”.

It has to be recalled that Value Functions are dependent on an agent’s policies: two different policies will determine generally two different values function. So, for a certain configuration of the players/ball on the field, we cannot have a “universal function” that gives us a measure of how much that configuration is good or bad. That measure is always dependent on the policy of our team (what my players are going to do in every situation) and on the policy of the other team (what the opponent is trying to do in every situation). This implies that a Value Function is not valid “in general”: different policies for our team and for the competitor will give rise to different Value Functions. And unless we know exactly the policy of the opponent (and

every dynamic change he makes to it), it may not be possible to compute the true Value Function, but only an approximation of it.

In the experiment that we are going to describe in the following paragraph we used something very similar to the Value Function described above, even if we are not developing a Reinforcement Learning system. Nonetheless we will develop something that works similarly to the Value Function (in fact we call it precisely “Value Function”): a function that is able to evaluate the current state and return a numerical value that expresses how much the state is good. This function will be developed using a supervised learning system, more specifically an artificial neural network.

Our “value function”, just like the one in reinforcement learning systems, is dependent on the policies/strategies of the two teams and so it is not generally applicable to each possible policy. Anyways, even if the resulting value function is dependent on the specific policies, if it turns out to be reliable it will show that the methodology works and that a deep learning system is able to detect when a certain game configuration is “good” or “bad”.

It must be noted that the search for this kind of Value Function calculated from a given behaviour may seem somehow related to the problem of “Inverse Reinforcement Learning”, in which, given the behaviour of agents, the goal is to calculate their “reward functions” [Ng and Russell, 2000], [Arora and Doshi 2021]. Inverse Reinforcement Learning is anyway a different problem to be solved, and it is more complicated, here we want just to calculate the goodness or badness of a situation, not the complete structure of rewards received by agents as in Inverse Reinforcement Learning.

Our experiment consists in processing a certain number of RoboCup 2D games and trying to compute a Value Function. To simplify things, we used the same fixed policy for both teams, that concretely means that we made a team play against another version of itself. More precisely, we used the HELIOS2018 team, the winner of RoboCup 2018 competition [Akiyama 2018], whose executable is freely available [web reference 6], and we ran it for both teams.
The RoboCup server used was the version 15.5.0. [web reference 5]. We ran 3015 games with standard settings and collected the log files containing all positions and actions of players to analyse them. During the running of the games, we discovered two bugs occurring under some conditions during the execution of penalty kicks, we then identified and corrected those bugs in the source code and pushed the fix to the official repository on GitHub where it was accepted by official RoboCup maintainer and included in the official version. Since our

experiment did not analyse penalties (which occurred only in penalties shootouts if teams drew after extra time), the presence of the bug before our fix did not influence our analysis.

We wrote a script to preprocess the 3015 games selecting only the game sequences in which the game was in active play, and excluding the other game sequences, such as when players were moving to position before a corner kick, a kick-off and so on.
The script also separates the different game action sequences: a sequence begins when the game was previously not active (such as before the beginning, before half time, before corner kick etc.) and ends when the referee “whistles”. Each action sequence is labelled either as “goal” if it ends with a goal score, or as “not goal” if it ends without a score.

Then we used a feed forward neural network as a function approximator, with the aim of learning to approximate the value function, or in other words to evaluate the “goodness” or “badness” of a certain game configuration.

The desired behaviour of our value function approximator would be to have the neural network receiving the game configuration as input and answering with a number that is =1 in case of good configuration, and =0 in case of bad configuration, and every intermediate value would have a proportional intermediate meaning.

Our neural network is not exactly like that but something similar: it receives a game configuration at some time t as input and tries to answer with a number =1 if it forecasts a goal score or =0 if it forecasts an unsuccessful outcome.

This is clearly a way to assess how much the configuration is good for our team to score a goal. To evaluate if the opponent team is going to score instead, we can use the same neural network and insert the same input with the players of the 2 teams swapped in the input slots and with the coordinates mirrored (or better: symmetric with respect to the origin).

In this way we use the forecast of the goal score as a proxy for “good/bad” configuration: the more the forecast indicates a scored goal the more the situation is good.

The output of the neural network has to be considered as a probabilistic value: it expresses a value between 0 and 1, and that value is the forecasted probability of having a goal scored by the first team at the end of the action sequence. So, every value >= 0.5 will forecast a goal, and every value < 0.5 will forecast and ending without a goal, and the value itself will represent

the “goodness” or “badness” of the configuration: the closer to 1, the better is the configuration, the closer to 0, the worst is the configuration.

A game configuration of a certain instant (“tick”) of the game is composed of 6 numbers for each player:

- x coordinate
- y coordinate
- x velocity
- y velocity
- body angle
- head angle

In addition to this, there are 4 numbers for the ball too:

- x coordinate
- y coordinate
- x velocity
- y velocity

So with 22 players in the pitch we have in total 136 numbers that describe the configuration of an instant of a game.

Obviously, the closer the analysed data point is to the end of action, the more relevant it will be: closer to the scoring (or to failing) it will be clearer if there is going to be a score or not. So, a system that recognizes if a situation is good or bad should not only be working well with close-to-the-end data points, but also with data that are far from the end of the action sequence.

If the instant in which the action sequence ends is T, and we use the configuration that is distant n ticks to the end, this means that we are using the tick at time $T - n$.
We ran 6 different neural networks trainings with different time distances with respect to the end of the action sequence.

In all experiments the neural network was a feed forward neural network with 1000 Rectified Linear Unit neurons in the first hidden layer, 200 Rectified Linear Unit neurons in the second layer, and one neuron with sigmoid (logistic) activation function as output.

In the first experiment the input was from two consecutive ticks: so we had 2*136=272 input neurons, while in all the other experiments we had only the game configuration of 1 tick as input, so 136 neurons.

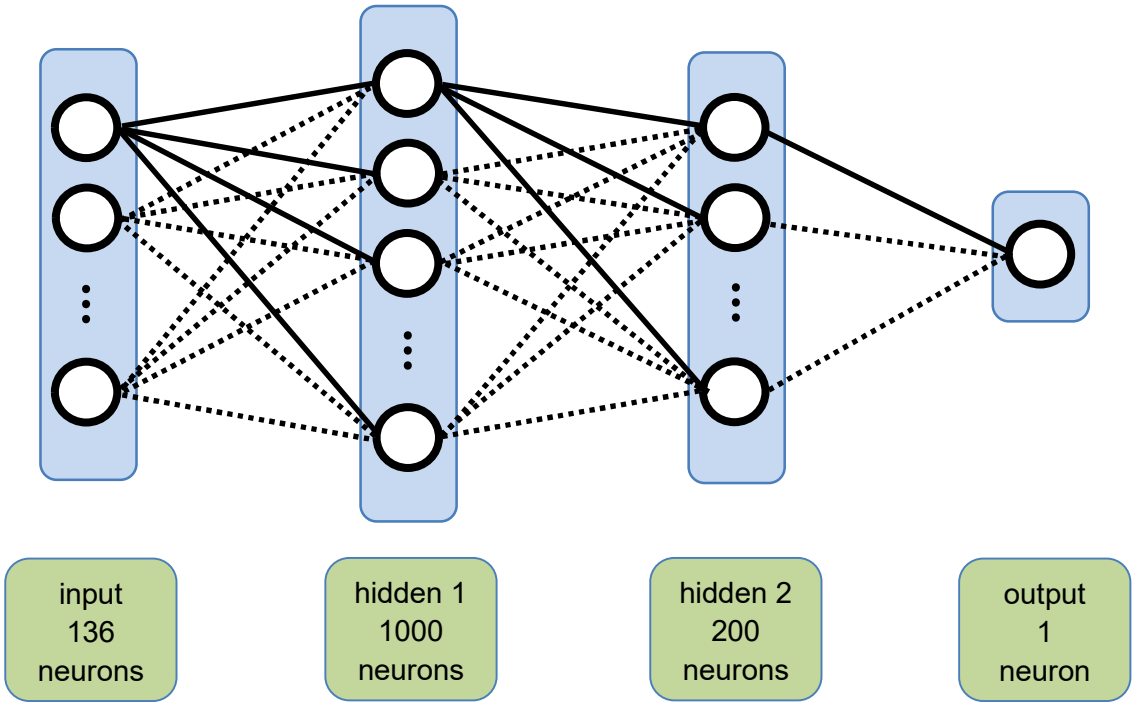


Figure 4.1 : the Neural Network layers used for forecasting (in the first experiment the input layer was composed of 272 neurons instead of 136).

We used a gradient descent optimization with training batch size of 4 samples, learning rate is fixed at $1*10^{-4}$, and the binary cross entropy loss function, typical for logistic outputs (see equation 5.15). We trained the network for 100 epochs and then tested the accuracy.

In all experiments the training, the validation, and the test set were class-balanced: in each set the number of samples ending with goal scoring was the same as the number of samples ending with no goal. This was necessary since otherwise the occurrences of sequences leading to goal would be very rare in comparison with the occurrences of sequences ending without goal, and any classifier could be potentially biased in guessing always “no goal” and still obtain a good degree of accuracy. So we used an equal number of goal ending sequences and not-goal ending sequences in each dataset (be it training, validation, or test set).

This implies that we used only a fraction of the samples available: we used all the samples with actions ending with goals, but we had to use only a part of the actions ending with no goal. We dedicated 70% of the samples to the training set, 15% to the validation set, and 15% to the test set. The validation set was used only to monitor learning but since we did not do any hyperparameters optimization and we did not use the loss on the validation set as indicator when to stop training (typically when the loss function on the validation set stops to decrease, the learning has to be stopped to avoid overfitting) because we wanted to compare all the results of the different experiments with the same level of training. This is the reason why we used a fixed number of 100 epoch for the training of each experiment.

In the first experiment the input was from the tick at time $T-10$ and the tick at time $T-11$. Since in RoboCup 2D usually the duration of a tick is 1/10th of a second, we are using samples a little more than a second before the ending of the action sequence.

The first experiment was run only to answer to the question if using 2 ticks instead of 1 as input would have benefitted the learning. Since we were interested only in the total accuracy to compare it with the total accuracy of the second experiment we computed only that, while from the second experiment onwards we computed the total accuracy and the two accuracies of each class (goal/not goal).

The second experiment revealed that using only 1 tick was obtaining slightly better results, so from that moment onwards we used always only 1 tick as input.

In the experiments 3 to 5, at each experiment we increased the time distance of the tick to the end of sequence by 10 ticks. If the distance to the end is n, we require the action sequence to be at least n+1 ticks long. Increasing the distance of the tick hence implies requiring longer sequences of actions. Unfortunately, as the length increases, the number of available sequences with that length is decreasing. For this reason, each time we increased the time distance we had a smaller number of samples available, a thing that may have limited the training efficacy.
In the last experiment (the 6th) we used samples from mixed distances: some from $T-11, T-21, T-26, T-31, T-36, T-41, T-46$. This is to train a network in different time-distances, that does not implicitly know when the action sequence end. In this case, for each sequence, up to 7 samples may be obtained.

Experiment	Time	Goal Sequences	Total Accuracy	Accuracy in goal actions	Accuracy in non-goal actions
1	-11, -10	9747	0.913	n.a.	n.a.
2	-11	9747	0.925	0.935	0.916
3	-21	9581	0.879	0.876	0.883
4	-31	9178	0.849	0.847	0.851
5	-41	8830	0.832	0.813	0.851
6	-11 to -46	8690	0.865	0.852	0.879

Table 4.1 : Accuracy in each experiment, after 100 epochs of training

4.4 Analysis and Discussion of the Experiment

4.4.1 Two Ticks vs one Tick

It looks that using 2 ticks per data point does not give any better information to the neural network with respect to only 1 tick. After all, the data related to 1 tick already includes the velocity of the players and ball, so it is already a dynamic information, so using 2 ticks adds only redundant data, and has the inconvenient of doubling the number of weights to learn in the first hidden layer.

In fact, using only 1 tick seems to give a slightly better performance, probably because the resulting neural network has less parameters and it's easier to fit with limited data.

4.4.2 Relevant Findings

It seems that the system in our experiment can forecast if an unseen before action sequence is going to be goal or not with an accuracy of more than 80% even 4 seconds before the ending of the action sequence. This is a very high level of accuracy of prediction, especially considered the difficulty of the task. The neural network is capturing some relationship in the game configuration that is indicative of the success of a configuration, we can say that there is an hypernetwork relationship implicit in the game state and the neural network is capable of recognizing it.

4.4.3 Similar Studies

After having conducted this experiment in 2019, we became aware of the existence of another research with some point in common. [Pomas and Nakashima 2019] aimed at using neural networks to evaluate the goodness of a situation in RoboCup 2D. Differently from our experiment, they only evaluated situations prior to goals. So, their system was able to guess which team was more likely to score, on the condition than in 100 ticks either one team or the other would have really scored. Our system, in contrast, is trained both with sequences of actions ending in a score and with sequences of actions ending without scores. Even the type of prediction is different: our system may forecast if a certain game configuration would end up in a goal scored for a certain team, a goal scored for the other team (applying the input data in mirrored form), or no goals scored.

They used a Convolutional Neural Network applied to the 2D image of the pitch in the simulation to test the capacity of CNNs but also, as a benchmark, used feedforward neural networks and other machine learning methods such as bagging trees and random forests applied to numerical data (not on the image). In a subsequent study [Suzuki and Nakashima 2019] did the same experiment with a feedforward network applied to numeric data, using as input data not only the current situation but the output of a recurrent neural network that was forecasting the positions three ticks in the future. This gave a slight improvement with respect to the usage of only past and current situation in the input.

4.4.4 Discussion

The findings in our experiment are unprecedented and the level of accuracy for such apparently difficult task is high. This shows the potential of neural networks in analysing RoboCup game configurations. Nonetheless, our method (as well as the previously discussed neural networks methods in [Michael et al. 2018], [Zare et al. 2018] and [Copete et al. 2015]) suffers from what I defined “fixed order input limitation”: switching the input position of a player may lead to a decrease in performance, because the weights of the network are associated to a certain order of the input, limiting the generalization capability. To avoid this limitation, input data should not be coupled to a particular order of agents. This may be achieved using pictures of the pitch instead of directly using the coordinates, an input format used by [Pomas and Nakashima 2019]. Using 2D images as input increases enormously the quantity of data to process, and implies using convolutional neural networks instead of fully connected layers such as in our experiment. To have a comparable level of accuracy in that case it would be necessary to have a much greater number of samples and greater computational resources.

4.5 Summary

In this experiment we used a neural network to forecast the outcome of a game configuration. Using the position and velocity of players and ball at a certain point in time, previous to the end of the action sequence (from one second to four seconds and a half before the ending) as input, the network has been able to forecast with good accuracy if the action sequence was ending with a goal or not. This is a novel result in the RoboCup literature. The fact that each player has a fixed place in the input data slots (“fixed order input limitation”) may affect performance and generalization, and leads the way to the future improvement of the method with convolutional neural networks, which do not depend on the input order of players.

Chapter 5. Robotic Soccer Simulator and Experiments

5.1 Introduction

To answer the third research question, that is about the possibility of make behavioural pattern emerge without specifying the rules in details, we decided to use Reinforcement Learning, because it is a method in which the behaviour of the agent is learnt through trials and errors and does not follow a fixed set of rule. The need to have a simulation that allows focusing on agents' interactions, rather than focusing on how agents sense the world and model the sensory inputs, along with the need of having fast simulation iterations because RL needs a big number of runs [Irpan 2018], motivated us to write our own robotic soccer simulator.
So we wrote a simulator called "Very Simplified 2D Robotic Soccer Simulator", or in short with its project name "robosoc2d".

The simulation computes simplified 2D physics and it is realized in optimized C++, hence it has a very fast execution time. Moreover, since it is turn-based and not real-time (but it may be possibly run in real-time using waiting intervals), it gives the possibility to run seeded deterministic experiments and it can execute simulations at maximum speed without waiting for timed interactions. The simulation time for a match (6000 ticks) may take as little as one second for core/thread or less, in a common current notebook CPU. Obviously if agents with high computational requirements run on the same core where the simulation is running it will take more time. This is a quite vague benchmark, but suffice to say that other simulators are much slower, both because they simulate more details and because they are real-time. RoboCup 2D server for instance, if run in real time, always needs 600 seconds for a match, while if run with different tick duration it still needs at least 1 millisecond per tick.

Another reason that makes our simulator faster is the usage of simplified 2D physics: some simulators use advanced 3D physics computation and hence are slower, such as DeepMind MuJoCo Multi-Agent Soccer Environment [web reference 12], or RoboCup 3D simulator (based on SimSpark) [web reference 15], as well as GrSim, the simulator for RoboCup Small Size Robot Soccer [Monajjemi et al. 2012] and the two simulators for IEEE Very Small Size Simulation (FIRASim [web reference 13] and VSS-SDK [web reference 14]). For these reasons our simulator may be a better choice to be used as Reinforcement Learning testbed: faster execution times imply a greater number of simulations, and Reinforcement Learning in

general needs a big number of samples. Some environments, like Google Research Football [Kurach et al. 2019], use only high-level actions and automatize parts of the game, so they actually simulate a very different type of game/environment/problem.

Within this perspective the aim is not to simulate a lot of environmental details to make it as much realistic as possible, but to make the environment as simple as possible and provide complete state information about the world to agents. The simulator is experiment-oriented rather than competition-oriented: it aims at generating simulations as fast as possible in order to have a good amount of experiments to analyze quickly. To do so, as we anticipated, simulation is not in real time but it is turn-based: so the simulation does not have to wait a certain amount of time before simulating the next step, and this can give a great speedup. For the same reason, agents do not communicate with simulator by TCP/IP connections: agents are compiled together with simulation code and communicate only by function calls. This means that this simulator is not designed for competitions because of two aspects that are demanded to agents and cannot be controlled by simulation: (1) time limits on computations (the simulation is waiting for an action from the agent, without control retrieval), (2) security issues (the simulation may run on the same process of the agents and a malicious agent may easily try to hack it). This does not mean that competitions cannot be held, but in case, it is necessary for teams to provide source code to be inspected, and that agents be self-limiting their computation time, or to write an additional software layer to have and independent simulator server. On the other hand those characteristics make the simulator fit for massive amounts of simulations that scale with processing power, so it can be proficiently used to scientifically study the behavior of agents or to train Reinforcement Learning agents, and it turns out to be very fast even on modest hardware (single core, low frequency).

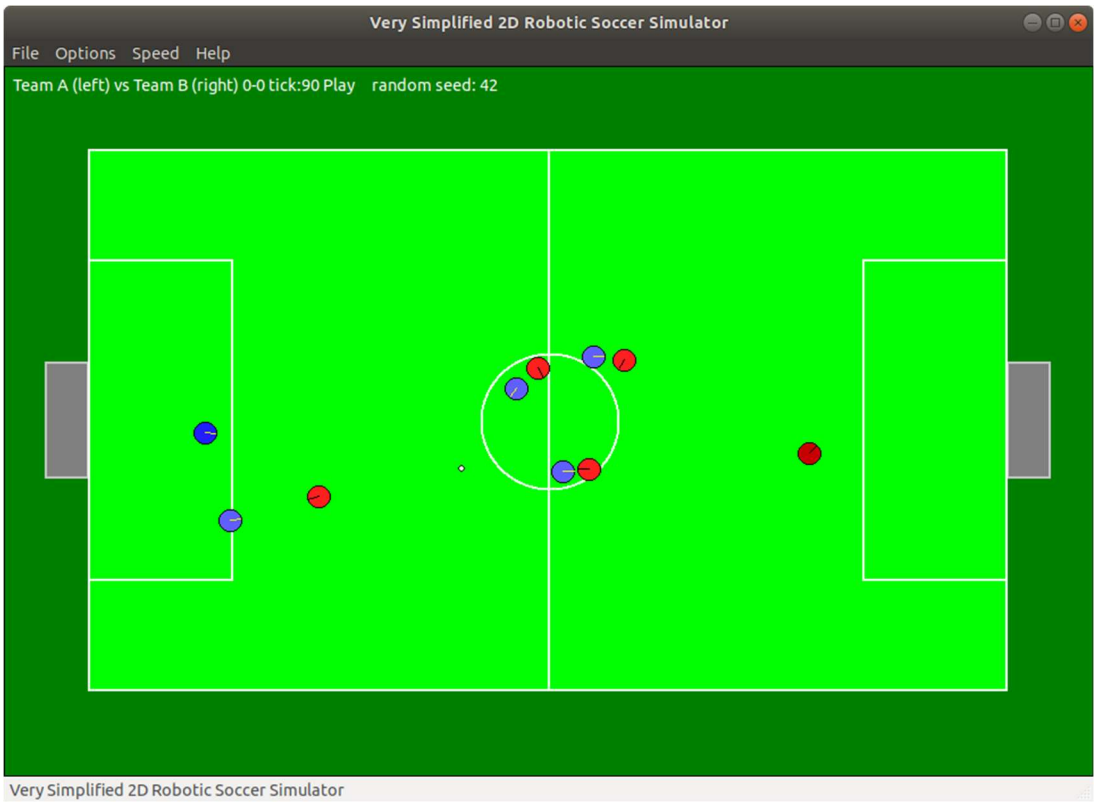


Figure 5.1 : the Very Simplified 2D Robotic Soccer Simulator on Ubuntu Linux 18.04, C++ version. A 5 vs 5 game. Left team is in blue, right team is in red, darker players are goalkeepers (only in this version of the program. In Python GUI goalkeepers are visualized with same color as their companions). The segment inside each player indicates the player's direction

5.2 Architecture and Implementation

5.2.1 Simulated Environment

The simulated environment is a 2D world were the pitch extends its length horizontally and its width vertically. The center has coordinates $(x, y) = (0,0)$ and the horizontal axis (x) increases toward right and decreases (becoming negative) toward left, and the vertical axis increases toward the top and decreases (becoming negative) toward the bottom. The size of the pitch can be set at initialization time to any preferred value, otherwise default settings will apply. Also the number of players can be chosen during initialization (a different number of players for each team is possible, including zero).

Players are supposed to be small robots moved by wheels, able to turn left/right up to 90 degrees and to move forward and backward. This means that at each command they can turn towards every possible angle (complete 360 degrees choice). They are moved by an engine through a "DASH" command that transmits an acceleration to the robot in the current direction.

A simplified acceleration and friction calculation is computed by the simulation, both for robots and ball. A simplified mechanism for collisions is computed too. There is a limit to the maximum velocity that robots and ball may achieve. Robots have a mechanism to kick the ball (“KICK” command) if it is within a certain distance to the robot and within a certain angle to the robot direction. The first robot for each team is a goalkeeper and has the special mechanic to try to "catch" the ball when it is within a certain range (the "CATCH" command can be used only inside a goalkeeper own area). The catch mechanism is not deterministic and has a probability of success. When a goalkeeper manages to catch the ball it holds the ball for a limited number of ticks and during that time other players are not able to kick the ball or take it away from the goalkeeper.

Robots do not exhaust their battery so they can always work at full power (this is very different from the popular Robocup2D Soccer Simulator Server). When the action is interrupted and the game is set to a restart, such as during kick-off, corners, and so on, agents can be positioned directly to a certain position in the field with the “MOVE” command, that is limited in reach (an agent cannot move more than the longest movement made by the players of the same team of the agent that is kicking the ball after the interruption).

Robots know the full state of the world, since a complete information state is given to them. This exempts the player agent to the duty of building an internal representation from incomplete and uncertain information and allows the researcher to focus on other aspects of intelligence. But it is still possible to use this simulator to study partially observable states: it just needs to be added a function that processes the complete information state and transforms it in a partially observable state before sending it to the rest of the agent computation pipeline.

The rules of the game are quite similar to the ones of real soccer, with the exceptions that, in order to keep the strategy space as basic as possible, there is not the offside rule, and a player can pass the ball to himself during a corner/throw in/goal kick/kick off. Moreover there are not faults nor penalties. Corners, throw-ins, goal-kicks and kick-offs exist, and during their execution there are mandatory distances to be respected by opponents.: each time an event like corner, throw-in, goal kick or kick-off happens, opponent players are required to move in a location that respects the distance, with the command "MOVE". If the distance is not respected, the simulation will automatically displace the player to the right distance. The match is composed of 2 half times (just like real games), the duration is typically 3000 ticks per half time (but may be set to any value), with a total of 6000 ticks. The time duration of a tick is considered to be 1/10th of a second, or in other words 10 ticks happen in a second, a match

hence is supposed to have a duration of 600 seconds or 10 minutes. That is just an hypothetical convention, since the simulation is turn-based and not time-based, and runs at maximum speed, that implies that the hypothetical time duration of a tick does not really matter. After the first half time the teams switch sides of the pitch to play the second half.

5.2.2 Implementation

The simulator itself is realized in C++ 17 and may be compiled stand-alone or added to another C++ project. There is also a Python extension that allows installing the C++ simulator as a Python package and use it in Python, with a similar API structure.
The C++ package uses CMake (web reference [20]) as project automation and this allows compiling the simulator for Linux, MacOs and Windows platforms without any modification in the source code.

The Python extension too is compilable for the 3 platforms, Linux, MacOs, and Windows.
The C++ version needs to have wxWidgets GUI library installed to have a graphical user interface, but the simulation part may be compiled and run even without it.

Similarly, the Python library needs to have the library Tkinter installed to have the graphical user interface. Tkinter is the standard gui library for Python and is usually present in each Python installation, except those explicitly built without it such some server-only distributions. Iif Tkinter is not present the simulator can still be run without graphical interface. The Python version has also a library to visualize the game through Matplotlib, a visualization that may be in static picture in desktop programs or dynamic in Jupyter notebooks (it may not work in some Jupyter derived products).

5.2.3 Architecture

The C++ simulator class, named `R2Simulator`, is instantiated passing to it a vector of players for the first team, a vector of players for the second team, and optionally the teams' names and a seed for the random number generator to make each experiment reproducible (if missing, a seed will be generated from current time).

The players, instances of classes derived from the `R2Player` class, must implement the method `Step()`, because the simulator will, at each tick, call that method of each player, passing the current world state as parameter and obtaining as a return value the action that the agent has chosen to execute.

The Python version follows a similar approach.
Usually the simulator calls the method `Step()` of players in random order, but in some occasion the order is fixed and determined by rules, such as when the game is restarted after an interruption (as in a corner, throw-in etc.) where the team that must restart acts first.
As anticipated, Players can choose 4 actions, plus an action that is a placeholder for not acting.

The actions are:

- Dash. It is used to accelerate the robot towards a direction, it is the action used to move the robot in the pitch. The agent must specify the direction angle (in radians) and power of dash.
- Kick. Its effect is to kick the ball (if it is in range). The agent must specify the direction angle in radians and the power of the kick.
- Catch (only for goalkeeper). It makes the goalkeeper try to catch the ball (if it is in range) and hold it temporarily
- Move. It is used to relocate the player in a certain position, usable only when the game is stopped by the referee and must be restarted. The agent must specify the coordinates of the position and the direction angle in radians of the player.
- NoOp. It means no operation, and is used when an agent decides to not take any particular action.

The simulation is computed at discrete events, but to avoid to miss collisions between players, ball, and goal's poles, that may go unnoticed if the discrete step in movement is too large with respect to the tick duration (too big time granularity), collision are computed in a complete way with time-continuous parametric geometry intersections, so to not miss any collision or bounce.

A simplification of the simulation inner loop is reported below, to present a general idea of the simulation mechanics. The names of real methods are used, but they are not exhaustive and many more methods are called but not reported for simplicity.

Algorithm 5.1 **Robotic soccer simulation inner loop.**

Note: the main loop of the simulator is contained in the method R2Simulator::step(). It consists of executing the following methods, in this precise order, at each iteration:

preState()	This sets some aspect of the environment depending on the game state. For instance it positions the ball at the center of the pitch in case of kick-off, or in the exit point in case of throw-in, etc. It will also set some internal variable depending on the simulation.
playersAct()	For each player, with the order of players decided randomly (or with a fixed rule in restarts): • calls the method step() of the player • processes the action returned by player with processStep(). This will add an acceleration to the player in case of Dash, or will add an acceleration to the ball in case of Kick, or will change position to player in case of Move etc. in general it will change some state of the environment.
limitSpeed()	It limits the speed of players and ball that reach the maximum
limitPlayersCloseToPitch()	It limits the position of players to the enclosure of the pitch
checkState()	It positions the players to the correct distance if they are too close in case of kick-off, throw-in, goal-kick or corner. Then, depending on their velocity, players and ball are moved, collisions are computed among them and against the goal poles, in a time-continuous computation, and players and ball are adjusted in position and speed accordingly to collisions. It is checked the occurrence of goals or if the ball exits from the pitch.
decaySpeed()	Being the end of the tick, the speed of players and ball are decreased.

A useful feature of Very Simplified 2D Robotic Soccer Simulator is the possibility of setting the environment (the game situation) in a certain configuration chosen by the user, such as deciding the positions and velocities of players and ball, the time tick, the current score. This is useful to developers of agents that may test the behavior in a certain well-specified situation, and to researchers of Reinforcement Learning systems, that may recreate some specific state for learning purposes (as in Curriculum Learning).

An example agent, with a reactive behavior has been developed in C++, and the same code has been translated in Python, to have a working example of simple agents for both languages (we named “SimplePlayer” the C++ player, and “HumblePlayer” the Python player).

The documentation of the package contains further programming information and tutorials.
A further description of the simulation mechanics in more details would be too bulky and complicated to be reported here because it would correspond to write in English language the operations programmed in the source code. Since the source code is almost self-explaining, the reader that wants to know the details of the simulation is recommended to directly read the source code, that is released as open source (see web reference [11]).

5.3 Reactive Agents and Emergence of Defender’s Dilemma

In addition to the example agents SimplePlayer and HumblePlayer, we wrote another reactive agent in Python, with just one characteristic that may be seen as not completely reactive since it uses memory: for a certain time frame after having passed the ball to a companion, the agent stays consistent with its choice of passing the ball and does not try to reach for it, at least for a fixed period (it uses memory to keep count of the elapsed time since the pass).

The agent is a simple reactive agent that decides which action to take based on a set of parametrized, fixed and rules that use hand-designed heuristics to rank the goodness of an action. The heuristics are based on things like the distance of the player to the ball, the distance to the opponent goal, the occlusion of the goal by opponents, the possibility that an opponent is close to the segment of the line towards the destination of our hypothetical kick, and so on).

We noticed that the agent with such characteristics, under some values of its parameters was reproducing quite often the Defender’s Dilemma (from the standpoint of the attacker) when set to play simplified games together with a companion against a lonely opponent goalkeeper (a 2 vs 1 game). It frequently ended up in situations in which the opponent goalkeeper was in a disadvantaged position, where if the goalkeeper tried to reach for the ball it would have had the effect of making the ball-owner agent pass it to the companion in a good position to score, and if the goalkeeper tried to stay in a defending position with respect to the free companion, the ball-owner would have tried to kick in a relatively open goal.

The five pictures below illustrate Defender's Dilemma that emerged with DilemmaPlayer, in a 2 vs 1 game inside our "Very Simplified 2D Robotic Soccer Simulator".

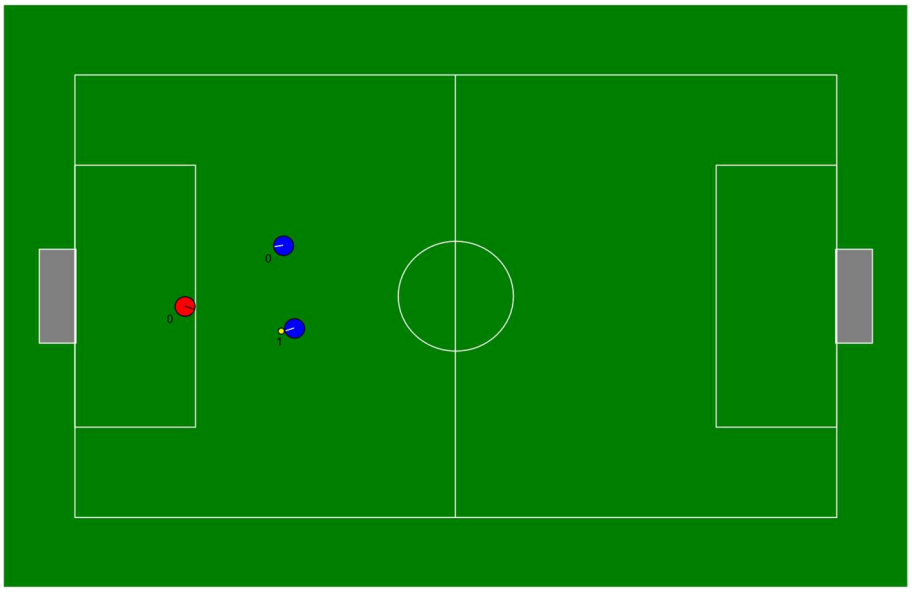


Figure 5.2 : A 2 vs 1 game with agents controlled by the "DilemmaPlayer" program. At tick t, the Defender's Dilemma emerges, with blue player 1 having the ball, but obstructed in perspective towards the goal by the red goalkeeper, while the other attacking player (blue 0) is less obstructed and slightly more advanced, so in a more favourable position to kick in case he receives the ball.

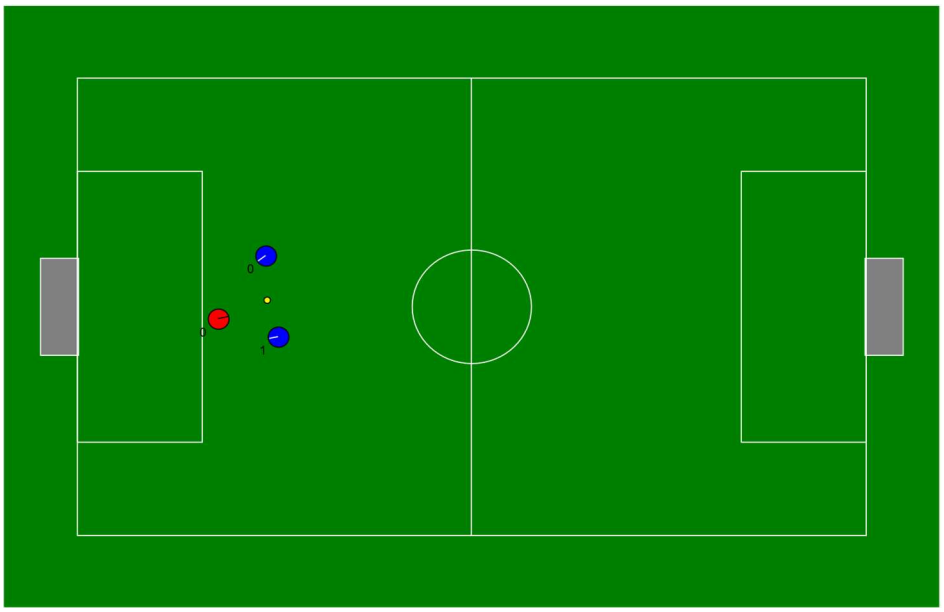


Figure 5.3: At tick t+5 the goalkeeper has advanced, but previous ball-owner (blue player 1) has already passed the ball to the companion (blue player 0), and the ball is halfway to reach him. The red goalkeeper starts to move towards the ball, but it may be too late.

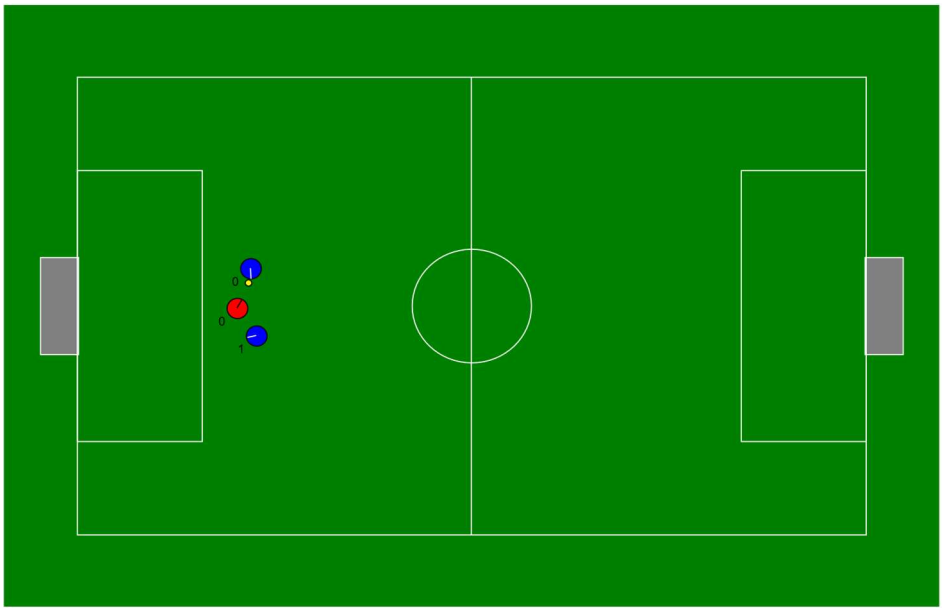


Figure 5.4: At tick t+9 the goalkeeper is directing towards the ball, but he seems late, as the blue player 0 already received it.

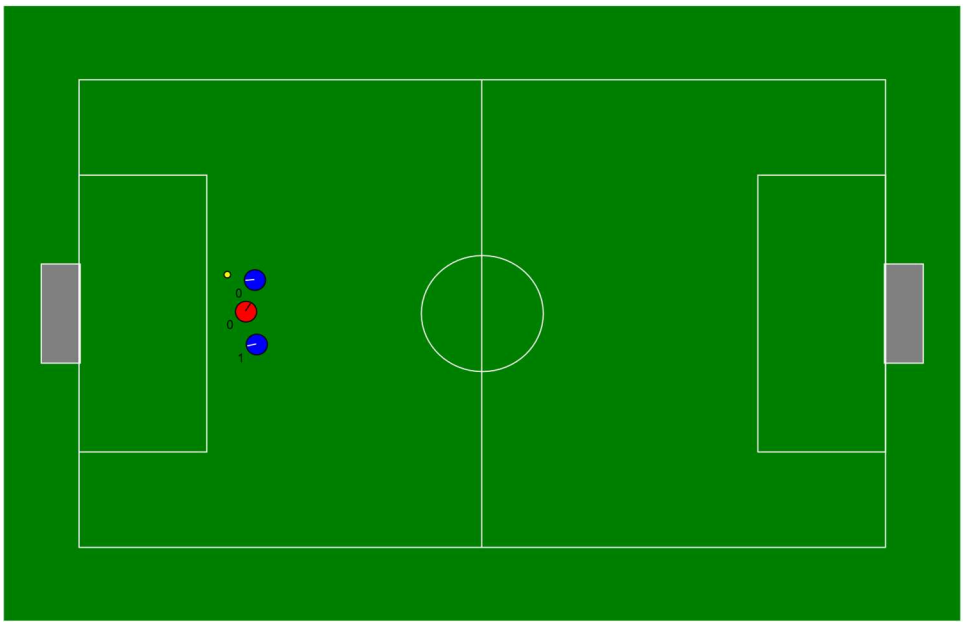


Figure 5.5: At tick t+10 the blue player 0 has kicked the ball towards the goal, since the potential trajectory it is not obstructed. The goalkeeper cannot do much.

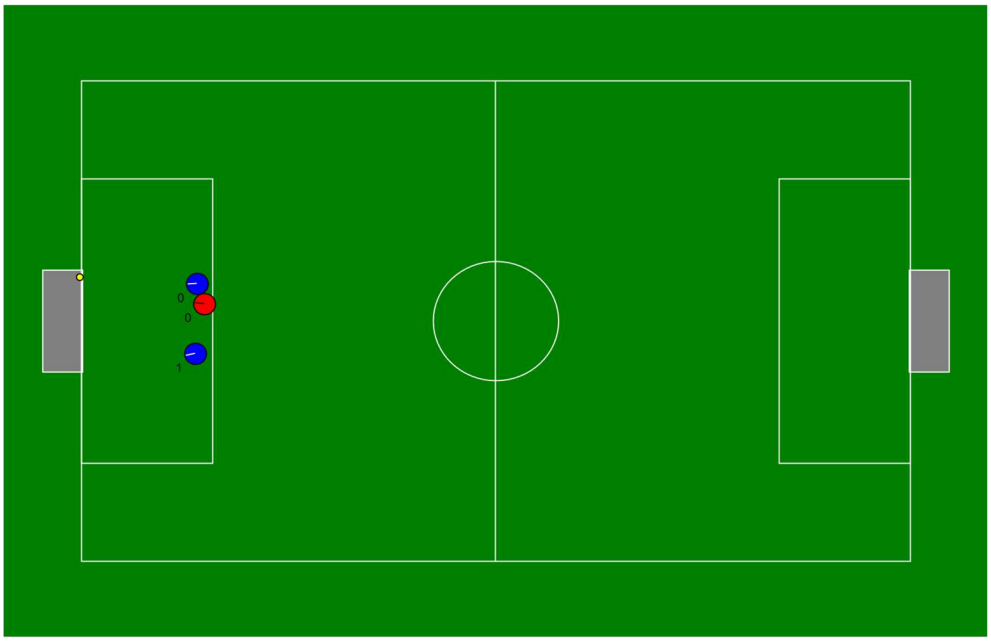


Figure 5.6: At tick t+22 , while all players are following the ball, the ball enters the goals, the blue team scored.

This means that we found one of the many ways to specify behavioral rules that make the Defender's Dilemma emerge, even with reactive-only agents and no planning (except for the "wait after passing" rule). The agent itself does not plan for any Defender's Dilemma creation, it has not a concept of tactics itself, but we see that a set of rules instilled in agents can give rise, through agents' interactions, to higher level structures that may identified as "strategies" that correspond to an emerging layer that may be described by a simplicial relationship according to Hypernetworks Theory.

The complete description of the algorithmic behavior of this agent would not be very suitable in English so we leave the interested reader to the source code that is available as open source. We named this agent "DilemmaPlayer".

5.4 Experiment Settings

An agent trained by Reinforcement Learning learns a policy that associates a state configuration with a distribution of probabilities over actions to be taken. That policy is actually an example of emerging behavior, because there is no prior information to follow about which action has to be taken: discovering the best action is the result of basic interactions combined with a learning mechanism that progresses autonomously. The policy, or learnt behavior, embodies in itself relevant knowledge about the environment, the agent, and the rewards. Hence, it is an example of implicit hypernetwork because it implicitly contains a discernment of relationships between actions to take, environment and agent.

In this paragraph we describe an experiment in which we made a robotic soccer agent learn a facilitated task using Deep Reinforcement Learning. From a conceptual standpoint this may be seen as making the agent build its hypernetwork of behaviors.

The motivation to use a facilitated task instead of the complete game with two opposing teams is due to the fact that in complex environments like this the Reinforcement Learning algorithms are known to learn very slowly, due to sample inefficiency: they need a great amount of samples to learn a task/skill (as explained also in [Irpan 2018]). Even if our simulator is very fast, Reinforcement Learning algorithms are computation intensive and require a lot of computing time and resources.

The facilitated task is set in our Very Simplified 2D Robotic Soccer Simulator (described in paragraph 5.2) and consists in two agents standing in front of the opponent goal, without goalkeeper, and without any prior knowledge. They must learn to score a goal using low-level actions, without using any human-coded skills.

The experiment uses the default settings of the simulator (that can be consulted in the simulator source code [11]), without changing pitch size, goal size, or any other parameters: the pitch length is 32 units, the pitch width is 19.2 units, the goal width is 4 units, the players have a radius of 0.4 units and the ball has a radius of 0.11 units. Two collaborating agents are created and placed in the neighborhood of opponent goal with a small amount of randomness in their position (one player is positioned at coordinates x=12.5 units, y=0 units, plus a 0.1*random value from a normal sampled in each coordinate, and the other at x=13.5 units, y=2 units plus another 0.1*random value from a normal sampled in both coordinates), with the ball close to one player plus a small random quantity (another 0.1*random value from normal

horizontally). Then players are left playing until either the ball goes outside the pitch, or a goal is scored, or 100 ticks passed. After that, they are placed again in a similar position to play again until a goal, a ball out or 100 ticks passed, continuously. This goes on for the 6000 ticks of a match, with pitch switching after 3000 ticks. The only time in which agents are not placed in front of the opponent goal is during half-time kick-off (that may be useful to give them just some different experience).

Agents receive a reinforce signal in the form of a reward. The reward is modeled in a suitable way for a complete game, even if this is a facilitated task. Whenever agents score a goal to the opponent, a reward equal to +100 is assigned. When there is a score in their own goal, the reward equals to -100. In case of a corner for the opponent, or a throw-in for the opponent, a reward of -10 is given.

There are also a small reward to advance in the pitch: the first time that the ball advances in the second quarter of the opponent pitch, a reward of +15 is given, the same happens the first time that the ball advances in the third quarter and in the fourth quarter of the opponent side. This is a way to signal that advancing towards the opponent goal is good (it is an example of "reward shaping"). Anyway in our facilitated task this ball advancing reward is not necessary because the starting position of the ball is already advanced, so these rewards are just added to the goal scoring reward. A reward system with some points in common has been used in the Google Research Football [Kurach et al. 2019].

5.5 Agents' Characteristics

As discussed in the chapter 6, while multi-agent reinforcement learning introduces an additional great level of complexity, good results may still be obtained also in multi-agent settings using just single-agent reinforcement learning, training each agent as if other agents where part of the environment. We followed that path and used standard, single-agent RL agents. Since the learning environment is rich and complex we could not use finite-states reinforcement learning methods but had to resort to methods with approximate functions, since the number of states is potentially infinite. Basing on current state of the art, one of the most promising methods is Proximal Policy Optimization with probability ratio clipping by [Schulman et al. 2017] (described extensively in the Reinforcement Learning paragraph 2.4), so we decided to adopt it, in its special version with the Generalized Advantage Estimation [Schulman et al. 2016] (also this described in paragraph 2.4), and with early stopping based

on Kullbach-Leibler divergence. We used a neural network for the policy function and another neural network for the value function. To implement that algorithm we started from the excellent examples of OpenAI in their “Spinning Up with Deep RL” tutorial (web reference [21]) , that has a version of Proximal Policy Optimization with probability ratio, and adapted it to our need and settings (for example we pooled the samples from multiple agents in the same buffer to be processed together).

The following is the pseudocode for the PPO algorithm that we used. In our experiment we collected 800 samples for each player for each training round (the number of trajectories may vary because it depends on how many samples compose a trajectory), and we used $M = 80$ and $N = 80$.

Algorithm 5.2 **Proximal Policy Optimization with ratio clipping**

Using same notation as in paragraph 2.4

Require: Policy network step size η

Require: Value network step size ω

Require: Initialize parameters θ of network π_θ with small random values

Require: Copy parameters θ_{old} of network $\pi_{\theta old}$ from θ

Require: Initialize parameters φ of network V_φ with small random values

For $k = 0, 1, 2, \ldots$ do:

collect trajectories $D_k = \{ \tau_i \}$ using policy $\pi_{old}(\theta_{old})$

compute rewards-to-go $G(\tau_t)$

compute advantage estimates $\widehat{A_t}$ using current estimate of value function $V_{\varphi k}$:

$$\widehat{A_t} = G(\tau_t) - V_{\varphi k}(s_t)$$

For $m = 0, 1, 2, \ldots, M$ do a policy gradient ascent iteration:

compute policy ratios:

$$d_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta old}(a_t|s_t)}$$

estimate policy gradient as:

$$\widehat{g_k} = \frac{1}{|D_k|} \sum_{\tau \epsilon D_k} \sum_{t=0}^{T-1} \nabla_\theta d_t(\theta) \hat{A}_t$$

update the policy network with gradient ascent (or other methods like Adam):

$$\theta \leftarrow \theta + \eta \widehat{g_k}$$

end of for

Copy optimized policy into fixed policy: $\theta_{old} \leftarrow \theta$

For $n = 0, 1, 2, \ldots, N$ do a value function gradient descent iteration:

estimate the value function gradients as:

$$\widehat{h_k} = \frac{1}{|D_k|T} \sum_{\tau \epsilon D_k} \sum_{t=0}^{T-1} \nabla_\varphi (V_\varphi(s_t) - G(\tau_t))^2$$

update the value function with a gradient descent step (or other method):

$$\varphi_{k+1} \leftarrow \varphi_k + \omega \widehat{h_k}$$

end of for

end of for

We had two possible architectural choices regarding the type of actions that our policy function was outputting: the first would have been to have a neuron outputting a continuous value indicating the direction angle of a dash or kick (a gaussian output), then another neuron outputting a continuous value indicating the power of dash or kick (another gaussian output), and three other neurons containing the logits of respectively the action of dash, the action of kick and the action of no-op (no operation, "do nothing"), that would have undergone a Softmax operation to obtain the probability distribution over them. Using continuous outputs for angles may potentially make the learning slower because the angles have a periodical behaviour and increasing them may bring the value in the opposite direction, and the neural network has to do additional work to learn this mechanism.

The second choice would have been to not use continuous output but instead simplify the action space creating 8 discrete kick actions (up, down, left, right, diagonal up-left, diagonal up-right, diagonal down-left, diagonal down-right) with fixed-level power, 8 discrete dash actions (same 8 directions as the kick actions) with fixed-level dash power, and a no-op action, for a total of 17 discrete actions. We opted for this second solution and used 17 final neurons containing the logits of the 17 discrete actions, to be operated by a final softmax to obtain a distribution of probabilities over the actions to take. This is somehow limiting the actions of the players and the final performance, but it may speed up the learning. To further speed up the learning process we used small neural networks: they need less training samples than big neural networks, but the price to pay is having a less powerful neural network, that implies a less efficient behavior. So we used a policy network with 20 input neurons, a first hidden layer of 40 neurons with hyperbolic tangent activation function, a second hidden layer of 20 neurons with hyperbolic tangent activation function, and an output layer with 17 neurons, representing the logits for the 17 actions, on top of which the sampling process did a SoftMax to convert them to probabilities.

The value network was like the policy network with the exception of the output that was just one neuron: 20 input neurons with hyperbolic tangent activation function, a first hidden layer of 40 neurons with hyperbolic tangent activation function, a second hidden layer of 20 neurons with hyperbolic tangent activation function, and an output layer with only one neuron representing the value of the state.

The 20 input features were the same for the two networks: x and y coordinates of the ball, x and y vectorial components of ball velocity, x and y coordinates of the two player, x and y vectorial components of the two players' velocities, the direction of each player, a flag for each player to indicate if he already acted in the current tick, the angle between current agent and

the ball, the angle between current agent and the center of the goal, a flag indicating if the ball is at kicking distance for the current player, and a flag indicating if the current player is the closest to the ball.

The optimization mechanism was not plain gradient descent but was Adam (a dynamically adapting gradient) [Kingma and Ba 2014].

We created two agents that were acting independently, without any coordination by design, so that each one was acting and training considering the other as part of the environment, but we used the experiences of both agents into the learning mechanics: we collected the states, the actions, the value estimates. and the rewards of each players and used them all together to train the same shared policy function and value function. In this way we sped up learning.
The learning loop consisted in collecting 800 samples from each actor, each sample containing the state information, the value function estimate, the policy function output for that state, the taken action, the reward. Then, when the total 1600 samples were all collected in a batch, the training iterations took place, computing the advantage function for each sample, and doing the policy network learning iteration for 80 iterations (or less, in case of a over-the-threshold Kullbach-Leibler divergence between the estimates of the old and new probability distributions) and the value network learning for 80 iterations (in the Proximal Policy Optimization algorithm described in algorithm 6.3 the variables M and N would be equal to 80). After this learning round, other 800 samples from each agent were collected and the process repeated in the same way with the learning iterations. After a match ended, a new one was started to continue to collect samples.
The value of ϵ for the clip of the probability ratio was 0.2 , the threshold for Kullbach_Leibler divergence early stop was 0.015, the parameters for the Generalized Advantage Function were γ=0.99 and λ=0.97 . Both the policy network and the value network were optimized with the Adam gradient descent/ascent algorithm, using the learning rate equal to 0.0003 for the policy network and 0.001 for the value network.

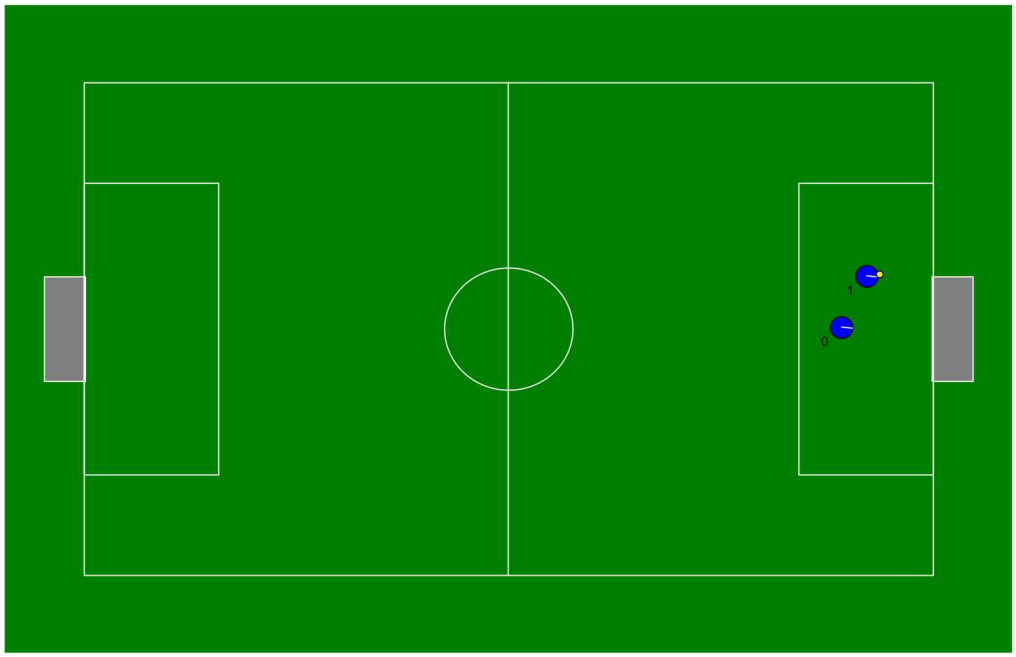


Figure 5.7: An example of initial setting of the experiment.

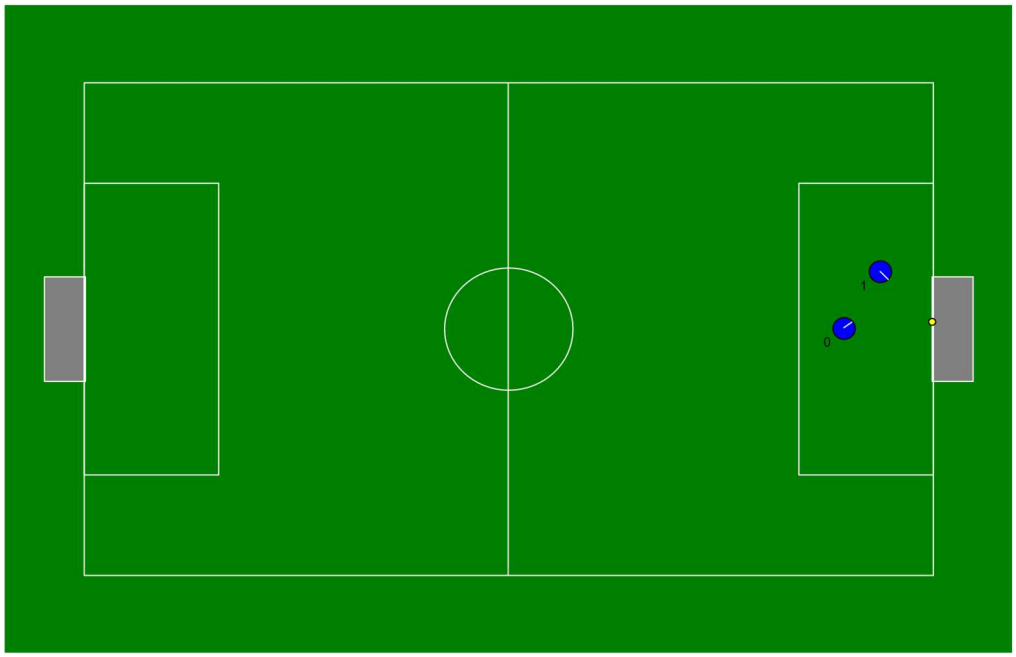


Figure 5.8: An example of scored goal.

5.6 Results and Discussion

We first ran approximately 70.000 learning “rounds”, with each round, as explained above, consisting of the usage of 1600 samples in a batch to do 80 value network Adam gradient descent iterations and a maximum of 80 policy network Adam gradient ascent iterations.

Despite dedicating a lot of computing time on a current standard notebook, the amount of samples collected and total computing is still very small with respect to the big quantity typically needed for Reinforcement Learning in complex environments (big experiments in complex environments like the one made by OpenAI to apply Reinforcement Learning to the Dota2 game, are run on the computing power equivalent to 180 years of human play per day, every day for many months, see web reference [22]). So we suppose that the learning process was far to be complete, but time constraints and lack of computing resources made not possible to have a big amount of training.

We then compared the performance of the trained agents with the performance of agents that choose randomly the action among the 17 available, to measure if there was any difference. Obviously the random agents were placed in the same environment with the same settings as the RL agents.

We ran 1000 games with the random agents and 1000 games with the reinforcement learning agents. For each type of agent we computed the goal difference, that is the goal scored minus the goal that have been scored in their own goal. The goals scored in own goal were very few for both types, because given the setting it was very difficult to accidentally kick the ball in own goal: when it rarely happened it was during half-time kick-off, a position from which the ball may end up in own goal if kicked. So usually for each game there have been 0 goals in own goals, or at maximum 1, both in random agents and in RL agents.

Anyway, the goal difference is a measure of the performance, the bigger the better.
The random agents made an average of 23.583 goals for game, with a variance of 28.331 .
The reinforcement learning agents made an average of 29.621 goals for game, with a variance of 37.547.

It seems that the reinforcement learning agents performed much better. To check it, we need to do a statistical test.

We begin with the Levene test to assess homogeneity of variances [Levene 1960] between the goal differences of RL agents and those of random agents, with level of significance = 0.01.

We obtain a P-Value= 0.000052, so we empirically refute homogeneity of variances.
We can proceed doing a t-test to compare the two distributions, and since the variances are not homogeneous we use the Welch's t-test, with level of significance 0.01 .

The one-sided Welch's test gives a very small P-Value: $3.408*10^{-108}$ that means that we can refute the hypothesis of the two samples belonging to the same distribution. That means that from a statistical standpoint the better performance of reinforcement learning agents is not due to a random chance, but it is due to an actual difference in ability.

Nonetheless, the Welch test is not robust when the samples are not normally distributed, even if with a larger number of samples the robustness increases. To have a safer test, we first test if the samples are normally distributed, because otherwise we may decide to also use another different type of test.

We use the Shapiro-Wilk test for normality of samples [Shapiro and Wilk 1965] and set the level of significance to 0.01.
The Shapiro-Wilk test applied to the goal differences of RL agents gives a P-Value = 0.000005 that means that the normality hypothesis is refuted.

The Shapiro-Wilk test for normality of random agents' goal difference samples gives P-Value = 0.000337 so even in this case the normality hypothesis refuted.

Given the assumption of non-normality of sample and non-homogeneous variances we use Mann-Whitney U test [Hettmansperger and McKean 2011 Ch.2] to assess the null hypothesis that it is equally likely that a randomly selected value from one population will be less than or greater than a randomly selected value from a second population. We use a level of significance =0.01 .

Mann-Whitney U test between RL goal differences and random goal differences gives a P-Value= $1.799*10^{-98}$, so with significance 0.01 we empirically refute the null hypothesis and this implies that the two samples have a different distribution.

This confirms that the better performance of reinforcement learning agents is to impute to their better ability.

This gives strong empirical evidence that the Proximal Policy Optimization algorithm has worked well and agents learned in the process.

After this promising result, we decided to continue the training process to check if it was possible to improve even more the learning. So, we continued to train the same agents for

other 262.000 training rounds. The results were successful. The average goal difference by the reinforcement learning agents was now 36.117, with a variance of 48.881. This second training process increased the average performance of 6.5 goals for match.

We compute the same statistical tests as before, with these new data, always with significance 0.01.

The Levene test for homogeneity of variances between random and rl results gives P-Value= $7.09*10^{-15}$. Hence we refute the homogeneity of variances hypothesis.

One sided Welch's t-test with non-homogeneous variances gives P-Value=$3.585*10^{-301}$.

That indicates very strong evidence that the difference of the two samples is not due to random chances. But since Welch's t-test is not robust in non-normal populations, we try also other tests.

We already know that random results are non-normal so we use the Mann-Whitney U test, and obtain a P-Value of $1.465*10^{-234}$ that is a very strong evidence against the hypothesis that the difference is due to random chance.

From this we accept the hypothesis that the random players and the reinforcement learning players have a difference in average performance that is due to the better ability of reinforcement learning players

The statistical tests ran on the results give a strong evidence that the Proximal Policy Optimization made the agents learn and improve their skills. The performance in goal difference is big, being on average more than 12.5 goals for game than the one achieved by random behaviour, an improvement of more than 53% . While this itself is a success, from an expert player we can expect even a much bigger difference. We conclude that this is due to an incomplete level of training. The main issue in reinforcement learning is the great amount of samples/experience needed, technically RL it is "sample inefficient", and in our experiment we trained the agents only for little time. Nonetheless the results are encouraging.

We showed that even in robotic soccer the Reinforcement Learning can work using directly basic actions and not only with high-level pre-made skills built by programmers combining basic actions.

To the best of our knowledge this is a unique and novel result in the application of PPO Reinforcement Learning to robotic soccer.

As reported in the Reinforcement Learning chapter, many experiments have been realized in the application of RL to robotic soccer. A certain part of those studies has been dedicated to train humanoid robots to execute basic movements, a purpose that is very different from ours. Another part of RL robotic soccer research has been devoted into learning to choose among the best hand-made high-level sequence of actions. Also that task is a different from ours, because we aim at train the agents without inserting prior knowledge into them, so they do not have available any pre-made sequence of action that are already able to solve some smaller tasks. Other studies aimed at training agents in “Keepaway Soccer” that is a game where the scope is to keep possess of the ball (or to gain it if you are not the ball owner) or similar tasks, and that is a different task from ours (and moreover that was achieved using human-made high-level skills).

Other studies were, like ours, using basic actions without human-made action sequences, but were much simpler in the aim, like learning to kick in one direction, or going to one direction.
In brief, previous application of RL to robotic soccer either concentrated on completely different tasks than ours, or the task was similar but agents were equipped with pre-made high-level actions (a thing that makes the problem different and easier to solve), or the task was somehow vaguely alike but easier and more limited, or the RL algorithm was different.

Hence this experiment is novel and expands the current landscape of application of reinforcement learning to robotic soccer with a successful result (an extensive literature review about previous RL applications to robotic soccer was discussed in more detail in paragraph 2.4.4).

From a Hypernetworks theory standpoint, the policy network learnt by the Reinforcement Learning algorithm is able to recognize the relevant hypersimplices about the game configuration from the input data, and is able to relate them with the proper action to take, building in this way a higher-level hypersimplex.

5.7 Summary

The third research question asked if “Is it possible to make behavioural patterns emerge in the game without specifying the behavioural rules in detail ?”. We decided to use Reinforcement

Learning to answer, making a robotic player learn the behaviour through trials and errors. Since we required a fast and simple simulator, we decided to write our own to better suit our needs. In that environment, we were able to train with Reinforcement Learning (Proximal Policy Optimization) two agents to score a goal in a simplified setting, answering to the third research question. From a hypernetworks standpoint that is equivalent to have the agent learn a hypernetwork containing the relevant relationships between the game configuration and the actions to take.

Chapter 6. Conclusions and Further Works

6.1 Summary of the Research

6.1.1 Summary

In this research we devised two novel methods to analyse emergent tactics in robotic soccer games.

The first method is an algorithm, fast in execution and parametrizable, that recognizes the rise of “Defender’s Dilemmas”. The importance of this structure was discussed in Section 3.1 and the algorithm is given in Section 3.2. The statistical analysis in Sections 3.3, and 3.4 shows that forming defender’s dilemma structure is associated with positive outcome as discussed in Sections 3.5 and 3.6.

Chapter 4 presents the second method. This method applies feed-forward neural networks to forecast the success of certain game configurations with a high degree of accuracy, *i.e.* it forecasts if a sequence of actions will result in a scored goal.

This second method is similar in conceptual meaning, but different in construction, training, and purpose, to the Value Function of Reinforcement Learning, from which it is inspired.

Chapter 5 presented our own fast robotic soccer simulator, namely “Very Simplified 2D Robotic Soccer Simulator”, which speeds up analysis and machine learning processes.

This simulator allowed us to write an agent based on simple rules and hand-made heuristics that created the emergence of the Defender’s Dilemma, in a beautiful example of how an unplanned high-level structure may arise from lower-level interactions (a thing that theoretically may be described with hypernetworks). The simulator is released as free/open source software, so it is available to other researchers to use.

The high speed of our simulator allowed us to train a reinforcement learning agent. Because of the sample inefficiencies of reinforcement learning algorithms, of the complexity of the environment, of the high computing requirements of reinforcement learning algorithms and of our limited availability of computing power, we did not train the agent to play a complete team

vs team game, that would have required much more time and computing power. We instead trained the agent in a limited task, consisting in placing two RL agents in the neighbourhood of the opponent goal, and train them to score goals, using basic actions and not hand-made programmed skills. The RL algorithm used was Proximal Policy Optimization, with probability ratio clipping, Kullbach-Leibler divergence early stopping, and Generalized Advantage Estimation. The experiment was successful and the agents learned to score, as confirmed by a statistical test.

6.1.2 Answers to the Research Questions

The research in this thesis supports the following answers to our research questions.

Research Question 1

> Is it possible to identify patterns of play, that lead a team to obtain an advantage ?

Our experiments show that it is. In Chapter 3 we identified the Defender's Dilemma as one pattern that is statistically associated to victory, and created an algorithm to recognize it.

Research Question 2

> Is it possible to forecast with a good level of accuracy if a certain game action or sequence of game actions is going to be successful, before it has been completed ?

Our experiments demonstrate that it is. In Chapter 4 we detailed how a feed forward neural network may be used to forecast with great accuracy if a certain game configuration will end up either with a goal for the first team, or a goal for the second team, or without goals.

Research Question 3

> Is it possible to make behavioural patterns emerge in the game without specifying the behavioural rules in detail ?

It is, and we answered in Chapter 5: through Deep Reinforcement Learning we trained soccer agents that had not any prior knowledge, any built-in rule, or any high-level pre-made skill,

and they learned to score goals, that means that a behavioural pattern emerged thanks to their training.

6.2 Thesis Contributions

The research contributes in various ways to the advance of the field.
The first method of analysis of a robot soccer game, the Defender's Dilemma algorithm (Ch. 3), offers a fast and efficient way to detect configurations that may lead to tactical advantages. It may be used in real-time, and it is an example of how an Hypernetwork-based description of a dynamic structure may be translated to a program that detects it through well-defined conditions.

The second method of analysis of robot soccer, the usage of neural networks to forecast if a game configuration will end in a score or not (Ch. 4), is novel. We listed some distinct methods from other authors with common aspects, and we also pointed out the differences. To our knowledge in literature there is no equivalent method. This is an example of how a neural network may capture and identify hidden structures that are equivalent to an implicit Hypernetwork.

The application of Reinforcement Learning to make two players score collaboratively (Ch. 5) is novel, since previous applications of RL to robotic soccer either were dealing with other purposes like coordinating the robot movements, or trained for different task like kicking to a certain target, or used hand-made high-level actions, or used a different algorithm and setting. Hence this RL application to robot soccer is an additional contribution to the field brought by the research.

Our "Very Simplified 2D Robotic Soccer Simulator" (Ch.5), released as open source [web reference 11], is a very fast simulator, well suited for Reinforcement Learning. It is free both as "freedom" and as "free beer", available to the scientific community to be used and modified, and so it is another contribution derived from the research.

Moreover, as a further, practical, contribution, during the research we noticed and corrected two bugs in the original RoboCup 2D server, related to the penalty phase, and our corrections have been accepted and inserted in the official source code on GitHub, to be used in the world cup tournament.

6.3 Further Work

The findings and methodologies of this research can be further expanded and be the starting point of a promising path of study.

The Defender's Dilemma Algorithm

The Defender's Dilemma algorithm may be used to obtain a dataset of game configurations leading to defender's dilemmas, and then a neural network may be trained to identify the situations from which the dilemma is likely to arise in next ticks: that may be used by defending agents to activate strategies to avoid the formation of the dilemma. Another research direction in defender's dilemma topic is to find out what are the higher-level structures that are formed when the dilemma appears: what are the players' configurations or the tactics activated that are based on the defender's dilemma.

Forecasting Game Configurations

The second path of research may be about the neural network that forecasts if a game configuration will end up in a goal for a team or not. This method may be improved, for instance using convolutional neural networks, in order to overcome the "fixed order input limitation", that, as we explained before, is the need of having a certain order in the players data that are fed as input to the neural network. With convolutional neural networks, a picture of the game state would be used as input, and the neural network will possibly learn to evaluate configurations without giving relevance to the players' shirt numbers. That will require a bigger amount of computing power, but could likely result in a system that generalizes better.

Proximal Policy Optimization

Also the experiment of using PPO Reinforcement Learning to train soccer agents can be extended to more demanding tasks, such as trying to score against a goalkeeper, then adding opponent defenders, up to experiment with complete games. This is really dependent on the computing power: the more available, the more complex the task may be.

The Very Simplified 2D Robotic Soccer Simulator

Our Very Simplified 2D Robotic Soccer Simulator may be improved: the first additional functionality that could be worth of being implemented is the creation of a software layer that allows running agents directly in separate machines, using either MPI or gRPC (at the current state of development this is a thing that is possible but that the user has to write by himself). This simulator has been kept as simple and as fast as possible, so any kind of extensions and additions to it are possible without much architectural difficulties.

Hypernetwork Theory and Applications

This thesis is one of the first formal application of neural networks to compute the relational structure of hypersimplices. It opens up new research directions not present in the hypernetworks literature. In hypernetworks the relation R maps an ordered set of unstructured vertices (a simplex) to a hypersimplex in which the multidimensional structure is explicit. The application of neural networks suggests a version of R that maps not just one simplex to a hypersimplex, but many simplices in a data space to that hypersimplex. There are potentially great practical and theoretical benefits in learning part-whole assembly relations from data rather than programming. These observations could stimulate new research in hypernetwork theory. Related to this is the research question posed at the end of paragraph 2.2: “if I am using a deep learning model, am I using a representation of an equivalent hypernetwork model” ?

APPENDIX A. RoboCup 2D Resources and Data

The RoboCup 2D soccer simulator server [web reference 5] is open source and can be downloaded, installed and run to launch simulations. It needs two opponent (software) teams to connect and play, each team being composed of independent soccer players (usually 11, each of which is an artificial intelligent software agent) and possibly a coach (an artificial intelligent software agent too). The binary executables of the teams that participated to past RoboCups are available for download in the RoboCup archive website [web reference 6], and some team also has its own download webpage such as the 2015 champion Wrighteagle [web reference 7].

The soccer simulator while running saves the information about games on log files on disk, making possible to subsequently analyse the games or to watch them again with the log player. Log files of past international matches, such as the yearly official RoboCup tournament, are available on a dedicated website [web reference 2] for download: there are thousands of recorded games. An explanation of the rules and inner mechanisms of RoboCup 2D can be found in the official RoboCup 2D server manual [5], further information has to be obtained from the source code of the server implementation and the log player (that can be used as an implicit reference to understand the format of the log files). Other useful sources of knowledge for game mechanics and data format may be the descriptions of winning agent architectures available such in [Stone et al. 1999], [Reis and Lau 2001], [Bai et al. 2012] and in [Akiyama & Nakashima 2013].

Log files are text files with information written in a structured way that is also readable by humans. There are two types of them: one whose names end with the ".rcl" suffix and contains the actions sent by players and received by the server, and one whose names end with the ".rcg" suffix and contains data that a log player could use to visualize the match, such as position and velocity of players and ball, game state, in addition to the game general settings. To understand the position of players and dynamics of a game, usually only the ".rcg" file is necessary, while the ".rcl" may be useful to help debugging the intelligent agents or the simulator because it contains the timed commands sent by players to the server. The ".rcg" files contain the information about players' positions, movement, stamina and so on, codified through textual tokens as "show", "msg", "playmode" separated by round brackets, and each token may contain numerical or textual parameters, surrounded by round brackets as well.

For instance the following line extracted from a match log:

(show 5973 ((b) -23.228 5.0538 -0.6786 1.7303) ((l 1) 0 0x9 -50.1586 2.4081 0.008 0.0978 86.794 -89 (v h 180) (s 8000 1 1 101799) (f l 11) (c 0 553 6616 0 8 7177 175 637 0 0 4603)))

means that during the time frame 5973 the ball is moving in the position at coordinates x=-23.228, y=5.0538 with velocity vx=-0.6786, vy=1.7303. Moreover the player number 1 of the first team is of the default player's type, is a goalkeeper, is moving into the position at coordinates x=-50.1586, y=2.4081 with velocity vx=0.008, vy=0.0978, with body orientation angle = 86.794 and neck orientation angle = -89. There are also other player's in-game variables and statistics, for example it reports that the player has high view quality, has view width=180, stamina=8000, effort=1, recovery=1, stamina capacity=101799 and so on.
Some information is coded only indirectly: for instance to know which player is actually controlling the ball it is necessary to check if the distance between the ball and every player is less than a certain radius of action, and if a unique player happens to be so close then it can be supposed to have ball possession.

Other indirect information is about passing the ball: the ".rcg" log doesn't explicitly record a "pass" action, but for every tick there is the information about the number of kicks executed by every player: checking for that number to increment gives a hint that a player has just kicked (or the information about kicks may be obtained from the other log file ".rcl" that records commands), and since players could potentially "kick" even without possessing the ball, that hint can be intersected with the property of the player of being close enough to the ball (or having ball possession, as written above) in the previous time tick, to discover that a player has actually kicked the ball, to pass it (if later a teammate received it) or to score. Things that of course are not encoded in the log files are the high-level concepts such as the action of "dribbling": there is not a "dribble" command for players, dribbling is the high-level result of lower-level actions such as "dash" and "kick". This means that a second stage parser or analyser may be written to reconstruct these "high-level" events.

In the log player package there is also the program "rcg2xml" that allows to translate log files into xml files for an easier parsing.

References

[Abreu et al. 2010] P. Abreu, J. Moura, D. Castro Silva, L. P. Paulo Reis and J. Garganta, "Football scientia- an automated tool for professional soccer coaches," 2010 IEEE Conference on Cybernetics and Intelligent Systems, 2010, pp. 126-131, doi: 10.1109/ICCIS.2010.5518568

[Abreu 2019] Miguel Abreu, Luis Paulo Reis, Nuno Lau, “Learning to Run Faster in a Humanoid Robot Soccer Environment Through Reinforcement Learning”, *LNAI 11531 RoboCup 2019: Robot World Cup XXIII* (2019). ISBN 978-3-030-35698-9

[Adachi et al. 2017] Adachi, Y., Ito, M., & Naruse, T. , “Classifying the strategies of an opponent team based on a sequence of actions in the RoboCup SSL.” , LNAI 9776 RoboCup 2016 Robot World Cup XX (2017). Springer, ISBN 978-3-319-68791-9

[Ahumada et al. 2013] Ahumada GA, Nettle CJ, Solis MA. , “Accelerating Q-Learning through Kalman Filter Estimations Applied in a RoboCup SSL Simulation.”, In: Robotics Symposium and Competition (LARS/LARC), 2013 Latin American IEEE; 2013. p. 112–117.

[Akiyama 2018] Hidehisa Akiyama, Tomoharu Nakashima, Takuya Fukushima, Jiarun Zhong, Yudai Suzuki, An Ohori , “HELIOS2018: RoboCup 2018 Soccer Simulation 2D League Champion”, In RoboCup 2018: Robot World Cup XXII (pp.450-461) August 2019 DOI:10.1007/978-3-030-27544-0_37

[Akiyama & Nakashima 2013] Akiyama H. and Nakashima T. , “HELIOS2012: RoboCup 2012 Soccer Simulation 2D League Champion”, in RoboCup 2012: Robot Soccer World Cup XVI ,Volume 7500 of the series Lecture Notes in Computer Science pp 13-19, 2013

[Almeida et al. 2009] Almeida R., Reis L. P., Alípio M. J. , “Analysis and forecast of team formation in the simulated robotic soccer domain”, Portuguese Conference on Artificial Intelligence. Springer Berlin Heidelberg, 2009.

[Almeida et. al 2013] Almeida, F., Abreu, P.H., Lau, N., Reis L. P. , “An automatic approach to extract goal plans from soccer simulated matches”, Soft Computing (2013) 17: 835. doi:10.1007/s00500-012-0952-z

[Arora and Doshi 2021] Arora, S., & Doshi, P. , A survey of inverse reinforcement learning: Challenges, methods and progress.”, Artificial Intelligence, 297, 103500. , doi:10.1016/j.artint.2021.103500

[Asik et al.2019] Okan Asık,, Binnur Gorer, H. Levent Akın, “End-to-End Deep Imitation Learning: Robot Soccer Case Study”, RoboCup 2018: Robot World Cup XXII (Lecture Notes in Computer Science Book 11374), 2019

[Atkin 1972a] Atkin R.H. , “From cohomology in physics to q-connectivity in social science”, International Journal of Man-Machine Studies, Volume 4, Issue 2, April 1972, Pages 139-167

[Atkin 1972b] Atkin R.H. , “Multi-Dimensional Structure in the Game of Chess”, International Journal of Man-Machine Studies, Vol. 4

[Aurenhammer & Klein 2000] Aurenhammer F. and Klein R. , “Voronoi diagrams”,

In J.-R. Sack and J. Urrutia, ed., Handbook of Computational Geometry, p 201–290. North-Holland, 2000.

[Ausiello et al. 1986] G. Ausiello, A. D'Atri AND D. Sacca, "Minimal representation of directed hypergraphs", SIAM Journal on Computing, 1986

[Bahdanau et al. 2014] Bahdanau, D., Cho, K., and Bengio, Y. , "Neural machine translation by jointly learning to align and translate", 2014, arXiv:1409.0473.

[Bai et al. 2012] Bai A., Chen X., MacAlpine P., Urieli D., Barrett S., Stone P. , "WrightEagle and UT Austin Villa: RoboCup 2011 Simulation League Champions" , RoboCup 2011: Robot Soccer World Cup XV, Volume 7416 of the series Lecture Notes in Computer Science pp 1-12, 2012

[Bai et al. 2015] Bai, A., Wu, F., Chen, X. , "Online planning for large Markov decision processes with hierarchical decomposition.", ACM Trans. Intell. Syst. Technol. 6(4), 45:1–45:28 (2015)

[Bai et al. 2018] Bai, A., Russell, S., & Chen, X. ,"Concurrent Hierarchical Reinforcement Learning for RoboCup Keepaway", in RoboCup 2017: Robot World Cup XXI
ISBN 978-3-030-00307-4 Springer (2018)
https://doi.org/10.1007/978-3-030-00308-1

[Balestriero et al. 2021] Randall Balestriero, Jerome Pesenti, Yann LeCun, "Learning in High Dimension Always Amounts to Extrapolation", arXiv:2110.09485

[Beetz et al. 2005] Beetz M., Kirchlechner B., Lames M. , "Computerized real-time analysis of football games", IEEE Pervasive Computing (Volume: 4, Issue: 3, July-Sept. 2005)

[Bengio et al. 2003] Y. Bengio, R. Ducharme, P. Vincent. , "A neural probabilistic language model.", Journal of Machine Learning Research, 3:1137-1155, 2003.

[Bengio et al. 2007] Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. , "Greedy layer-wise training of deep networks.", In Advances in Neural Information Processing Systems. 2007.

[Bengio et al. 2021] Yoshua Bengio, Yann Lecun, Geoffrey Hinton, "Turing Lecture: Deep Learning for AI", Communications of the ACM, July 2021, Vol. 64 No. 7, Pages 58-65
10.1145/3448250

[Benjamini and Hochberg 1995] Yoav Benjamini and Yosef Hochberg, "Controlling the False Discovery Rate: A Practical and Powerful Approach to Multiple Testing", Journal of the Royal Statistical Society. Series B (Methodological). Vol. 57, No. 1 (1995), pp. 289-300, Wiley

[Berge 1973] Claude Berge, "Graphs and Hypergraphs", North-Holland 1973, ISBN volume: 0 7204 2453 4ISBN volume: 0 7204 2453 4

[Bishop 2006] Christopher M. Bishop, "Pattern Recognition And Machine Learning", ISBN 9780387310732, Springer Nature 2006

[Bonabeau 1998] Eric Bonabeau, "Social Insect Colonies as Complex Adaptive Systems", Ecosystems, September 1998, Volume 1, Issue 5, pp 437–443

[Busoniu et al. 2008] L. Busoniu, R. Babuška, and B. De Schutter, “A Comprehensive Survey of Multiagent Reinforcement Learning” IEEE Transactions on Systems, Man, and Cybernetics—Part C: Applications and Reviews, vol. 38, no. 2, March 2008, pages 156–172. © 2008 IEEE.

[Bromley et al 1994] Bromley, J., Guyon, I., LeCun, Y., Säkinger, E., and Shah, R., “Signature verification using a Siamese time delay neural network.”, Advances in Neural Information Processing Systems, 1994, 737–744.

[Browne et al. 2003] Browne, A., Hudson, B., Whitley, D., Ford, M., Picton, P., & Kazemian, H., “Knowledge extraction from neural networks.”, IECON’03. 29th Annual Conference of the IEEE Industrial Electronics Society (IEEE Cat. No.03CH37468). doi:10.1109/iecon.2003.1280352

[Celiberto et al. 2007] Celiberto LA, Ribeiro CH, Costa AH, Bianchi RA. , “Heuristic reinforcement learning applied to robocup simulation agents.”, In: Robot Soccer World Cup Springer; 2007. p. 220–227.

[Chaitin 2004] Gregory J. Chaitin, “META MATH! The Quest for Omega”, IBM Research, arXiv:math/0404335v7 [math.HO]

[Cintia et al. 2015] Cintia P., Pappalardo L., Pedreschi D., Giannotti F., Malvaldi M. , “The harsh rule of the goals: data-driven performance indicators for football teams.”, In: IEEE international conference on paper presented at the data science and advanced analytics (DSAA), 2015. 36678 2015.

[Copete et al. 2015] Copete J.L., Suzuki J., Wei Q., Iwaki R., Endo N., Mori H. Nagai Y., Asada M., “Estimation of Players's Actions in Soccer Matches Based on Deep Autoencoder”, JSAI Technical Report, SIG-Challegne-042-02(5/3) http://www.osaka-kyoiku.ac.jp/~challeng/SIG-Challenge-042/SigChallenge-042-02.pdf

[Dashti et al. 2006] Dashti H.T., Aghaeepour N., Asadi S., Bastani M., Delafkar Z., Disfani F. M., Ghaderi S. M., Kamali S., Pashami S., Siahpirani A. F., “Dynamic Positioning Based on Voronoi Cells (DPVC)”, RoboCup 2005: Robot Soccer World Cup IX, Volume 4020 of the series Lecture Notes in Computer Science pp 219-229, 2006

[de Medeiros et al. 2020] Thiago Filipe de Medeiros; Marcos R. O. de A. Máximo; Takashi Yoneyama, “Deep Reinforcement Learning Applied to IEEE Very Small Size Soccer Strategy”, 2020 Latin American Robotics Symposium (LARS), 2020 Brazilian Symposium on Robotics (SBR) and 2020 Workshop on Robotics in Education (WRE), 2020, pp. 1-6, doi: 10.1109/LARS/SBR/WRE51543.2020.9306954.

[De Nooy et al. 2018] Wouter De Nooy, Andrej Mrvar, Vladimir Batagelj, “Exploratory Social Network Analysis with Pajek: Revised and Expanded Edition”, Cambridge University Press 2018 ISBN:9781108565691, DOI:https://doi.org/10.1017/9781108565691

[Devlin et al. 2019] Devlin, J., Chang, M., Lee, K., and Toutanova, K. , “Bert: Pre-training of deep bidirectional transformers for language understanding.”, In Proceedings of ACL'2019; arXiv:1810.04805.

[Duchi et al. 2011] John Duchi, Elad Hazan, Yoram Singer, “Adaptive Subgradient Methods for Online Learning and Stochastic Optimization”, Journal of Machine Learning Research 12(61):2121−2159, 2011.

[Dutt-Mazumder et al. 2011] Dutt-Mazumder A., Button C., Robins A., Bartlett R.

“Neural network modelling and dynamical system theory: are they relevant to study the governing dynamics of association football players?”, Sports Med 41(12):1003–1017. doi:10.2165/11593950-000000000-00000

[Enokida et al. 2001] Shuichi Enokida, Takeshi Ohasi, Takaichi Yoshida and Toshiaki Ejima, “Extended Q-learning : Reinforcement Learning using Self-Organized State Space” Robot Soccer World Cup IV / RoboCup 2000 (2001)

[Erdogan and Veloso 2011] Erdogan, C., Veloso, M., “Action selection via learning behavior patterns in multi/robot domains.”, In: Proceedings of International Joint Conference on Artificial Intelligence 2011, pp. 192–197 (2011)

[Fahami et al. 2017] Mohammad Amin Fahami, Mohamad Roshanzamir and Navid Hoseini Izadi , “A Reinforcement Learning Approach to Score Goals in RoboCup 3D Soccer Simulation for Nao Humanoid Robot”, 7th International Conference on Computer and Knowledge Engineering (ICCKE 2017), October 26-27 2017, Ferdowsi University of Mashhad,

[Fathzadeh et al. 2006] Ramin Fathzadeh ; Vahid Mokhtari ; Morteza Mousakhani ; Fariborz Mahmoudi, “Mining Opponent Behavior: A Champion of RoboCup Coach Competition”, 2006 IEEE 3rd Latin American Robotics Symposium
DOI: 10.1109/LARS.2006.334315

[Floyd et al. 2008] Floyd M.W., Esfandiari B., Lam K. , “A Case-based Reasoning Approach to Imitating RoboCup Players”, Proceedings of the Twenty-First International FLAIRS Conference (2008) aaai.org

[Frencken et al. 2011] Frencken W., Lemmink K., Delleman N., Visscher C., “Oscillations of centroid position and surface area of soccer teams in small-sided games.” , Eur J Sport Sci 11(4):215–223. doi:10.1080/17461391.2010.499967

[Fukushima et al. 2019a] Takuya Fukushima, Tomoharu Nakashima, and Hidehisa Akiyama, “Mimicking an Expert Team Through the Learning of Evaluation Functions from Action Sequences”, RoboCup 2018: Robot World Cup XXII (Lecture Notes in Computer Science Book 11374), 2019, ISBN 978-3-030-27543-3

[Fukushima et al. 2019b] Takuya Fukushima, Tomoharu Nakashima , and Hidehisa Akiyama, “Similarity Analysis of Action Trajectories Based on Kick Distributions”, LNAI 11531 RoboCup 2019: Robot World Cup XXIII (2019), ISBN 978-3-030-35698-9

[Gabel et al. 2021] Thomas Gabel, Philipp Kloppner, Yalcin Eren, Fabian Sommer, Steffen Breuer, Robert Litschel, Niklas M¨uller, Eicke Godehardt, “FRA-UNIted–team description 2021.”, In RoboCup 2021 Symposium and Competitions, Worldwide.
http://tgait.de/fileadmin/user_upload/documents/Gabel_EtAl_FU-20.pdf
http://tgabel.de/fileadmin/user_upload/documents/Gabel_EtAl_FU-20.pdf

[Gallo et al. 1993] Giorgio Gallo, Giustino Longo, Stefano Pallottino, Sang Nguyen, “Directed hypergraphs and applications”, Discrete Applied Mathematics Volume 42, Issues 2–3, 27 April 1993, Pages 177-201
https://doi.org/10.1016/0166-218X(93)90045-P

[Glorot and Bengio 2010] Glorot, X., & Bengio, Y. “Understanding the difficulty of training deep feedforward neural networks. “, In Proceedings of the thirteenth international conference on artificial intelligence and statistics (pp. 249-256). JMLR Workshop and Conference Proceedings. (2010)

[Glorot et al. 2011] Xavier Glorot, Antoine Bordes, Yoshua Bengio, “Deep Sparse Rectifier Neural Networks”, in Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, PMLR 15:315-323, 2011.

[Goodfellow et al. 2016] Ian Goodfellow, Yoshua Bengio, Aaron Courville. , “Deep Learning”, MIT Press ISBN-10: 0262035618, ISBN-13: 978-0262035613

[Goodfellow et al. 2020] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio, “Generative Adversarial Networks”, Communications of the ACMVolume 63 Issue 11 November 2020 pp 139–144 https://doi.org/10.1145/3422622

[Grunz et al. 2012] Andreas Grunz, Daniel Memmert, Jürgen Perl, “Tactical pattern recognition in soccer games by means of special self-organizing maps Human movement science”, 31(2):334-43 DOI: 10.1016/j.humov.2011.02.008

[Guestrin et al. 2002] C. Guestrin, M. G. Lagoudakis, and R. Parr, “Coordinated reinforcement learning,”, in Proc. 19th Int. Conf. Mach. Learn. (ICML-02), Sydney, Australia, Jul. 8–12, pp. 227–234.

[Grund 2012] Thomas U. Grund, “Network structure and team performance: The case of English Premier League soccer teams” ,Social Networks 34 (2012) 682–690

[Ha et al. 2016] David Ha, Andrew Dai, Quoc V. Le, “HyperNetworks.”, arXiv:1609.09106

[Harary 1969] Frank Harary, “Graph Theory”, Westview Press/Addison Wesley 1969

[Hastie et al. 2009] Hastie T. , Tibshirani R., Friedman J., “The Elements of Statistical Learning: Data Mining, Inference, and Prediction.”, Second Edition. Springer February 2009, ISBN 978-0-387-84858-7.
https://web.stanford.edu/~hastie/Papers/ESLII.pdf

[Hatcher 2001] Allen Hatcher, “Algebraic Topology”, Cambridge University Press 2001, ISBN 978-0521791601

[Hausknecht and Stone 2016] Matthew Hausknecht, Peter Stone, “Deep Reinforcement Learning in Parameterized Action Space”, arXiv:1511.04143v4

[He et al. 2016] He, K., Zhang, X., Ren, S., and Sun, J. , “Deep residual learning for image recognition”, In Proceedings of CVPR'2016, 770–778.

[Hettmansperger and McKean 2011] Thomas P. Hettmansperger, Joseph W. McKean, "Robust Nonparametric Statistical Methods" 2nd ed. , CRC Press, 2011, ISBN 978-1-4398-0908-2

[Hochreiter and Schmidhuber 1997] Hochreiter, S. and Schmidhuber, J. , “Long short-term memory”, Neural Computation 9, 8 (1997), 1735–1780.

[Huang et al. 2003] Huang Z., Yang Y., Chen X., “An approach to plan recognition and retrieval for multi-agent systems.” ,Proceedings of AORC (2003).

[Iglesias et al. 2008] Iglesias, J.A., Ledezma, A., Sanchis, A., Kaminka, G.A., “Classifying efficiently the behavior of a soccer team. “, Intelligent Autonomous Systems 10: IIAS-10, pp. 316–323 (2008)

[Iñiguez et al 2020] Iñiguez, G., Battiston, F. & Karsai, M., “Bridging the gap between graphs and networks.”, Commun Phys 3, 88 (2020). https://doi.org/10.1038/s42005-020-0359-6

[Ioffe and Szegedy 2015] Sergey Ioffe, Christian Szegedy, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”, Proceedings of the 32nd International Conference on Machine Learning, PMLR 37:448-456, 2015.

[Irpan 2018] Alex Irpan, “Deep Reinforcement Learning Doesn't Work Yet”, (2018) https://www.alexirpan.com/2018/02/14/rl-hard.html

[Johnson 2001] Johnson J. , “Visual communication in swarms of intelligent robot agents”, Artificial Life and Robotics March 2001, Volume 5, Issue 1, pp 1–9

[Johnson 2006] Johnson J. , “Hypernetworks for reconstructing the dynamics of multilevel systems”, European Conference on Complex Systems 2006, 25-29 Sep 2006, Oxford.

[Johnson 2013] Johnson J. , “Hypernetworks in the Science of Complex Systems”, 2013, Imperial College Press, 978-1-86094-972-2

[Johnson & Iravani 2007] Johnson J., Iravani P. , “The Multilevel Hypernetwork Dynamics of Complex Systems of Robot Soccer Agents”, ACM Transactions on Autonomous and Adaptive Systems, Volume 2 Issue 2, June 2007

[Johnson & Rossi 2018] Jeffrey H. Johnson, Ruggero Rossi , “Dynamic Structures for Evolving Tactics and Strategies in Team Robotics”, 2018, in Proceedings of the 2018 International Conference on Artificial Life and Robotics, ISBN 978-4-9908350-3-3 2019, Journal of Robotics Networking and Artificial Life 6(3):203, DOI:10.2991/jrnal.k.191203.005

[Johnson & Rossi 2020] J. Johnson, R. Rossi, “A Structural Language for Multilevel Dynamics in the Design of Robot Soccer Systems”, Proceedings of International Conference on Artificial Life and Robotics 25:270-276, DOI:10.5954/ICAROB.2020.PS-3

[Kalyanakrishnan et. al 2007] Shivaram Kalyanakrishnan, Yaxin Liu, and Peter Stone, “Half Field Offense in RoboCup Soccer: A Multiagent Reinforcement Learning Case Study”, RoboCup 2006: Robot Soccer World Cup X. RoboCup 2006. Lecture Notes in Computer Science, vol 4434. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-74024-7_7

[Kalyviotis and Hu 2002] Nikolaos Kalyviotis and Huosheng Hu, “A Neural Network decision module for the Adaptive Strategic Planning Framework in RoboCup”, Proc. of International Symposiumon Robotics and Automation, Toluca, Mexico, 1-4 September 2002

[Kaminka et al. 2003] Kaminka, G.A., Fidanboylu, M., Chang, A., Veloso, M., “Learning the sequential coordinated behavior of teams from observations.”, In: RoboCup 2002. LNCS (LNAI), vol. 2752, pp. 111–125. Springer (2003)

[Karimi & Ahmazadeh 2014] Karimi M., Ahmazadeh M. , “Mining RoboCup Log Files to Predict Own and Opponent Action”, International Journal of Advanced Research in Computer Science, Vol 5 No. 6 (2014)

[Kingma and Ba 2014] D.P. Kingma, J. Ba , “Adam: A Method for Stochastic Optimization”, Proceedings of the 3rd International Conference on Learning Representations, 2014

[Kingma and Welling 2014] Kingma, D. and Welling, M. “Auto-encoding variational bayes.”, In Proceedings of the Intern. Conf. Learning Representations, 2014.

[Kitano, et al. 1995] Kitano, H. and Asada, M. and Kuniyoshi, Y. and Noda, I. and Osawa, E., “RoboCup: The Robot World Cup Initiative”, IJCAI-95 Workshop on Entertainment and AI/Alife , 1995

[Kohonen 1990] T. Kohonen, “The self-organizing map”, in Proceedings of the IEEE (Volume: 78, Issue: 9, Sep 1990) pp. 1464 - 1480, DOI: 10.1109/5.58325

[Kostiadis and Hu 1999] Kostas Kostiadis and Huosheng Hu, “Reinforcement Learning and Co-operation in a Simulated Multi-agent System”, Proceedings 1999 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human and Environment Friendly Robots with High Intelligence and Emotional Quotients, 10.1109/IROS.1999

[Kostiadis and Hu 2001] Kostas Kostiadis and Huosheng Hu, “KaBaGe-RL: Kanerva-based Generalisation and Reinforcement Learning for Possession Football”, Proc. of IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems, 2001.

[Kuhlmann et al. 2005] Kuhlmann G., Stone P., Lallinger J. , “The UT Austin Villa 2003 Champion Simulator Coach: A Machine Learning Approach”, RoboCup 2004: Robot Soccer World Cup VIII, Volume 3276 of the series Lecture Notes in Computer Science (2005) pp 636-644

[Kurach et al. 2019] Karol Kurach, Anton Raichuk, Piotr Stańczyk, Michał Zając, Olivier Bachem, Lasse Espeholt, Carlos Riquelme, Damien Vincent, Marcin Michalski, Olivier Bousquet, Sylvain Gelly, “Google Research Football: A Novel Reinforcement Learning Environment”, arXiv:1907.11180

[Larik & Haider 2015] Larik S.A., Haider S., “Opponent Classification in Robot Soccer” Current Approaches in Applied Artificial Intelligence 28th International Conference IEA/AIE Seoul 2015

[Lattner et al. 2006] Lattner A. D., Miene A., Visser U., Herzog O., “Sequential Pattern Mining for Situation and Behavior Prediction in Simulated Robotic Soccer”, RoboCup 2005: Robot Soccer World Cup IX, Volume 4020 of the series Lecture Notes in Computer Science pp 118-129, 2006

[Latzke et al. 2007] Tobias Latzke, Sven Behnke, Maren Bennewitz, “Imitative Reinforcement Learning for Soccer Playing Robots”, RoboCup 2006: Robot Soccer World Cup X. , 2007

[Laviers et al. 2009] Laviers, K., Sukthankar, G., Klenk, M., Aha, D. W., & Molineaux, M. “Opponent modeling and spatial similarity to retrieve and reuse superior plays.”, KNEXUS RESEARCH CORP SPRINGFIELD VA. (2009) http://www.dtic.mil/cgi-bin/GetTRDoc?AD=ADA593111

[Le et. al 2011] Le Q. V., Zou W. Y., Yeung S. Y., Ng A. Y. , “Learning hierarchical invariant spatio-temporal features for action recognition with independent subspace analysis” IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2011 DOI: 10.1109/CVPR.2011.5995496

[LeCun et al. 2010] Yann LeCun, Koray Kavukcuoglu and Clement Farabet, “Convolutional networks and applications in vision”, Proceedings of 2010 IEEE International Symposium on

Circuits and Systems, DOI: 10.1109/ISCAS.2010.5537907, ICLR Workshop, 2013, arXiv preprint arXiv:1301.3781.

[Leottau et al. 2015] Leottau, L., Celemin, C., & Ruiz-del-Solar, J. , “Ball Dribbling for Humanoid Biped Robots: A Reinforcement Learning and Fuzzy Control Approach.”, RoboCup 2014. Lecture Notes in Computer Science, vol 8992. Springer, Cham. (2015), https://doi.org/10.1007/978-3-319-18615-3_45

[Levene 1960] Howard Levene, "Robust tests for equality of variances". In Ingram Olkin; Harold Hotelling; et al. (eds.). Contributions to Probability and Statistics: Essays in Honor of Harold Hotelling. Stanford University Press. pp. 278–292. (1960)

[Littman 1994] Littman, Michael L. , “Markov games as a framework for multi-agent reinforcement learning.”, Machine learning proceedings 1994. Morgan Kaufmann, 1994. 157-163.
https://courses.cs.duke.edu//spring07/cps296.3/littman94markov.pdf

[Lobos-Tsunekawa et al. 2018] Kenzo Lobos-Tsunekawa, David L. Leottau, Javier Ruiz-del-Solar, “Toward Real-Time Decentralized Reinforcement Learning Using Finite Support Basis Functions”, RoboCup 2017: Robot World Cup XXI, 11175, 95. (2018)

[Loshchilov & Hutter 2019] Ilya Loshchilov & Frank Hutter "Decoupled Weight Decay Regularization", ICLR 2019

[Ma and Cameron 2008] Ma, J., & Cameron, S. , “Combining policy search with planning in multi-agent cooperation.”, LNAI. RoboCup 2008: robot soccer world cup XII, Suzhou, China. Berlin: Springer.

[MacAlpine et al. 2015] MacAlpine, Patrick, Depinet, Mike, and Stone, Peter, “UT Austin Villa 2014: RoboCup 3D simulation league champion via overlapping layered learning.”, In Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI), January 2015.

[Mackworth 1993] Mackworth, A. K. , “On seeing robots.”, In Computer Vision: Systems, Theory and Applications (pp. 1-13). World Scientific Press

[Mann 2013] Prem S. Mann, “Introductory Statistics” 8th ed., Wiley 2013

[Masson et al. 2015] Warwick Masson, Pravesh Ranchod, George Konidaris, “Reinforcement Learning with Parameterized Actions”
CoRR, abs/1509.01644, 2015. http://arxiv.org/abs/1509.01644

[Matsubara et al. 1999] Matsubara H., Frank I., Tanaka-Ishii K., Noda I., Nakashima H., Hasida K. , “Automatic Soccer Commentary and RoboCup”, RoboCup-98: Robot Soccer World Cup II , Volume 1604 of the series Lecture Notes in Computer Science pp 34-49, 1999

[Merke and Riedmiller 2001] A. Merke and M. A. Riedmiller, “Karlsruhe brainstormers—A reinforcement learning approach to robotic soccer,” in Robot Soccer World Cup V (RoboCup 2001). Lecture Notes in Computer Science, vol. 2377, Washington, DC, Aug. 2–10, pp. 435–440

[Michael et al. 2018] Olivia Michael, Oliver Obst, Falk Schmidsberger, Frieder Stolzenburg, “Analysing Soccer Games with Clustering and Conceptors”, RoboCup 2017: Robot World Cup XXI (2018), ISBN 978-3-030-00307-4

[Mikolov et al. 2013] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. “Efficient estimation of word representations in vector space.”

[Mitchell 1997] Tom M. Mitchell, “Machine Learning”
McGraw-Hill ISBN - 9780070428072, 0070428077

[Mnih et al. 2013] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. , “Playing Atari with Deep Reinforcement Learning.” (2013), arXiv:1312.5602.

[Monajjemi et al. 2012] Monajjemi, V., Koochakzadeh, A., Ghidary, S.S. , “GrSim – Robocup Small Size Robot Soccer Simulator”, in RoboCup 2011: Robot Soccer World Cup XV. pp. 450–460. Springer Berlin Heidelberg,
Berlin, Heidelberg (2012)

[Mota et a. 2010] Mota L. , Lau N., Reis L. P. , “Co-ordination in RoboCup's 2D simulation league: Setplays as flexible, multi-robot plans “, IEEE Conference on Robotics Automation and Mechatronics (RAM), 2010 , DOI: 10.1109/RAMECH.2010.5513166

[Munkres 1999] James R. Munkres, “Topology 2nd ed.”, Pearson, ISBN 978-0131816299

[Muzio et al. 2020] Alexandre F. V. Muzio, Marcos R. O. A. Maximo, Takashi Yoneyama, “Deep Reinforcement Learning for Humanoid Robot Dribbling”, 2020 Latin American Robotics Symposium (LARS), 2020 Brazilian Symposium on Robotics (SBR) and 2020 Workshop on Robotics in Education (WRE), 2020, pp. 1-6, doi: 10.1109/LARS/SBR/WRE51543.2020.9307084.

[Nakanishi et al. 2008] Nakanishi R., Murakami K., Naruse T., “Dynamic positioning method based on dominant region diagram to realize successful cooperative play.” , In: Visser U, Ribeiro F, Ohashi T, Dellaert F (eds) Robo cup 2007: Robot Soccer
World Cup XI, Vol 5001. Springer, Berlin, pp 488–495

[Nakanishi et al. 2010] Nakanishi R., Maeno J., Murakami K., Naruse T. , “An Approximate Computation of the Dominant Region Diagram for the Real-Time Analysis of Group Behaviors”, RoboCup 2009: Robot Soccer World Cup XIII
Volume 5949 of the series Lecture Notes in Computer Science pp 228-239

[Nakashima et al. 2015] Tomoharu Nakashima, Satoshi Mifune, Jordan Henrio, Oliver Obst, Peter Wang, Mikhail Prokopenko, “Kick Extraction for Reducing Uncertainty in RoboCup Logs” , HIMI 2015: Human Interface and the Management of Information. Information and Knowledge in Context pp 622-633

[Ng and Russell, 2000] Ng, A. Y. and Russell, S. (2000). “Algorithms for inverse reinforcement learning.” , In Proc. International Conference on Machine Learning, pages 663–670

[Ocana et al 2019] Jim Martin Catacora Ocana, Francesco Riccio, Roberto Capobianco, and Daniele Nardi, “Cooperative Multi-agent Deep Reinforcement Learning in a 2 Versus 2 Free-Kick Task”, LNAI 11531 RoboCup 2019: Robot World Cup XXIII (2019)
ISBN 978-3-030-35698-9

[Oishi & Sugihara 1995] Oishi Y., Sugihara K. , “Topology-Oriented Divide-and-Conquer Algorithm for Voronoi Diagrams”, Article in Graphical Models and Image Processing 57(4):303-314 July 1995

DOI: 10.1006/gmip.1995.1027

[Ollino et al. 2018] Franco Ollino, Miguel A. Solis, Héctor Allende, “Batch Reinforcement Learning on a RoboCup Small Size League keepaway strategy learning problem”, Proceedings of the 4th Congress on Robotics and Neuroscience

[OpenAI 2019] OpenAI: Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique P. d.O. Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, Susan Zhang, “Dota 2 with Large Scale Deep Reinforcement Learning”, arXiv:1912.06680

[Pena et al. 2020] Pena, C. H., Machado, M. G., Barros, M. S., Silva, J. D., Maciel, L. D., Ren, T. I., Bassani, H. F. , “An Analysis of Reinforcement Learning Applied to Coach Task in IEEE Very Small Size Soccer.” ,In 2020 Latin American Robotics Symposium (LARS), 2020 Brazilian Symposium on Robotics (SBR) and 2020 Workshop on Robotics in Education (WRE) (pp. 1-6). IEEE.

[Perl 2002] Perl J. , “Game analysis and control by means of continuously learning networks.” Int J Perform Anal Sport 2002; 2: 21-35

[Perl 2004] Perl, J. , “A neural network approach to movement pattern analysis.” Human Movement Science, 23, 605–620.

[Perl & Dauscher 2006] Perl J., Dauscher P. , “Dynamic pattern recognition in sport by means of artificial neural networks.” In: Computational intelligence for movement science. Hershey (PA): Idea Group Publishing, 2006: 299-318

[Peters and Schaal 2008] Jan Peters, Stefan Schaal , “Reinforcement learning of motor skills with policy gradients.” , Neural networks, 21(4), 682-697. (2008)

[Picton 1994] Phil Picton, “Introduction to Neural Networks”, Macmillan 1994

[Pomas and Nakashima 2019] Tanguy Pomas and Tomoharu Nakashima, “Evaluation of Situations in RoboCup 2D Simulations Using Soccer Field Images”, RoboCup 2018: Robot World Cup XXII (Lecture Notes in Computer Science Book 11374), 2019

[Pretolani 2013] Pretolani, D. , “Finding hypernetworks in directed hypergraphs.” European Journal of Operational Research, 230(2), 226-230. (2013)

[Raines et al. 2000] Raines T., Tambe M, Marsella S. , “Automated assistants to aid humans in understanding team behaviors.”, In: Proceedings of the fourth international conference on Autonomous agents. ACM, 2000. p. 419-426.

[Ramos & Ayanegui 2008] Ramos F. and Ayanegui H., “Discovering Tactical Behavior Patterns Supported by Topological Structures in Soccer Agent Domains. “, In: International Conference on Autonomous Agents, Proceedings of the 7th International joint conference on Autonomous Agents and Multiagent Systems, Estoril, vol. 3, pp. 1421–1424 (2008)

[Ramos et al. 2017] Ramos, J., Lopes, R. J., Marques, P., Araújo, D., “Hypernetworks Reveal Compound Variables That Capture Cooperative and Competitive Interactions in a Soccer Match”, Frontiers in Psychology, 28 Aug 2017

[Ramos et al. 2018] João Ramos, Rui J. Lopes, Duarte Araúj, “What’s Next in Complex Networks? Capturing the Concept of Attacking Play in Invasive Team Sports”, Sports Med. 2018 Jan, DOI 10.1007/s40279-017-0786-z

[Rein & Memmert 2016] Robert Rein and Daniel Memmert, “Big data and tactical analysis in elite soccer: future challenges and opportunities for sports science”, SpringerPlus (2016) 5:1410 DOI 10.1186/s40064-016-3108-2

[Reis and Lau 2001] Paulo Reis L. P. and Lau N., “FC Portugal Team Description: RoboCup 2000 Simulation League Champion”, RoboCup 2000: Robot Soccer World Cup IV pp. 29-40 ISBN 3-540-42185-8 Springer-Verlag 2001

[Reynolds 1987] Reynolds C. W. , “Flocks, herds and schools: A distributed behavioral mode”, SIGGRAPH '87 Proceedings of the 14th annual conference on Computer graphics and interactive techniques pp. 25-34, ISBN:0-89791-227-6 doi:10.1145/37401.37406

[Ribeiro et al. 2019] Ribeiro, J., Davids, K., Araújo, D. et al. The Role of Hypernetworks as a Multilevel Methodology for Modelling and Understanding Dynamics of Team Sports Performance. Sports Med 49, 1337–1344 (2019). https://doi.org/10.1007/s40279-019-01104-x

[Riedmiller and Gabel 2006] M. Riedmiller and T. Gabel, “Brainstormers 2D – Team Description 2006”
https://ml.informatik.uni-freiburg.de/former/_media/publications/bs06main.pdf

[Riedmiller et al. 2009] Martin Riedmiller, Thomas Gabel, Roland Hafner, Sascha Lange , “Reinforcement learning for robot soccer”, Auton Robot (2009) 27: 55–73, DOI 10.1007/s10514-009-9120-4

[Rumelhart et al. 1986] David E. Rumelhart, Geoffrey E. Hinton & Ronald J. Williams, “Learning representations by back-propagating errors”, Nature volume 323, pages533–536 (1986)

[Rucco et al. 2015] Matteo Rucco, David Sousa-Rodrigues, Emanuela Merelli, Jeffrey H. Johnson, Lorenzo Falsetti, Cinzia Nitti & Aldo Salvi, “Neural hypernetwork approach for pulmonary embolism diagnosis”, BMC Research Notes, volume 8, Article number: 617 (2015)
https://bmcresnotes.biomedcentral.com/articles/10.1186/s13104-015-1554-5

[Sabour et al. 2017] Sara Sabour, Nicholas Frosst, Geoffrey E Hinton, “Dynamic Routing Between Capsules”, arXiv:1710.09829 [cs.CV]

[Salakhutdinov 2015] Ruslan Salakhutdinov, “Learning Deep Generative Models”, Annu. Rev. Stat. Appl. 2015. 2:361–85, The Annual Review of Statistics and Its Application, doi: 10.1146/annurev-statistics-010814-020120

[Saxe et al. 2013] Saxe, A. M., McClelland, J. L., & Ganguli, S. ,”Exact solutions to the nonlinear dynamics of learning in deep linear neural networks.” ICLR (2013), arXiv preprint arXiv:1312.6120.

[Scardua et al. 2000] Leonardo A. Scardua, Anna H. Reali Costa, and Jose Jaime da Cruz , “Learning to Behave by Environment Reinforcement”, in Lecture Notes in Artificial Intelligence 185 RoboCup-99 Robot Soccer World Cup III (2000)

[Schwab et al. 2019] Devin Schwab , Yifeng Zhu , and Manuela Veloso, “Learning Skills for Small Size League RoboCup “, in RoboCup 2018: Robot World Cup XXII (Lecture Notes in Computer Science Book 11374), 2019, ISBN 978-3-030-27543-3

[Schulman et al. 2015] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, Pieter Abbeel , “Trust Region Policy Optimization”, In International conference on machine learning (pp. 1889-1897). PMLR, arXiv:1502.05477

[Schulman et al. 2016] John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan and Pieter Abbeel, “High-Dimensional Continuous Control Using Generalized Advantage Estimation”, ICLR 2016, arXiv:1506.02438

[Schulman et al. 2017] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). , “Proximal policy optimization algorithms” , arXiv preprint arXiv:1707.06347.

[Shahri 2008] Amin Habibi Shahri, “An Introduction to a New Commentator for RoboCup 3D Soccer Simulation”, SIMPAR 2008 (ed. Carpin, Noda, Pagello, REggiani, von Stryk) , LNAI 5325 (ed. Goebel, Siekmann, Wahlser), pp. 283-292

[Shapiro and Wilk 1965] Shapiro, S. S., & Wilk, M. B., "An analysis of variance test for normality (complete samples)", Biometrika, 52(3/4), 591-611, (1965)

[Silva et al. 2014] Silva P., Travassos B., Vilar L., Aguiar P., Davids K., Araujo D., Garganta J., “Numerical relations and skill level constrain co-adaptive behaviors of agents in sports teams.”, PLoS One 9(9):e107112. doi:10.1371/journal. pone.0107112

[Sorrentino 2012] Francesco Sorrentino, “Synchronization of hypernetworks of coupled dynamical systems”, New J. Phys. 14 033035 (2012)

[Sprado and Gottfried 2009] Sprado J. and Gottfried B., “What Motion Patterns Tell Us about Soccer Teams”, in RoboCup 2008: Robot Soccer World Cup XII, Volume 5399 of the series Lecture Notes in Computer Science pp 614-625

[Srivastava et al. 2014] Nitish Srivastava,Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov, “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”, Journal of Machine Learning Research 15 (2014) 1929-1958

[Stone and Sutton 2001] Peter Stone and Richard S. Sutton, “Scaling Reinforcement Learning toward RoboCup Soccer”
In The Eighteenth International Conference on Machine Learning (ICML 2001)
pp. 537-544, Williamstown, MA, USA, June 2001.

[Stone and Sutton 2002] Peter Stone and Richard S. Sutton , “Keepaway soccer: A machine learning testbed. “, RoboCup-2001: Robot soccer world cup V (pp.214–223). Berlin: Springer. 2002

[Stone and Veloso 1999] P. Stone and M. Veloso, “Team-partitioned, opaque-transition reinforcement learning,” in Proc. 3rd Int. Conf. Auton. Agents (Agents-99), Seattle, WA, May 1–5, pp. 206–212.

[Stone et al. 1999] Stone P., Veloso M., Riley P., “The CMUnited-98 Champion Simulator Team”, RoboCup-98: Robot Soccer World Cup II, Volume 1604 of the series Lecture Notes in Computer Science pp 61-76, 1999

[Stone et al. 2005] Peter Stone, Richard S. Sutton, Gregory Kuhlmann, “Reinforcement Learning for RoboCup Soccer Keepaway”, Adaptive Behaviour ,Vol 13, Issue 3, 2005

[Sutton & Barto 2018] Richard C. Sutton, Andrew C. Barto, “Reinforcement Learning - An introduction. 2nd Ed.”, MIT Press - ISBN: 9780262039246

[Suzuki and Nakashima 2019] Yudai Suzuki and Tomoharu Nakashima, “On the Use of Simulated Future Information for Evaluating Game Situations”, LNAI 11531 RoboCup 2019: Robot World Cup XXIII (2019), ISBN 978-3-030-35698-9

[Szepesvári 2010] Csaba Szepesvári, “Algorithms for Reinforcement Learning”, Morgan & Claypool 2010, ISBN: 9781608454921

[Taki & Hasegawa 1998] Taki T., Hasegawa J., “Dominant region: a basic feature for group motion analysis and its application to teamwork evaluation in soccer games”
Proc. SPIE 3641, Videometrics VI, 48 (December 14, 1998); doi:10.1117/12.333797

[Tieleman and Hinton 2012] T Tieleman, G Hinton, “Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude”, COURSERA: Neural networks for machine learning 4 (2), 26-31

[Torrey et al. 2005] Torrey, L., Walker, T., Shavlik, J., Maclin, R., “Using advice to transfer knowledge acquired in one reinforcement learning task to another.”, ECML 2005. LNCS, vol. 3720, pp. 412–424. Springer, Heidelberg, (2005). https://doi.org/10.1007/11564096 40

[Uchibe et al. 1996] Uchibe, E.; Asada, M.; and Hosoda, K., “Behavior coordination for a mobile robot using modular reinforcement learning.” In Proc. of the 1996 IEEE/RSJ International Conference on Intelligent Robots and Systems, 1329–1336.

[Ullmann 1976] Ullmann, J.R., “An algorithm for subgraph isomorphism.”
J. ACM 23(1), 31–42

[Uthmann & Dauscher 2005] Uthmann, T., & Dauscher, P., “Analysis of motor control and behavior in multi agent systems by means of artificial neural networks.”, Clinical Biomechanics, 20(2), 119-125.

[van Hasselt 2010] Hado van Hasselt, “Double Q-learning.”
Advances in neural information processing systems, 23, 2613-2621. (2010)

[Vapnik 2000] Vladimir N. Vapnik, “The Nature of Statistical Learning Theory 2nd ed.” Springer-Verlag. ISBN 978-0-387-98780-4.

[Visser et al. 2001] Ubbo Visser, Christian Drucker, Sebastian Hubner, Esko Schmidt, Hans-Georg Weland, “Recognizing Formations in Opponent Teams”, in RoboCup-2000: Robot Soccer World Cup IV, P. Stone, T. Balch, and G. Kraetschmar, Editors. 2001, Springer-Verlag

[Vlassis 2007] Nikos Vlassis , “A Concise Introduction to Multiagent Systems and Distributed Artificial Intelligence”, Morgan & Claypool publishers 2007, ISBN ISBN: 9781598295269

[Voelz et al. 1999] Voelz D., André E., Herzog G., Rist T., “Rocco: a Robocop Soccer Commentator System”, RoboCup-98: Robot Soccer World Cup II, Volume 1604 of the series Lecture Notes in Computer Science pp 50-60, 1999

[Volpentesta 2008] Volpentesta, A. P., “Hypernetworks in a directed hypergraph.”, European Journal of Operational Research, 188(2), 390-405. (2008)

[Warncke & Uhrmacher 2016] Tom Warncke and Adelinde Uhrmacher, “Spatiotemporal Pattern Matching in RoboCup”, MATES 2016, Multiagent System Technologies, Springer International, ISBN: 978-3-319-45889-2

[Wasserman 2004] Larry Wasserman, “All of Statistics”, Springer 2004

[Watkins 1989] Christopher J. C. H. Watkins, “Learning from delayed rewards.”, PhD Thesis, King’s College (1989).

[Watkinson and Cam 2019] Warren Blair Watkinson II, Tracy Cam, “Training a RoboCup Striker Agent via Transferred Reinforcement Learning”, RoboCup 2018: Robot World Cup XXII (Lecture Notes in Computer Science Book 11374), 2019, ISBN 978-3-030-27543-3

[Wiering et al. 1999] M. Wiering, R. Salustowicz, and J. Schmidhuber, “Reinforcement learning soccer teams with incomplete world models,” Auton. Robots, vol. 7, no. 1, pp. 77–88, 1999.

[Williams 1992] Williams, R. J. , “Simple statistical gradient-following algorithms for connectionist reinforcement learning.”, Machine learning, 8(3), 229-256. (1992)

[Wünstel et al. 2001] Wünstel, M., Polani, D., Uthmann, T., & Perl, J., “Behavior classification with self organizing maps.”, In RoboCup 2000 (pp. 108-118), Berlin; Heidelberg; New York: Springer. ISBN 978-3-540-45324-6

[Yoon et al. 2016] M. Yoon, J. Bekker, and S. Kroon, “New reinforcement learning algorithm for robot soccer,”, ORiON, vol. 33, 02 2016., ISSN 2224–0004

[Zare et al. 2018] Nader Zare, Mohsen Sadeghipour, Ashkan Keshavarzi,Mahtab Sarvmeili, Amin Nikanjam, Reza Aghayari, Arad Firouzkoohi, Mohammad Abolnejad, Sina Elahimanesh, Amin Akhgar, “Cyrus 2D Simulation Team Description Paper 2018“ https://archive.robocup.info/Soccer/Simulation/2D/TDPs/RoboCup/2018/CYRUS_SS2D_RC2018_TDP.pdf

[Zeiler and Fergus 2014] Matthew D Zeiler, Rob Fergus, “Visualizing and Understanding Convolutional Networks”, arXiv:1311.2901 [cs.CV]

[Ze-Kai et al. 2013] Ze-Kai C., Qian L., Feng Q., “Research and Application of Data Mining Based on RoboCup Soccer Logs”, Computational and Information Sciences (ICCIS), 2013 Fifth International Conference on Computational and Information Sciences, DOI: 10.1109/ICCIS.2013.104

[Zhang et al. 2021] Kaiqing Zhang, Zhuoran Yang, Tamer Başar, “Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms”, arXiv:1911.10635

[Zhu et al. 2019] Zhu, Y., Schwab, D., & Veloso, M. , “Learning primitive skills for mobile robots.”, In 2019 International Conference on Robotics and Automation (ICRA) (pp. 7597-7603). IEEE. (2019)

Web References

[1] RoboCup Soccer
https://www.robocup.org/leagues/24

[2] RoboCup Log Files
https://archive.robocup.info/Soccer/Simulation/2D/logs/RoboCup/
http://chaosscripting.net/files/competitions/RoboCup/WorldCup/

[3] Chyronhego Tracab
https://tracab.com/products/tracab-technologies/

[4] Neo4j
https://neo4j.com/

[5] RoboCup 2D software and manual repository
https://github.com/rcsoccersim/
https://rcsoccersim.github.io/manual/
https://sourceforge.net/projects/sserver/files/

[6] All the Robocup participant binaries by year
https://archive.robocup.info/Soccer/Simulation/2D/binaries/RoboCup/

[7] Wrighteagle2d binary (the official team executable) and WrighteagleBase source code (a stripped down version with only very basic functionality, to use as a base for new agents)
https://github.com/wrighteagle2d

[8] librcsc: lib to ease the communication with robocup server
https://osdn.net/projects/rctools/downloads/51941/librcsc-4.1.0.tar.gz/

[9] agent2d: source code to ease the creation of a player for Robocup2D
https://osdn.net/projects/rctools/downloads/55186/agent2d-3.1.1.tar.gz

[10] Gliders2D open source player agent
http://www.prokopenko.net/gliders2d.html

[11] Robosoc2D (the 2d robotic soccer simulator written by the author of this PhD thesis
https://github.com/rug/robosoc2d

[12] DeepMind MuJoCo Multi-Agent Soccer Environment.
https://deepmind.com/research/open-source/mujoco-soccer-environment

[13] FIRASim
https://github.com/fira-simurosot/FIRASim

[14] VSS-SDK
https://vss-sdk.github.io/book/general.html

[15] RoboCup 3D simulator rcssserver3d
https://gitlab.com/robocup-sim/SimSpark/-/tree/master/rcssserver3d

[16] OpenAi
Vanilla policy gradient and Expected Grad-Log-Prog Lemma:

https://spinningup.openai.com/en/latest/spinningup/rl_intro3.html

[17] OpenAi
Reward-to-go proof
https://spinningup.openai.com/en/latest/spinningup/extra_pg_proof1.html

[18] OpenAI
Q-function in policy gradient proof:
https://spinningup.openai.com/en/latest/spinningup/extra_pg_proof2.html

[19] Dennis Soemers
Reward-to-go proof
https://ai.stackexchange.com/questions/9614/why-does-the-reward-to-go-trick-in-policy-gradient-methods-work/10369

[20] CMake
https://cmake.org/

[21] OpenAI
Spinning Up in Deep RL
https://github.com/openai/spinningup
https://spinningup.openai.com/

[22] OpenAI
OpenAI Five
https://openai.com/blog/openai-five/

[23] Robocup History
https://www.robocup.org/a_brief_history_of_robocup

[24] FIRA WorldCup
https://firaworldcup.org/