Learning MSL Team Behavior from Digitized Game Logs

Peter van Lith
Technische Universiteit Eindhoven
vanlith.peter@gmail.com

March 18, 2020

Abstract

First Draft

This is a first draft of a paper, describing a project to learn the behavior of competing MSL Soccer Teams. This draft is meant to explain the way the project is being executed and will capture results and insights as they are achieved.

The goal of this work is to create a system, consisting of a collection of networks that learns the behavior of competing teams from digitized log files of past competitions. This encapsulated behavior will be used in the TURTLE simulation environment, to test new approaches against our known competitors. The input consists of log files of actual competitions, containing position details of all players, the ball and information about the game situation like ball possession and starting conditions like kickoff, corner and throw-in. The output of the network consists of the actions of all team players, including ball movements and an indication if the current situation represents a threat or an opportunity.

Current Status

This project is in an early stage. The document serves as a design blueprint and also as a report of the achievements of the project as we go along. It contains short descriptions of all consulted papers as well as considerations why we took certain decisions. At a later stage parts of this document will serve as input for one or more papers to be submitted for publication.

The Network based simulation environment is working, as well as the facilities to read in the log data and select samples from it. A first version of the Set-Pieces editor is working and part of the KNVB Futsal Coaching Manual examples are included.

Where we first concentrated on using a DDRQN neural network, we are now extending this with additional facilities to provide more detailed analysis and an explanatory capability. We are concentrating on learning the intentions of individual players in time-series of actions in order to better describe a team's strategy. To this end we investigate techniques like Conditional Random Fields (CRF) and Hierarchical Attention Networks (HAN) to allow descriptions of actions on a symbolic level, where symbols are grounded in the soccer-playing world. A first DDRQN network has been coded and is working, but we need to extend this with the facilities to learn intentions.

The first part we are developing is the classification of Game Situations, which includes Formation Analysis. We found many more articles on this subject, mainly from the Sports Analysis community, now included in this study.

To anticipate the forthcoming TensorFlow 2.0 version, the current implementation needs to be re-implemented in Keras instead of native TensorFlow.

1. Preamble

First we describe the motivation and design goals for the intended system. We will then explain the way in which we use the information, collected in digitized log files from competitions of the past years. We then describe how a collection of Neural Networks captures the responses of our competitors to game situations. Finally we explain how this network is used in a simulation environment, developed to test our approach.

1.1. Reinforcement Learning

Many different approaches to learning have been explored in the past 10 to 20 years. Robotic Soccer has been a popular subject for this topic and in this paper we explore the most important ones. We intend to use the most suitable of these approaches to create a system that is capable of learning a model of the behavior of competing teams. Although Opponent Modeling is a frequently explored topic, almost all approaches concentrate on implicit models to let a system learn the best policy. We want to concentrate on learning an explicit model, in which we capture the strategy of competing teams and we want to distill from such models not only the actions of our opponents but also to find the changes, teams have made during successive years. This information will help us to determine our and their weak and strong points and to decide, which parts of our strategy we need to concentrate on. Only after having mastered building such models, will we be in a position to let our robots

learn to improve their performance during competitions by learning from their mistakes. Such a real-time continuous learning approach is known as Continual Learning.

1.2. Continual Learning

The long-term goal of the study to learning approaches is to allow our MSL Turtle robots to learn how to play soccer. Much research into learning how to play soccer has already been done, but generally in different environments like the Simulation League and the Small Size League. The Midsize League differs from these in that all robots are fully autonomous and communicate with each other, while the other approaches work in a shared environment. Most learning approaches are based on batch environments, in which many examples are presented at the same time. This type of network was used in an earlier phase of this project in which we attempted to recognize and classify robots, playing in a competition [104]. In that case, pre-collected images were used to train a static neural network that was capable of identifying and localizing robots, based on their team membership and their position in the camera image.

When confronted with ongoing competitions, such an approach is not feasible. Robots need to be able to learn from situations encountered during competitions, which means that new situations are continuously presented by the robot's sensors and camera. This presents a big problem for existing learning systems in the following areas:

1. *High impact events cannot be handled well*. All experienced events are treated with the same importance. In real-live situations high-impact events should be learned much faster than low-impact events. Current neural networks cannot make this distinction.
2. *Network reconfiguration*. As new situations occur, new classes may be needed, which means that the network architecture must change. With current technology this means that a new architecture is created manually and all learned situations must be trained again. It also means that all input situations need to be kept for later re-training. Newer approaches allow dynamic expansion of the network and retrain only those neurons that are affected by the new information.
3. *Catastrophic Forgetting*. When a network is trained with new events, existing networks quickly forget older events. To prevent this, older events need to be kept and the network needs to be retrained. Some newer approaches use the trained weights of the network to reconstruct the learned inputs and periodically retrain the network to preserve previously learned events.

These problems lead to an entirely new way of constructing and training neural networks, collected in approaches, known as zero-shot and one-shot learning, continual learning and lifelong learning. We will explore these new approaches in the separate self-learning mini-Turtle project. In the current project we will be using batch-oriented networks only, but we will keep our options open to allow later integration of these techniques.

1.3. Reinforcement and Q-Learning concepts

Before going into details about the way that we intend to build an Explicit Opponent Model, we will first explain a number of important concepts, found in most existing approaches, as to familiarize readers with the most relevant terminology. We will frequently find the following keywords in the literature:

1. Markov Decision Processes
2. The Bellman Equation
3. On-policy and Off-policy learning
4. Model-based and Model-free learning
5. Discrete actions and Continuous actions

But first we must make an important distinction about the nature of our learning task. Most contemporary learning systems are aimed at finding an optimal strategy by exploring possible actions. Exploration is an important aspect of these systems that allow them to find better solutions by randomly searching the possible action space. In our case we want to build a model that is the best possible representation of the opponent's strategy, including its weak spots. So we do not seek to improve on it; on the contrary, we want to create a model that offers an explanation of an opponent's behavior.

That means that many of the elements of Reinforcement Learning strategies are irrelevant in our case. Especially exploration is an undesirable property that we need to eliminate from the equation. So therefore we explore the field of Reinforcement Learning with a different attitude and want to keep all elements that are helpful in learning from examples, found in our logfiles, while ignoring all attempts to improve on them.

1.3.1 Markov Decision Processes

A Markov Decision Process (MDP) is a mathematical method to describe decision processes that are non-deterministic. It is used in many disciplines and in AI systems like speech recognition. It forms the basis for many Reinforcement Learning implementations. In an MDP the current situation is described in a State, in our case the current game situation. An agent (or player) has the option to choose from a collection of Actions, which are based on a model, that is unknown to us, like the strategy of a soccer team. In an MDP the Actions are represented as a probability distribution. Every Action results in a change of the State and a value is assigned to taking an action, which is called a Reward. So every action has a result, represented in a new State and associated with that change is a Value, indicating how desirable that Action is towards achieving a certain goal. The process of a learning system is to find out the probability distribution of all actions and the calculation of the Values of all Actions.

1.3.2 The Bellman Equation

The Bellman equation is created to calculate the sum total of contributions of all actions that lead to a certain result. In many cases the result will only be known at the end of a series of events, for instance when scoring a goal. In order to find out what each step in a series of events has contributed to scoring the goal, the value of the end result needs to be distributed over all contributing events, preferably based on the importance of their impact. Most RL approaches make use of a modified form of the Bellman equation to assign values to the steps in a sequence of events. Sometimes this is partly done manually, in an ideal situation this process is automated. Most approaches are variations on this theme, sometimes with ingenious and complex solutions to overcome difficulties. In many cases the problem domain will have a large influence on the kind of problems that one may encounter.

1.3.3 On-policy and Off-policy learning

There are many different approaches to learn how to solve a problem. The most direct one is to learn the best Action for each State, called a Policy. Some approaches take a different view and learn to recognize and assign a value to State-Action pairs, which is the case with Q-Learning. The value of a State-Action pair is called it's Q-Value. Because this approach does not learn a Policy, it is called Off-Policy learning. In theory Off-Policy learning should not work, since the goal is to learn a Policy. However in practice it works very well and leads to very successful systems.

1.3.4 Model-based and Model-free learning

In order to find out which Actions are possible and allowed, two different approaches can be taken. In a Model-based system a data structure is provided, that represents all allowable actions. This can be done in the form of a table, a decision tree or a Monte-Carlo tree. Another way is to let the environment provide this information or use a neural network to learn these relationships. In such cases there is no model, but the environment is used to ask questions. For instance in Q-Learning, the system needs to explore the possible actions, given a certain state. The model may provide the probabilities of each action and the learning algorithm may decide to explore an unknown variation. It will then try this new action and may use a test environment to find out what may happen. The result is then added to the learned structure, which is slowly expanded in this way.

1.3.5 The Learning Algorithm

A Neural Network usually learns by finding the difference between the network prediction and the known output (the ground truth). This difference is called the loss factor and this factor is used in a feedback loop to update the weights in the hidden layers of the network. Some networks use a forward loop, but the principle remains the same. Different strategies have been developed on how this error is used in updating the weights and the main approaches are based on back propagation. The original algorithm for this was Steepest Descent, later followed by the popular Stochastic Gradient Descent or a more recent variation Restarted Stochastic Gradient.

1.3.6 Discrete actions and Continuous actions

When an agent needs to learn how to treat a certain situation, the learning system is to learn situation-action pairs. Each situation can lead to one or more actions, which are called policies. A policy can be a simple action like move forward, backward or stop. When the robot needs to move in different directions, we may need to increase the number of classes of such a movement, for instance with left and right. But if finer control is needed, the number of classes may grow to an unmanageable size. It does not seem reasonable for an agent to learn a separate action for every degree in

direction that the robot might move. For such situations a continuous action space is more appropriate. We would like to let the robot learn a parameterized function like `move(angle)`. Neural networks that are able to learn to integrate a regression with a classification are called parameterized action spaces. Such functions could also have multiple parameters, like in `move (angle, speed)`.

1.4. Motivation and Design Goals

During the past few years, participants in the RoboCup Middle Sized League (MSL) have attempted to develop a shared simulation environment in which all teams could play their software against that of other teams. This requires all teams to encapsulate the strategy part of their software in a standardized container and publish this information to be shared by other participants. This project has met several problems, mostly related to interfacing the different software approaches and tools to a single standardized environment.

In an attempt to develop a self-learning capability for our TURTLE robots, the idea grew to create a model that would be able to learn the existing behaviors from the detailed log files that our robots collect during all competitions. By studying the log files and designing a model to represent strategic decisions, we expect to learn a lot that would help in improving our current MSL software.

We also got in touch with a group of the Leiden Institute for Advanced Computer Studies (LIACS), that is analyzing the behavior of soccer players. Our detailed game logs form a rich source of data for their studies and we are interested to see if our model and their analysis methods would converge.

So as a prelude to a self-learning capability, we set out to develop a system that learns the behavior of all competing MSL teams and use this in our simulation environment, instead of canned strategic software from our competitors. We take our inspiration from ideas like the DeepMind Atari games [68] and AlphaZero approaches [89] and other approaches that might help in achieving our goal.

During the development of this system we use a simple simulation system in which we visualize the contents of the log files and the responses of the model to game situations. This environment is developed with PyGame and allows replay of the captured log files as well as letting two teams play, using the model that captured the behaviors of the various teams.

Figure 1: *Simulation environment to visualize the log file content and test the learned Neural Network against game situations.*

1.5. Backgrounds of our approach

Several attempts have been made in the past to build models of RoboCup Soccer. The main problem is to develop a representation for such a model, that is useful to interpret the followed strategies as well as for a later self-learning capability. While earlier approaches have concentrated on recording know repertoires of good policies, recent approaches seem to focus on zero-knowledge systems that are capable of finding out optimal policies by themselves. Although these approaches have reached impressive results, they suffer from a big credibility problem; their actions cannot be verified nor can they be explained and in some cases not even be understood.

In our case an explicit model is only useful if it helps explaining what an opponent does and has a representation that allows inspection and analysis. In Good-Old-Fashioned-AI (Gofai), systems were built based on knowledge and on symbolic representations, that were grounded in the real world. In order to be understandable and to explain their behavior, such systems need to be grounded on symbols and therefore we need to find a representation that facilitates this requirement. Many modern approaches attempt to let a system learn the distinguishing features of a domain, like for instance in recognizing objects in images. When analyzing positional data from a group of robots of two competing teams, the interpretation of such data cannot be learned autonomously. In many earlier approaches, hand-crafted features are used to speed up the learning process.

We propose to use a similar approach, where we rely mostly on hand-crafted features, but with this restriction that we want to keep the possibility open for

the system to discover additional features, if the hand-crafted features are not sufficiently discriminative. But where do we get such features from?

We started out with the already known features, used in Futsal, which is a 5 against 5 soccer-playing competition. We used the coaching manuals of FIFA and UEFA [28, 99], as well as the one from the Dutch National Futsal team, developed by Max Tjaden from KNVB [97]. These contain a rich collection of game situations, called set-pieces and documented in the form of small action diagrams, that we intend to use to classify game situations.

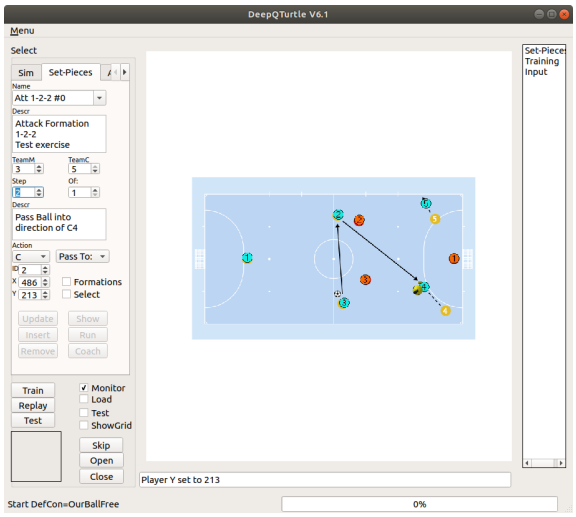

Figure 2: *Visual Editor for Set-Pieces. Each Set-Piece has a number of steps that describe the strategy to be followed, given a game situation*

The second source is formed by the way that our soccer-playing software is organized. In 2019 we implemented the STP framework [12], based on earlier work done by the CMUDragons SSL team. This approach consists of three parts; Skills, Tactics and Plays. The game situation, encoded in team player positions, ball position and possession is used to select a Tactic. From this an appropriate Play is selected. A Play or Set-Play (which is equivalent to the Futsal Set-Pieces) is selected and based on the skills of the robots, a role is assigned to each robot. The STP framework consists of a declarative structure in which these Skills, Tactics and Plays are defined and will serve as the grounding symbols, or features in our approach.

That has an implicit risk, that other teams may operate with different interpretations of the game situation and utilize features or grounding symbols that differ from ours. This may result in interpretation differences. but as a first approach it should be a good attempt, interpreting opponent actions in terms of our own model of the game. When we encounter misfits, we can always adapt our underlying STP model. The main task of the learning system is then to interpret the game situation and robot actions in terms of the underlying Set-Pieces and STP model.

In order to do so, we will build a hierarchical structure, that will learn to interpret the game log entries in three layers as follows:

1. Classification of the Game Situation
2. Identifying individual Agent Actions
3. Identifying actions of the Agent with the Ball

1.6. Classification of the Game Situation

This is the first part of the system that we will implement. Its main goal is to interpret each frame from the logfile and determine the game situation. It will be translated to a game situation per agent, for deeper analysis in the next phases. It should result in information that leads to the same conclusion as the selected Tactic in the STP framework. We will first develop this for our own team, since we have all data about the actual decisions and actions that were taken by our robots. This allows us to test if the analysis is correct. If so, we can apply the same approach to the opponents, for which we do not have this information and thus learn the Game Situation and Tactics for our opponent. This is only a first approximation. Because most of our opponents are not using STP, but even if they do, we have no access to their implementation of the framework, we need to refine this later on. We will describe this approach in section 3 and following.

The process of classifying the current game situation is the task of generating a structure, similar to the Set-Pieces that we retrieved from the Futsal Coaching Manuals and then match these structures with the corresponding STP structures. Because a Set-Piece structure consists of a map with the positions of all players and a number of steps, this process requires a complex matching of time-steps. The STP version of the Set-Pieces serves as a model. Once this model is created, a matching STP model is searched for. If this cannot be found, this is an indication that we encountered a novel situation and the STP model for the given team needs to be expanded with the learned information. This way an entire STP model is learned that can be inspected with the Visual Editor and serves as an explanation of the team strategy.

A visual editor is developed that allows the definition of known Futsal set-pieces. So when a game situation is presented to the network it should respond with

a reference to the most appropriate set-piece, showing the Tactics that are known for this game situation. This should provide the context for the neural network to interpret the possible roles and actions of all agents in the set-piece.

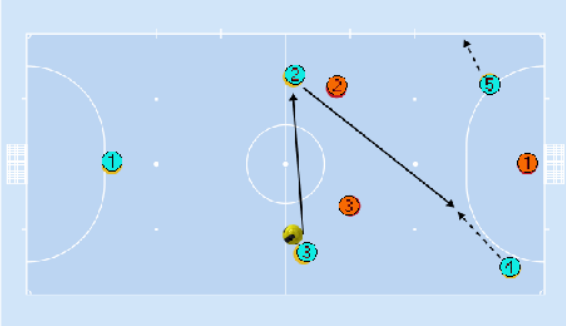

Figure 3: *Set-Piece for Attack in 1-2-2 Formation.*

An important part of classifying a game situation is the position of individual players in the field. A good way to identify the configuration of player positions is the use of a Formation. In Soccer as well as in Futsal a number of well-known formations are used, like the most popular 1-2-2 Formation.

1.7. Identifying individual Agent Actions

When interpreting the movements of individual agents, we need to find the begin and end of an episode, which usually coincides with the transition of ball-possession. To identify Agent Actions, we take an entire episode and isolate the movements of each agent of a team. First we begin with the starting positions of all agents in the field. Then we take the sequence of actions of each agent and analyze these movement patterns. We need to match them with known movement patterns.

Agent actions are simple in principle. We divide the playing field into a grid of 50x50 cm tiles. Each tile can occupy a robot or the ball. Because the log files are recorded at a frequency of about 10 Hz, each cycle will at most record movement of a robot of a single tile. So we can learn a classification of movement in 8 different directional classes per time step.

The problem however is to analyze what these movements mean. We need to interpret a time-series of movements and from it infer the intention of the agent. This analysis is similar to Human Action Recognition, which is a widely studied subject in Vision-Based systems. Therefore we have studied the methods that have been developed in this research area and apply these to our domain. Luckily our agent's movements are simple 2D movements in a series of Game Situation Images. We classify the movements based on a set of pre-defined features, for which the Set-Pieces and the STP formalism form the basis. So there is a starting Game Situation Image (GSI) like we have seen in the Coaching Manuals. Each path that an agent has followed then becomes a step in the Set-Piece.

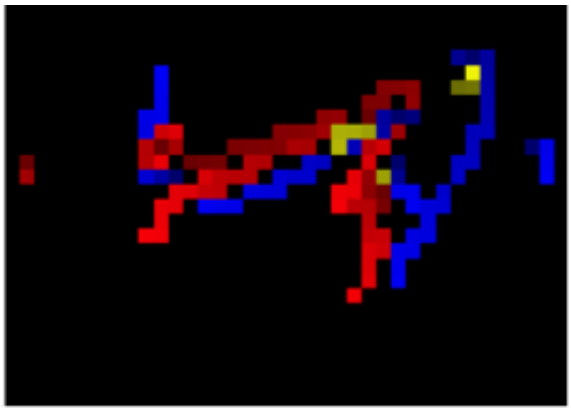

Figure 4: *Game Sitution Image with traces for all agents and the ball. The episode will be translated into an image similar to figure 3 in which the traces are converted to lines that show the movements of team members and the ball.*

The Game Situation Image will be transformed into one in which the starting position of each agent is depicted with a number, as in the Set-Pieces. The steps in the episode are then inferred from the movements and added to the generated Set-Piece. This process transforms the entire episode into a complete set-piece, which we then need to match with all known Set-Pieces in the library. When the closest Set-Piece is found, that one serves as an explanation of what happens during the analyzed episode. If no matching Set-Piece is found, we either need to add it to the library or need to modify the classification of this game situation.

1.8. Identifying actions of the Agent with the Ball

Interpreting actions of the Agent with the Ball is much more complicated. First of all, here we are in most cases not dealing with a series of actions, because a single action with the ball, will result in a number of ball movement events. So in this case we need to infer from the Ball movements in successive time steps, what the intention of the Agent was, at the moment that the ball was kicked. The same applies to actions like dribbling, passing or shooting at the goal. Here

a combination of Agent movements, along with Ball movements must be an indication of the Agent's intent. Other than in case of a simple Agent, here movements cannot be discrete-sized, because they happen in the continuous domain, so we need to learn regressions in a continuous parameterized action space. A different type of network is required for such operations.

2. Related Work on Opponent Modeling

In this paper we describe many existing approaches to learning in an attempt to find the ones, best suited to our goal.

Opponent Modeling has been the subject of much research, dating as far back as the nineties. These studies are mostly executed in the domain of Multi Agent Systems (MAS) and are often also related to learning agent strategies. Two domains have dominated this research; the Pursuit environment and Robotic Soccer. Soccer is a rich area to study multiple cooperating and competing agents in a simplified world [94]. We have seen many different approaches in this domain with many formalisms. In our work we concentrate on Robotic Soccer Games and have observed a number of modeling approaches, described in the following sections. In the survey of MAS of Stone and Veloso (2000) [94], a systematic overview is given of different types of Multi Agent Systems. We concentrate on Heterogeneous Communicating and Non-Communicating agents. Another overview describes the efforts on Opponent Modeling in the RoboCup Coach Competitions (Pourmehr2010 [76]), which were held from 2001 through 2006. This sub-league of the Simulation League was specifically aimed at finding ways to analyze the opponent behavior and allow the coach software to formulate strategic advice to the team. A more recent summary of work on Opponent Modeling in relation with more modern Deep Learning approaches is given in the work of Hernandez-Leal2018 [42].

2.1. Policy Models

. *Statistics*
. *Situation/Action pairs*
. *Policy/Value*
. *Monte Carlo Trees*

Actions that agents take, given certain game situations are expressed in policies, recorded in Policy Models. The first models were mostly based on statistics, like the positions of the agents, distance to the ball, distance to the opponent's goal like the M* algorithm (Carmel2000 - [13]). More detailed information is recorded in Situation/Action pairs, where associations are detected between game situations and actions that prove successful. In learning environments, this resulted in systems that would learn to assign a value to a certain policy, related to a game situation in Steffens2004 - [92].

What is recorded in a model, mainly depends on the domain and the situation. Because the number of possible situations is often intractable, many approaches simplify the world, using hand-crafted features. This generally is a lot of work and the performance of such a system largely depends on the quality of the chosen features. Therefore the real problem is to let a learning system find relevant features automatically. In general, discovery of features is speeded-up by simplifying the environment, either by reducing the granularity of the playing field or by restricting the classification space.

In later studies policies were recorded as Policy/Value pairs, where Reinforcement Learning (RL) and later Q-Learning were used to learn the best scoring policies. Selene1997, Ledezma2002 and Iglesias2009 [88, 54, 45]. In other work, Monte Carlo Trees are used to represent opponent actions Hausknecht2015 [40].

2.2. Model Features

. *Hand-Crafted Features*
. *Team Strategy Classification*
. *Action Patterns*
. *Commentary*

Hand-crafted features are used to classify strategies, agent's roles or actions taken by agents. In general learning team strategies works better than learning individual agent strategies. Different methods have been proposed, but in most cases we see models that consist of three layers, as described in the next subsection. Different strategies are described in Pourmehr2010 [76], like Team Strategy Classification, Agent Action Classification, Formation Analysis, Offense/Defense Strategy and Pass Graphs. It ranges from recognizing simple behaviors like Intercept, pass, dribble, where the special Viena Pattern Editor was used, and a language (CLang) to describe the action patterns that are being used. (Iglesias2006 [46]). More complicated are so-called Trie structures (Iglesias2009 [45]), where sequences of actions are used to model agent behaviors (`pass1to2->dribble2->pass2to10->goal10`). Case Based Reasoning is also used in some cases (Steffens2004 [92]).

2.3. Three Level Layered Models

. *Strategy/Formation*
. *Belief/Desire/Intention (BDI)*
. *Play/Role/Action*
. *Skills/Tactics/Plays (STP)*

Analyzing what a team does as a whole is recorded in team strategy, where team formation is often used to define cooperation patterns like 1-2-2 (the most popular Futsal formation) as used in Strategy/Formation approach. (Pourmehr2010 [76]). In the lowest level the individual agent actions are then placed within this framework.

Another approach is to determine a team's goals, based on the beliefs of individual team members about their intentions. Like the team wants to move to the opponent half, the intention of player X is to pass the ball to player Y. This is done in the Belief/Desire/Intention (BDI) approach, described in (Haddadi1995 [**?**]). Taking inspiration from human soccer teams, game plays are often used as examples in which each player assumes a role that results in actions, executing that role, as used in Play/Role/Action and Skills/Tactics/Plays (STP) Browning2004 [12].

A ground-breaking development was Dynamite, which was used to develop Reactive Deliberation which was based on a number of hard-wired behaviors (Sahota1994 [82]). It served as the basis and inspired much of Stone's work on robotic soccer. Minimax-Q Learning for Markov games was used in a simulated soccer game (Littman1994 [61]) and formed the basis for the later Soccer Simulation Server (Stone and Veloso1996 [107]) . This server was also used in a commentator system ROCCO (Andre1998 [5]) and MIKE (Matsubara1999 [66]) and Byrne (Birnsted1999 [10]).

Gradually more work was done on decomposing soccer tasks into different roles. A system with dynamically changing roles was developed by (Balch1998 [8]) later followed by a RL approach, where an entire team learned effective behaviors instead of individual behaviors. Hierarchical task decomposition became the accepted way to let entire teams learn collective behaviors (Uther and Veloso1997 [108]).

While these models are mostly used to analyze the game situation, a similar approach is often used to control a team of agents, like in the STP approach of the CMDragons SSL team (Browning2004 [12]) or a similar method used by the Cambada MSL team (Koning2012 [20]). These approaches can be viewed as both an analysis and synthesis of the same problem. The CMDragons system also used a more statistical approach in a Feature Based Opponent Strategy Model, in which distance to the ball, distance to goal and distance between robots was used to determine the game situation. (Trevizan2010 - [98]). It is an interesting observation that many of these approaches are using three-level structures to describe the tactics of teams, although they use different ideas and formalisms to achieve the same goal.

A statistical approach is taken by the Leiden Institute of Advanced Computer Studies (LIACS Meerhoff2019 [67])

Opponent Modeling is mostly used as part of a learning system. Almost all systems use an implicit opponent model to improve the learning skills of a team of agents, using the model to predict the possible counter-actions of the opponent.

2.4. Action Recognition

All these approaches use instances of a form of Action Recognition, which has been the subject of study for the past 10 years and has seen many new ideas. In the comprehensive review of recent developments of Zhang2019 [121] most of the recent ideas are described. Another good overview of recent developments is given in (Ghosh2018 [34]), where the most influential contributions are listed. However most of these recent developments concentrate on Human Action Recognition, in which Depth analysis and Skeleton analysis play a growing role. In our case a much simpler approach is required, in which only the trajectories of agents and the ball play a role.

A study in which people movements in a home setting were registered with simple on/off switches on doors and cupboards (Kasteren2008 [102]) is much more representative of our type of problem.

In this study both a Hidden Markov Model (HMM [26, 33]) and Conditional Random Fields (CRF) [25, 112] are used and perform equally well. The main difference is that CRFs are more flexible and allow more complex dependencies to be modeled. It can be assumed that in many cases, the behavior of an agent not only depends on the previous state, but also on that of surrounding agents and the position of the ball. So we may encounter situations that require the implementation of a CRF as part of a Reinforcement Learning environment, or similar approaches.

2.5. The Intentional Stance

Because we are looking for ways to offer explanations of the learned behavior, it is important that the elements of the model are symbols that are grounded in the robotic soccer-playing world. Important work on Symbol Grounding has been done by Steels2005 [91] and Harnad1993 [38].

Attributing beliefs and intentions to agents like robots is taking the Intentional Stance, which allows

us to understand and explain the behavior of such systems as developed by Dennet1971 [23]. Building a model that includes a robot's beliefs and intentions creates a semiotic network [91], that forms the basis for communication between the robot and the user, grounded in the soccer-world, and provides the basis for explanatory capabilities of a system.

Therefore we need to classify sequences of actions into known behaviors like `Mark`, `Block-Passage`, `Pass-Ball` and determine for every agent the intentions that agent has. In the soccer-playing domain intentions have been investigate in the work of Rabinowitz2018 [**?**] and the BDI approach Haddadi 1995 [**?**].

We will be using beliefs and intentions as described in the STP implementation of our software [12, 20] and also in the Set-Pieces of the Futsal Coaching Manuals of FIFA, UEFA and KNVB [28, 99, 97].

2.6. Reinforcement Learning

In the informative survey of Hernandez-Leal2018 - [42] most of the recent RL approaches are described, in which we see a gradual development from simple Reinforcement Learning (RL) to many different approaches of Q-Learning:

RL Methods	
ID	Descr
DQN	Deep Q-Learning Networks
BDQN	Bootstrapped DQN
DRQN	Deep Recurrent Q-Learning Networks
DRON	Deep Reinforcement Opponent Networks
A3C	Actor-Critic systems A2C/A3C
Continuous	
DDPG	Deep Deterministic Policy Gradients
NAF	Normalized Advantage Functions
PGN	Policy Gradient Networks
PPO	Proximal Policy Optimization

Table 1: *Various Reinforcement Learning Methods.*

Over the past years we have seen a gradual shift in learning, starting with Reinforcement Learning (RL), growing into Q-Learning and Deep Q-Learning Networks (DQN) Mnih2013 [68]. In a DQN a separate network estimates the Q-Values, that are used to learn the Policy Values of the Target network. In such a network the Policy consists of Actions, that are selected based on the best long-term reward for a series of actions. This approach was also taken in a small Soccer simulation program with implicit Opponent Modeling, playing hand-crafted strategies with 2 players on a 6x9 grid to find out if the system could learn a better strategy by itself (He2016 - [41]). An extension to this approach is bootstrapped DQN [74], which learns faster, because it uses deep exploration and achieves better results. In another study, a simple 2-player simulation was created to learn to recognize action patterns (Martinovic2010 - [65]). This system used three occupancy matrices to represent the Ball, Own team and Opponent. There were two representations of the playing field; a Strategy Level Model (coarse) and and Abstract Level (fine). The compressed representation of the field allowed patterns to be more similar, while Pattern Matching was based on a statistical approach.

Where the earlier approaches used stacked input frames to represent successive actions, later approaches included a Recurrent Neural Network (RNN) Liang2015 [56] in the form of a Long Short Term Memory cell (LSTM) [49, 73] that can learn to represent longer sequences or sometimes even multiple sequences with different time scales. This type of network uses an Experience Replay (ER) [85] buffer, from which the learning system randomly takes sequences of fixed length, to gradually learn the best policies.

This approach has been extended into Deep Deterministic Policy Gradients (DDPG) Lillicran2016 - [60], where the discrete actions are replaced by a Policy Gradient. Gradients allow the network to learn gradual functions like kicking in a certain direction or with a given force. This work was further extended in a system that learned fully parameterized action spaces that are similar to program functions `Dash(power, direction)`, `Turn(direction)`, `Kick(power, direction)` (Hausknecht2016 - [39]). In this work hand-crafted intermediate rewards are used to overcome the problem of sparse rewards. Also bounds checking for the parameterized spaces is done by inverting the gradients when they extend beyond the acceptable values. In a successive study of this approach, a hybrid between DQN and DDPG was created, which directly learns actions in one network and parameters in a second network (Xiong2018 - [117]). More work was also done on networks that learn Continuous Action Spaces, like Normalized Advantage Functions (NAF [35]), which is a much simpler approach than DDPG. A similar approach is taken in the work on Proximal Policy Optimization (PPO) [87].

More recent approaches use separate networks as critics to improve the accuracy of the learned values in Actor Critic Systems (A3C) [50, 69]).

Different methods are used to store these models,

where Monte-Carlo Trees and other tree structures are most popular. A more explicit model of opponent behaviors was used in a system that used Opponent-driven planning, in which opponents were coerced into actions that would allow easier attack by the playing team. Strategies in this system were called `Pass-Ahead` and `Coerce-and-Attack` and is known as Threat-based defense (Biswas2014 - [11]).

Several new approaches have been developed like Agents Modeling Agents (AMA Albrecht2018 [4]). Learning with Opponent Awareness (LOLA Foerster2018 [29]) and Theory Of Mind Networks (ToMnet Rabinowitz2018 [**?**]) where the intentions of the opponent are being analyzed. As these networks grow larger, parts of the network are being shared by different versions, using so-called Parameter Sharing (Sunehag2018 [95]).

More recent developments use the Opponent Models to explore the possibilities with Self-play, like in Alpha-Go (Silver2016 [**?**]), Alpha-Zero (Silver2017 [**?**]) and Alpha-Star (Arulkumaran2019 [6]). Another approach is the use of a specialized Deep Reinforcement Opponent Network (DRON) He2016 [41].

2.7. Recurrent Networks

Most of the approaches to learn to recognize action sequences are based on Recurrent Neural Networks (RNN), using one or more Long Short Term Memory (LSTM) layers ()Karpathy2015 and Olah2015 [49, 73]). Recent developments in this area have shown that when the number of steps between the beginning and end of a series is large, the memory and processing requirements become large. Also when there are dependencies that are not directly related to earlier steps in the sequence, these systems are unable to learn these relationships. Conditional Random Fields (Echen2012 and Wallach2004 [25, 112]) are one way to solve this problem, but recent developments offer a better alternative in the form of Attention Mechanisms (Culurciello2018 [19]).

We see an increased use of combinations of Reinforcement Learning, supplemented with Recurrent Networks, based on the Attention Mechanism [31], but Attention Mechanisms offer many more advantages. First of all they allow richer representations of the dependencies between elements of a series of events. In Hierarchical Attention Networks (Yang2008 [119]) several layers like in our case the Game Situation and Agent Actions can be represented by nested networks. And although most of these networks are used in Natural Language Processing (NLP) tasks, processing of time-series is a good topic to be handled by this technology (Vinayavekhin1806 [110]).

One of the biggest problems with Neural Networks, however is that it is hard to understand what they are doing. Finding ways to explain their behavior is very important and Attention Mechanisms offer a good way to tackle this problem (Liao2018 [57]). An important aspect of being able to explain why a system is selecting a certain action is to work with symbolic information. Actions and Objects need to be grounded in the real world. This is referred to as the Grounding Problem [91]. In the early days of AI (GOFAI) [55] most processing was symbolic, based on reasoning with heuristics, that were derived from human experts [21, 103].

Being able to combine the sub-symbolic nature of Neural Networks and grounded symbolic information in a single system will allow for better facilities to provide explanation facilities, like were available in Expert Systems [103]. Several proposals have been made to allow this approach [83, 38]. In our case these facilities could be used together with the Set-Pieces as part of a Hierarchical Attention Network, combined with GRUs to form an explainable model of the behavior of opponents.

2.8. Previous Work

Earlier work on predicting opponent behavior was done by one of our team members, van 't Klooster (2018) [105]. In this work, traces of game situations were created in occupancy grids, representing the movements of opponents in various game situations. A Convolutional Neural Network was used to learn to recognize these traces and predict the action of an opponent. (See Figure 5)

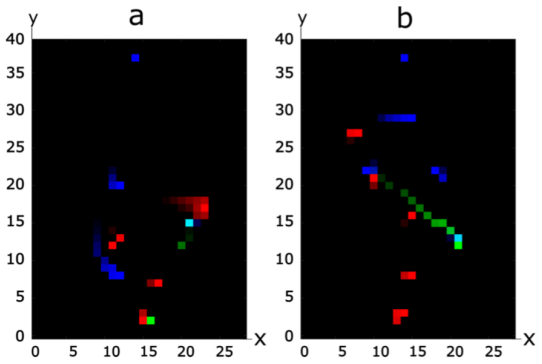

Figure 5: *Example of a trace of two game situations: a) a Pass, b) a Shot at Goal. Represented in 15 and 12 time steps*

Learning to play soccer is the subject of many earlier studies, that inspired our work. The German Brainstormers Tribot Team [37] was one of the first to try such an approach. [80, 81]. In the RoboCup Simulation league several approaches have been tried [93]. Several teams from other leagues also have made attempts at this. [86, 18, 14, 120, 52, 37]

2.9. Contribution of this Work

Here we will list our contribution. The list underneath is a first attempt.

1. Explicit Opponent Modeling as an analysis method
2. Team Formation Analysis
3. Agent Action Prediction and Intention analysis
4. Team play selection and classification
5. Using STP as an analysis and synthesis model
6. Using Set-Pieces as a visual model to explain strategies

3. Our Opponent Modeling Approach

Our goal is to create an explicit Opponent Model that can be used to drive a simulator and allows comparing the various team strategies. Analyzing a game is done on three different levels:

1. *Team Level* - Classifying Game Situations
2. *Tactical Level* - Identifying Formations and Set-Pieces
3. *Agent Level* - Recognizing Action Patterns, Roles and Intentions

3.1. Classifying Game Situations

Here we attempt to define the current game status. It is generally done by defining features like Attack and Defend and information about the location where the game is concentrated like OurHalf and OpponentHalf. In some cases more detailed information is required by dividing the field into a number of sections.

3.2. Identifying Formations and Set-Pieces

Based on the Game Situation there may be more detailed qualifications about the way the game situation is handled. Set-Pieces are standard patterns that are used in pre-defined situations and Opponent Modeling approaches attempt to identify which of a number of given Set-Pieces is being used. Recognizing a formation is used to identify which roles have been assigned to the players based on their relative positions.

Each team will have its own Model, consisting of a collection of Set-Pieces. These will be shared with a generic model, so we can find out which strategies are shared by different teams. During analysis, first a template Set-Piece is generated and then matched against the database. If the strategy is found, it is added as a reference. If not, the new strategy is added to the database. Also we will keep a collection of verified Set-Pieces as a central reference, that we can use as a basis for comparisons of all encountered strategies.

3.3. Recognizing Action Patterns and Roles

On the level of individual players, the Opponent Model tries to recognize a certain pattern, expressed as a role like Attacker, Defender, Assist. In other cases sequences of actions are strung together as patterns like Dribble, Pass, Receive, Shoot. In order to determine an agent's intentions, we take the entire episode and trace the movements of an agent. From these movements we intend to learn what these movements mean, just as was done in the study of people movements in a house (Kasteren [102]) in which a Conditional Random Fields approach was taken. We will investigate if we can combine this approach with a Hierarchical Attention Network (HAN).

3.4. The contents of our logfiles

The logfiles that our TURTLE robots create during competitions, contain details about the position of the robots of both teams and the ball in time-steps of about 10 milliseconds. In addition this file contains information about the game situation, like ball possession and the starting condition of fragments, like kick-off, throw-in etc. We divide these traces into groups that start and end with a referee command, communicated to the robots using a so-called RefBox computer. Each of these game moments is split into episodes, spanning the period between switching of ball-possession, called Game Turnover Points (GTP). Episodes can last from 1 time-step to hundreds, spanning periods of 1 to 10 seconds or more. More details can be found in Section 4, also see Figures 10 and 11

The input to the Neural Network, that we will be building is formed by Game Situation Images (GSI), generated from the TURTLE log files. This image contains information about the positions of all robots in the field and the position of the ball in 40x28 pixel RGB images, where R represents our team, B the opponent team and G the ball.

In addition to this, a visual representation will be made of the starting situation, similar to the images, used in the Set-Pieces (see Figure 3)

The episodes in the log files can be divided into three different types:

1. *RefBox Situations* - Start- and endpoint of an episode
2. *Game Start Situations* - Start after a RefBox Start event
3. *Game Turnover Points* - Transfer of ball possession

Initially we will only use the last type of events, the Game Turnover Points. The reason for this is that for Refbox Situations all teams use standard Set-Pieces, that we will learn at a later stage, because they only involve preparing for a game-start situation like moving to their positions for a kick-off. These have no bearing on the strategy of the team during a competition. The actual Game Start Situation is generally controlled by Set-Pieces as well and describe how to handle a Kick-Off or a Throw-In. We will

concentrate on these at a later stage, since they have only a minor influence on the overall strategy.

So the situations that we will concentrate on are those directly following the first Game Turnover Point after a RefBox situation. This is where a team first loses or gains possession of the ball. At every Game Turnover Point a new episode starts and is treated as the start of a Set-Piece. A game situation may have several alternative set-pieces and the network needs to learn the probability distribution of choosing one of these alternatives. The traced patterns of agent movements are then represented as a step in the Set-Piece.

3.5. Treating Soccer as a Classical Game

Most recent approaches to learn to play games, like the Atari Games, AlphaGo and Alpha-zero ([**?**, **?**, 89]) are based on so-called Classical Games. In a Classical Game two players take turns in moving pieces on a board. So if we want to use the same technology that has been so successful in recent years, we need to answer the question if Soccer can somehow be interpreted as a Classical Game. To answer this question we introduce a number of simplifications.

First we convert the playing field from a continuous space into an occupancy grid of 50x50 cm cells, making it similar to a playing board. Each cell can only contain a single robot or the ball. Although in reality the ball could partly overlap with the robot, we allow the ball to fully overlap with a robot or to directly connect to its grid cell. A robot possesses the ball if it overlaps or connects with it. Secondly we replace the continuous actions of robots into time slots of 10 ms and allow each team to move their robots a single cell during each cycle, thereby responding to movements from the opposing team. This results in a turn-based system where each team may move their robots once every step. The ball may be moved over more cells, representing the higher velocity of the ball, but only by the player that possesses the ball.

3.6. Turn Based Soccer

Although there is no scientific literature about turn-based soccer we found three computer games in which Turn Based Soccer has been implemented [79, 51]. Another one (GridSoccer) is using Reinforcement Learning to play a game on a 20x30 cell grid [47]. This begs the question: "Is Turn Based Soccer similar enough to Continuous Soccer to serve as a basis to describe a real soccer match?"

More importantly however is that we want to use this approach to control game strategy. This is currently done based on assigning roles to the robots that, once assigned are executed for as long as possible until the game situation deviates too much from the situation that led to the current role assignment. In such a case new roles are assigned. In a turn-based system every new step will re-assess the current situation and suggest player moves to adapt to the new situation. Therefore the network needs to receive traces of a game that allows it to learn to anticipate desirable or undesirable situations and associate these with actions that the team usually makes under these circumstances. By using a recurrent network we expect such a network to learn to associate time-separated game situations so that some kind of look-ahead behavior will occur. We need to find out what the best trace length is for this to happen and will test trace lengths between 4 and 10 steps, spanning a time period of about 1 second.

In earlier work on Opponent Action Predicting (Klooster2018 [105]) the game state was represented as an occupancy grid with stacked input frames, that were analyzed by a Convolutional Neural Network (CNN). In our approach we need to find an agent's intentions, for which we will be using Conditional Random Fields, possibly combined with a Recurrent Neural Network or a Hierarchical Attention Network.

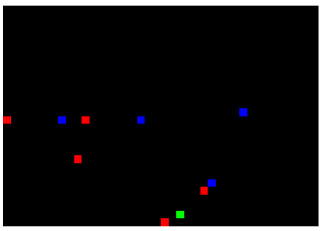

Figure 6: *A Game Situation Image shows all robots of our team in Red, opponents in Blue and the ball in Green.*

3.7. Objectives of the Neural Networks

We will be creating a set of Neural Networks that takes an episode as input, consisting of all steps that lead to a Game Turnover Point and that outputs three pieces of information, relevant to a simulation program:

1. Agent Actions
2. Ball Actions
3. Game Situation Classification

Figure 7 shows the relationships between these inputs and outputs. The input consists of a GSI and

Aspects of Game Log files			
Aspect	Loss%	Error%	Remark
Agent Action	0.0	0.0	
Ball Action	0.0	0.0	
Team Advantage	0.0	0.0	
Partial Observability	0.0	0.0	
World Model	0.0	0.0	
4 Steps	0.0	0.0	
10 Steps	0.0	0.0	
N Steps	0.0	0.0	

Table 2: *Overview of the different aspects that we will investigate.*

the network predicts the Tactics that will be selected as well as the actions, taken by the players of the team. Actions are split between the active player, that possesses the ball and the other players that assist or defend. In terms of STP, the network will classify the Set-Piece, identify the best Tactics and predict the actions for each player.

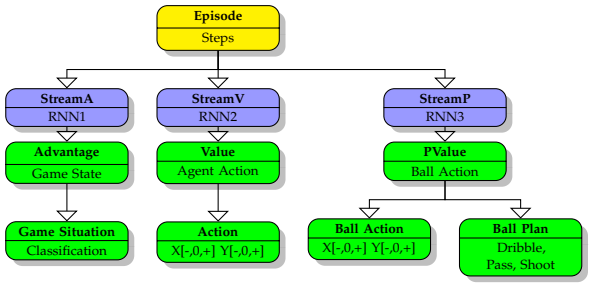

Figure 7: *Learning Goals of the Neural Network*

The model learns the actions for all agents, so when playing a simulation, every agent consults the network to find out the strategy when playing in that position, given the current state of the world model.

We will include an additional network that concentrates on learning the Advantage of the current Game State, given the learned team strategy in order to find weak and strong points of the team strategy. During a simulation this could warn for dangerous or advantageous situations. Table 2 lists the possibilities and the results that will be gathered during the research. When we have collected the information, we will rewrite this section to represent the results.

3.8. Our Neural Network Approach

As explained in the previous sections, there is a large body of research, related to learning. In our case we are looking for the best method to build an Opponent Model for each MSL team and associate these with a reference model in order to make comparisons. Because each year teams improve their strategies, we need to make a new model for each team for every year.

Most learning systems are based on some form of a Reinforcement Learning approach, where Recurrent Networks are employed to learn time sequences. Because we also want to be able to offer explanations, or current thinking is to use a Hierarchical Attention Network (HAN Figure 8), combined with Gated Recurrent Units (GRU) and probably a Convolutional Neural Network (CNN). This type of network may possibly be combined with a reinforcement learning network like Double Dueling Q-Learning Recurrent Network (DDRQN) and for continuous action spaces with a Normalized Advantage Function Network (NAF).

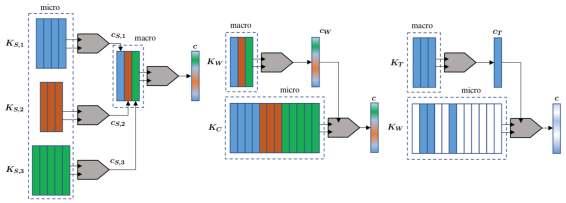

Figure 8: *Hierarchical Attention Network Model (HAN).*

Phases of the project				
Nr	Descr	Status%	Result	Remark
1	Converting Logfiles	100		
2	Classifying Game State	0		
3	Generate Set-Pieces	0		
4	Learning Agent Actions	5		
5	Learning Ball Actions	0		
6	Combining Models	0		
7	Learn Advantage/Threat	0		
8	Running in Simulation	10		
9	Testing with Set-Pieces	5		
10	Packaging in module	0		
11	Testing in Greenfield	0		
12	Publication	5		

Table 3: *Phases of the project.*

3.9. Project Planning and Status

In table 3 we list the project phases and will include updates for the status of the project. The first step will be the classification of Game States, for which all preparations are now made and training of the first network can begin.

We explain our approach in the following sections:

4. Analyzing The MSL Game Log files
   - Determining Game Turnover Points
   - Classifying Game Situations
   - Treating Soccer as a Classical Game
5. Training the Neural Network
6. Using the network in a Simulator
7. Conclusions and further work

4. Analyzing the MSL Game Log files

The Tech United TURTLE robots keep logfiles of all played competitions in digitized form. We use these logfiles to determine the game situation for every step of a match and divide these game situations into three different categories:

1. RefBox Situations
2. Game Restart Situations
3. Game Turnover Points

4.1. RefBox Situations

During MSL Robot Soccer competitions, a human referee judges the behavior of the participating robots and calls for a stop when a violation of the rules is detected or a goal is scored. A single computer is dedicated to communicate these decisions to the robots in a so-called RefBox command. These commands are situations like a kick-off, a throw-in or a corner.

When a RefBox situation occurs, the game is stopped and the event is captured in the logfile. The robots then prepare for the given situation and take their positions. For instance when a throw-in is called, the referee first places the ball and then the robots regroup to resume play given the throw-in situation. The robot closest to the ball positions itself behind the ball, a second robot gets in a position to receive the ball. The team of the opponent takes positions at a distance from the ball as described in the rules. The logfiles describe the movements of all players to reach their starting positions. These, however are not the robot actions that determine game situations and are therefore not used to learn team behaviors.

4.2. Game Restart Situations

When both teams are positioned, the referee calls for a Start to resume the game (See Figure 10). The team that is in possession of the ball tries to keep the ball and move to the opponent's goal in order to score a goal. The defending team tries to block these attacks and tries to gain possession of the ball. As soon as the attacking team loses the ball, or shoots at the goal, the game situation changes (see Figure 9 and 11). These situations are called Game Turnover Points (GTP).

4.3. Game Turnover Points

While the two situations, described before; the RefBox and Restart situations, are mostly based on so-called set-pieces, game Turnover Points (GTP) are the most important game situations that determine how well a team plays the game. Learning team behavior is therefore mostly concentrated on these GTPs and we want to learn which actions a team performs to get and maintain possession of the ball. In addition we want to learn how a team avoids losing the ball or prevents the opponents from seizing the ball. Additional situations are dribbling, passes, receives and shots at the goal.

During a match, the main aim is to move the ball into the opponent's goal and there are a number of steps, that can be identified to classify how well this aim is being achieved. In Figure 14 we show the various state transitions that may occur between GTPs.

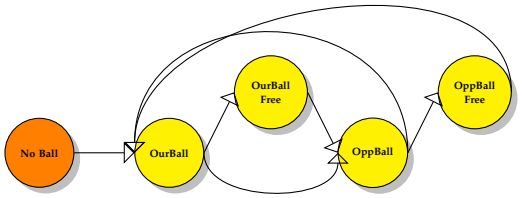

Figure 9: *Ball Possession State Transition Diagram. NoBall means that ball is not seen. BallFree means that the ball is in between players*

4.4. Input for the Neural Network

The TURTLE logfiles are loaded into two selection grids, in which the game situations are listed. A manual selection can be made to inspect the contents of a file.

The basic data structure is a collection of steps, where each step is identified by a step number, followed by information about the RefBox situation and information about ball possession. It also contains location information for the ball and the players of both teams.

4.5. Multi Agent behavior

The network learns the behavior of each role by taking the positions of all players of a team and trace the actions of all of them. This is done, because all field robots have the same skills. The keeper robot has some special skills and is not included in this learning process. In our current system, all robots operate on a shared world model, which is built by combining the position information from all robots. We can train the network from this world model represented in a Game Situation Image (See Figure 12 and 12).

	RefBox	Start	Sto
1	ThrowIn Magenta	0	12
2	Start	12	134
3	Stop	134	386
4	GoalKick Cyan	386	516
5	Start	516	592
6	Stop	592	677
7	Dropped Ball	677	696
8	Stop	696	982
9	FreeKick Magenta	982	106
10	Start	1066	121
11	Stop	1211	143
12	KickOff Cyan	1434	156
13	Start	1566	176
14	Stop	1762	196
15	KickOff Cyan	1962	209

Figure 10: *List of RefBox situations from a log file*

	DefCon	Start	Stop	Steps
1	OurBallFree	12	96	84
2	OppBall	96	98	2
3	OurBall	98	99	1
4	OppBallFree	99	105	6
5	OurBall	105	110	5
6	OppBallFree	110	112	2
7	OurBallFree	112	115	3
8	OppBallFree	115	119	4
9	OurBallFree	119	121	2
10	OppBallFree	121	133	12

Figure 11: *List of Game Turnover Point episodes, selected from a RefBox situation, consisting of time-steps*

Although we have detailed information about the position of all robots as well as their orientation and acceleration data, we convert this information to a grid as input to the Neural Networks. By doing so, we lose some information and therefore the question is if there is not a more natural way to represent trajectories. There exist a class of Neural Networks, called Graph Neural Networks, that represent different kinds of graphs (Wu2019 [116]). In these networks however the graph representations are also first converted to a grid-like representation. So currently there is no Neural Network type that can handle trajectories other than in grid-like representations. Recognizing trajectories and action patterns are more a vision problem than a sequence classification problem (Singh2016 [90]).

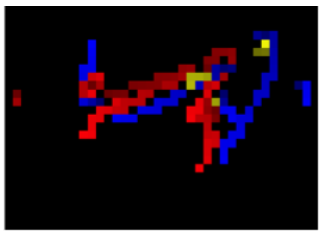

Figure 12: *Our development environment visualizes traces of an episode in deminishing colors to give an overview of what happens during an episode*

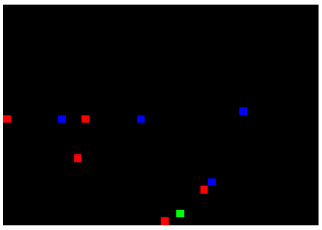

Figure 13: *The positions of all robots are shown in a grid for each time stpe of an episode. Red is our team. Blue is the opponent team and Green is the ball.*

4.6. Learning vs Capturing

In the next sections we describe the approach we take to build a learning system. The most important decision is the reward system, that tells the network how good it is currently performing. In a self-learning system we would, for instance give a reward if a team seizes or keeps the ball and give a negative reward when it loses the ball. When we want to represent the behavior of a team, without improving on it, the reward should be given when the network correctly predicts the next action and a punishment if the predicted action is wrong.

So we need to concentrate on establishing when an action is correctly predicted, where it becomes important to also distinguish between grades of correctness. Given the fact that our logging data is captured at a frequency of 10Hz, there can be small fluctuations between successive frames, because they will be quantized. Given the maximum speed of the robots, each frame would normally show a movement of at most a single cell. Because the system representation does not allow movements of a fraction of a cell, there could be differences between predictions of a single cell. So partial movements could be correct and still not match the output of the system. Therefore moving in the wrong direction should be punished, while moving in the right direction should be awarded, but a failure to move should not count as an error. So we need

to consider to average out movements over multiple frames in order to determine the correct movement predictions.

What we actually want to achieve is that the network is able to recognize action patterns and classify these as the equivalent Roles as defined in STP. Because all Roles are included in the log-files these serve as ground truth. However, our interpretation of Intentions and Roles may differ from these of our opponents. So we will include a second way of establishing ground-truth, which is done using the inspection boxes of the logfile (See Figure 11). Here we can select a particular agent, inspect it's trajectory and offer a classification and role, allowing the network to learn to recognize similar situations.

4.7. Validity of our approach

Converting a continuous movement to a single action and then back again to a continuous motion requires careful consideration because it involves a loss of information. We expect the simulator to correct for this by integrating the movements into a velocity that increases by the time it continues in the same direction. We carefully need to determine if this is the case. Therefore we perform a number of calculations to find out how large the differences are.

For agent actions this problem is less severe, because every next frame will form a successive refinement of the direction. But for the Ball movements this is more difficult. In case of a pass to another robot, the movements are divided into 8 angles, spanning a circular areas of 45 degrees. The farther away the target is from the robot, passing the ball, the larger the deviation will be. At the moment of shooting, the robot needs to accurately determine both the angle and distance that has to be covered. This cannot be calculated from the information in two successive time frames. So the network needs to learn the angle and distance by observing, preferably the entire time of the pass.

An better approach would be to find out where the pass is targeted to. We can do this by taking the end position of the ball in the episode and find the agent that is closest to that position. Successive steps cannot correct the ball direction, because this is determined at the moment of shooting. So guessing who the receiver of the pass or shooting will be the best approach.

4.8. Further Analysis

Using the logfiles not only allows capturing the behavior of opponent teams, it also allows analysis of weak spots in the strategy of both teams. Any time when the ball is lost or when an shot on goal is attempted, the team that suffered from a weak move has a disadvantage. By assigning a negative reward to these episodes, the network learns to recognize undesirable situations and can identify such moments in the strategy. Using the network, configurations can be found, where this weakness leads to an advantage for the opponent. This allows concentration of fixing these weak spots or taking advantage for the opponent's weak spots.

Therefore our research is aimed at the following points:

1. Learning to predict the actions of all participating teams in a package, suitable for inclusion in a simulator. It takes a simplified world model as input and outputs actions for each agent of a team for all game situations.
2. Have a second network, working from the same input that signals the strength of the current situation, taking advantage of learning the weak points and strong points of both teams. The network can signal if we enter such a situation.

Considerations for design and development:

1. Deep Recurrent Q-Learning Networks (DRQN) networks are mostly used to learn a policy, given a certain state. In our case we want to accurately capture a team's behavior, not improve on it.
2. Is a DRQN or DDRQN the best architecture for this? Calculating the Q value should concentrate in this case on how well the model predicts the behavior. The best way to test this is to train the model on the first half of the competition and verify it with the second half of the competition, which the model has not seen before.
3. Do we also take setup and buildup situations into account like preparing for a kickoff and performing an active and passive kickoff. For now we only concentrate on a running game, which starts after the kickoff has occurred.
4. Do we train on the world model or on per-agent partial observable images, like the agents do in a real game. Or maybe a combination of a partial image, enhanced with info about the ball and goal positions and location on the field.
5. Study the use of Parameterized Action Spaces to allow learning of functions with parameters

like kicking with a certain force and direction. This can be realized by some of the newer DRQN networks with continuous control functions. (Hausknecht2016 [39, 60]).
6. Investigate CRF and HAN to learn the intention of every agent and infer a role from the Set-Piece.
7. Generate a complete Set-Piece and find a match in the database to classify the strategy.

5. Game Situation Classification

Game Situation Classification is a matter of recognizing a number of important game states, like Ball Possession, the area on the field where the play is taking place and the Formation that both teams have selected. Game State Transitions are depicted in Figure 14. The network must learn to predict both the Game Status as well as the Roles of all players in each time step. The game state is expressed in a number of situations as shown in Tables 4 and 6.

However there is an important inter-dependency between the observed game situation and the actions of agents. The Formation cannot be deduced directly from the initial positions of all agents, but must be inferred from their roles. These roles in turn can only be inferred from the actions each agent is taking during an episode. So the initial stated is given by features like Ball-Possession and the playing half and position on the field, while the Formation is inferred later and then added to the initial game situation. When roles are changed during an episode, a new formation may have been selected and we need to be able to detect this. Role changes are recorded in the logfile for our own team, so we use this information as ground-truth to learn to recognize actions and classify them as Roles. This learned classification is then used on the opponent's team in order to make the same analysis on the Opponent's behavior.

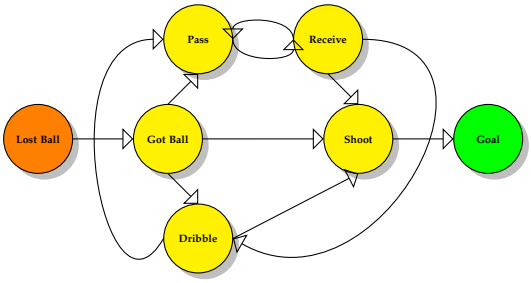

Figure 14: *Game State Transition Diagram. Lost Ball means that ball is lost to the opponent*

The main purpose of the Game Situation Classification is to evaluate the current position of each team, in order to determine where important decision points occur. At any moment the ball may be lost and the state then returns to Lost Ball. This is not shown in the diagram to simplify the layout. This part of the network must learn, what the critical decision points are during an episode, that may lead to ball loss or ball possession. It is also needed to give rewards or punishments to actions that an agent takes to learn the behavior of a team of cooperating agents.

When training a neural network to classify images, the main component is the detection of features, that discriminate one object from another. In our case we have a time-series of successive images, leading to a certain event. The network needs to learn, where in this sequence a situation occurs that is crucial to the outcome of the episode. This is generally best done with a Recurrent Neural Network.

In other domains, where similar approaches are used, like Natural Language Processing (NLP), normally a set of words or items is defined and the network learns to predict the probability of the next word occurring. As the number of words in spoken or written text can be enormous, special techniques have been developed to manage such a library of words or tokens, using so-called embeddings.

The game situations that occur in our case, are mostly defined by the configuration of positions of players on the field. The number of different combinations may be overwhelming. We can drastically cut the number of possibilities, by preprocessing the game state into a number of properties, that we know are relevant to the game. These of course, are the same kind of properties we are looking for when programming our robots to play soccer. Table 4 lists examples of these properties in approximate order of importance.

Game States			
Aspect	Reward	Occur%	Prop
Ball possession	1.0	0.0	0.0
Offence, Defence	0.8	0.0	0.0
Our Half, Opp half	0.6	0.0	0.0
Ball dist to goal	0.6	0.0	0.0
Free path to goal	0.4	0.0	0.0
Dist to peer	0.2	0.0	0.0
Free path to peer	0.2	0.0	0.0
Num of players	-1.0	0.0	0.0

Table 4: *Different Game States with manually assigned features*

These aspects are constantly calculated during a game and are recorded in our log files, but we do not have such information about our opponent's players. The game situation is evaluated in a world model and in individual models for each agent. So as a first attempt we will let the network learn to classify each game situation according to the properties that we have selected.

We start with these hand-crafted properties but later on we might let the network attempt to find these

features by the learning process, if we find that a more refined scheme is required. In Table 6 we give a more detailed classification, based on the Futsal Set-Pieces, in which the STP classifications also needs to be integrated. Note that the hand-crafted features of the Futsal classifications are more symbolic than those of the STP framework. We must find a way to bring these two world closer together.

It would be very beneficial if we could compare different matches against the same team to see if we can find any differences in strategy.

Game Classification					
Attack					
Half	Ball	Action	Form	Opp Action	OppF
Opp		BuildUp		Small Own Half	
Opp		BU 3rd Man		Small Own Half	
Opp		BU Switch Center		Small Own Half	
Opp		BU HOVO		Small Own Half	
Own	1	BU Spanish	1-3-1	Pressure Opp Half	1-1-3
Own	1	BU Spanish	1-4-0	Pressure Opp Half	1-1-3
Own		BU Own Keeper		Pressure Opp Half	
Defend					
Half	Ball	Action	Form	Opp Action	OppF
Opp	1	Disturb Buildup	1-1-3	Buildup	1-3-1
Switch					
Half	Ball	Action	Form	Opp Action	OppF
Opp	*	Disturb Buildup		Buildup	
Own	*	Buildup		Disturb Buildup	

Table 6: *Describing Game States in terms of Set-Pieces and STP Plays. This table only contains the KNVB Set-Pieces. FIFA and UEFA still have to be added, as well as the current STP Plays*

5.1. Previous Work

Most of the work on Multi-Agent Systems and Opponent Modeling has been done in the RoboCup domain. In recent years however a growing interest is shown in work from the Sports Analysis world. For a variety of sports, video files from competitions are used to analyze team behavior.

In work on Formation Analysys for the RoboCup Simulation League, an 8x8 grid is used to let a Neural Network recognize pre-defined formations in 10x10 soccer. (Visser2001 [111]). Here a decrease in granularity is used to recognize formations, where a 72.27% accuracy was achieved. In work, performed by Disney in cooperation with CalTech and STATS, a method called Stochastic Variational Imitation Learning is used to predict the behavior of agents in tracking data from 45 European Professional soccer teams. Based on the trajectories of the players, a structural model is made, which serves as a formation analysis. From this the roles of the agents are inferred in a subsequent phase.

Team Formations					
Type	Name	Phase	D	M	A
2-1-1	Pyramid	Defense	2	1	1
3-0-1	Wall	Defense	3	0	1
2-0-2	Square	Neutral	2	0	2
1-2-1	Diamond	Neutral	1	2	1
1-1-2	Y	Attack	1	1	2
1-0-3	AllOrNone	Attack	1	0	3

Table 5: *The most commonly used formations in Futsal. The last columns show number of Attackers, Defenders and Midfielders. There are additional formations in which the keeper is involved as another player.*

5.2. Analyzing Formations

One of the biggest problems in classifying a game situation is the large number of possible permutations as a result of relative player positions. Some attempts have been made to overcome this by decreasing the granularity of the position grid, but this has a big impact on recognizing patterns. In Soccer- and Futsal analysis, formations are used to determine such patterns and we want to use the same approach by letting the network learn to recognize the formations of both teams.

In Futsal there are several formations, some of which are used more frequently than others. They also depend on whether the team is attacking or defending. In order to properly classify the game situation, we therefore need to determine the current formation of the teams as part of the game situation. This classification is important, since the position of each player determines its role in the game during that game situation.

Formations can change dynamically, and while classifying the game situation, each robot is assigned a temporary number and role, so we can communicate about the Set-Piece and the steps that are taken. The various formations are listed in Table 5 and some of these formations are also used in the Game Situation Classification in Table 6. A good overview of these formations is given in [30].

6. Analyzing Agent Actions

Once the initial Game Situation has been classified and the Set-Piece has been established, the steps of the Set-Piece, e.g. the movements of the agents need to be found and classified. This involves finding the trajectories of all agents and classifying these into the roles they play as part of the current formation.

This analysis takes place on three levels:

1. *What* the robot does (Actions). These are the movements, derived from the temporal differences in an episode.
2. *Why* the robot performs these actions (Intention / Role). These must be learned, using hand-crafted features or from an AutoEncoder.
3. *How* the robot performs that role (Direction / Velocity). These can be learned from the logfile for our own robots, but cannot be derived for opponent robots.

Especially the *How* aspects can play an important role, since there may be fine nuances in the way that a role is being executed. For instance a robot may be rotating with the ball, to hide it from an opponent, or may be involved in a scrum. That information is not available for our opponents but could be important. Therefore we first will investigate if this is an important factor with our own robots. If so, we must find way to infer this information from camera images.

Game Situations			
Aspect	Reward	Steps	Occur%
Goal	1.0	0.0	0.0
Shoot	0.8	0.0	0.0
Pass	0.6	0.0	0.0
Receive	0.6	0.0	0.0
Dribble	0.4	0.0	0.0
Get Ball	0.2	0.0	0.0
Lost Ball	-1.0	0.0	0.0

Table 7: *Different Game Situations the network trains for, their stepsize, occurrence and rewards. (Info still to be collected)*

6.1. Previous Work

An important part of classifying Agent Behaviors is to determine the Intention of an agent. This is described in the belief/desire/intention (BDI) approach, which is a popular technique for modeling other agents (Haddadi1995 [**?**]). Another way is to let a user to enter the Intention or Role manually during collection of episodes, like is done in the Pretend Play environment (Singh2016 [90]). A more recent approach was taken by STATS and Disney, using a number of AutoEncoders that take the Game Situation (Context), Trajectory and Player Identities (Felsen2018 [27]) as inputs to a Conditional Variational AutoEncoder (CVAE). Another approach is used in a study of people movements in an apartment Kasteren2008 [102], where Conditional Random Fields (CRF) are used and compared with a Hidden Markov Model (HMM). A similar approach, combined with Attention Networks is described in (Chen1711 [15])

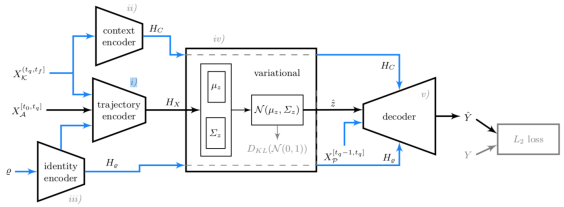

Figure 15: *CVAE Encoder/Decoder (Felsen2018).*

6.2. Agent Actions

Using the log data, we know the positions of all players on the field and so we can calculate the actions of all agents of a team during an episode. Although we know the differences in position, the Game State Images are reduced to a 40x28 pixel grid, in which position details are rounded to grid cells. We calculate what action an agent makes and code these in a one-hot classification, as depicted in Table 8. This reduces the movements of an agent to a maximum of one cell per step. Given the maximum speed of the robots, this is not a problem, but when distances of less than a cell are traveled, this could mean that information is lost. Therefore the movements are always calculated on the actual positions and not on the grid positions.

The network must learn to associate Game Situation Images with agent actions. To this end first a Convolutional Network (ConvNet) is trained, that transforms the GSI images to groups of features, that help in determining the actions an agent takes. Since this happens in a time sequence, a Recurrent Cell is used in the last layer of the network, that learns to predict the actions for each agent. So every step of an episode is first converted to an image in which the current agent is identified, and then fed into the network, so it learns all actions for all agents in a team.

A Recurrent Network is able to make such predictions by receiving rewards towards achieving a goal and we need a mechanism to give these rewards or punishments. This will be done by another network, that learns to classify the game situation and will be discussed in section 5.

We verify how well the network learns to predict these actions by comparing the predictions of the network with the actual actions that are defined in the logfile. However, when using these predictions in the simulator, all we know is that an agent moves in a certain direction during a single time step, we do not know its velocity and also directional information is less accurate, since it is split into 45 degree steps. The simulator needs to integrate this information over time and we will verify if this leads to reliable results.

If this is not the case, we will need to change our approach from learning a classification of actions to a regression, and learn the velocity in the X and Y direction. For that we will need to implement a network with parameterized action spaces.

Agent Actions				
Pos	X,Y	Degr	Class	Remark
	0,0	0	0	
	+1, 0	0	1	
	+1,-1	45	2	
	0,-1	90	3	
	-1,-1	135	4	
	-1, 0	180	5	
	-1,+1	225	6	
	0,+1	270	7	
	+1,+1	315	8	

Table 8: *Coding agent moves into 9 classes.*

6.3. Analyzing Intentions

We generate step-related actions for each agent, because the simulator needs an update on every cycle. But such an approach has no bearing on what actually happens during the game. Therefore we take a more symbolic approach and take the Intentional Stance. This has explanatory power, especially if we can link it on a symbolic level to the known elements in a Set-Piece or part of the STP framework.

The way that we intend to do that is to take the trajectory of each robot during an episode and deduce from this trail, what the intention of the agent is at that moment. This approach is similar to the way people movements in a house are classified as activities, using either Conditional Random Fields or a Hierarchical Attention Network. We will be training a network to classify movements into a number of representative categories, as depicted in Table 9.

This however introduces a problem, because the Set-Piece learns a number of steps on the symbolic level, that spans a number of time-steps. The simulator needs position information for each time step. The network learns a series of steps, while the Set-Pieces learn the meaning of these steps. STP operates on the level of a number of roles that are translated into continuous actions by the TURTLE software. Somehow we need to find a similar distribution of functions between the symbolic and sub-symbolic levels. We have yet to decide how to do that.

6.4. Ball Actions

We also need to learn the behavior of the agent that handles the ball. Here we need to learn actions and in addition we have to deal with velocities that are much higher than one cell per time step. We use the same approach as with agent movements, but also need the agent's intentions with respect to the ball. These intentions are depicted in Figure 14.

Where agent actions are normally continually monitored and change with each time step, ball actions are always momentarily. When an agent shoots, this happens at a given moment, giving the ball a direction and velocity that shows up in subsequent time frames, but are no longer correlated with the action of that step. So the network must learn what the agent did at the originating moment.

So for Ball actions we need two distinct action components, first the action direction part and secondly the intentional part. Therefore a third stream of the network must learn both the action sequence and the intention during the most important step in an episode, namely the moment where the agent releases the ball. This situation is called a BallFree moment, namely OurBallFree or OppBallFree. During this transition period, the ball can be moving towards a team member in a Pass, or be moved during a Dribble or during a Shooting action. Only at the end of the episode will be clear if this action results in ball loss. This component

Agent Actions		
Attack		
Game Situation	Action	Remark
Buildup (BU)	Pass Ball To X	
	Outside In	
	Inside Out	
	Triangle on Side	
	Deep on Blind Side	
	Shoot on Goal	
BU 3rd Man	Walk Away Diagonally	
	One-Two	
	Rest-Defense	
	Ball Diagonally	
	Ball Along Line	
	Close By X	
	Close By Other Side	
	Pass Ball To Incoming X	
BU Switch Center	Move Deep	
	Offer In Back Of X	
	Dribble Shoot On Goal	
	Play Ball At Pole 2	
BU HOVO	Play Back and Move Deep	
	Offer At Side	
BU Spanish	Keeper Play Deep	
	Move Behind Opp	
	Make Space	
	Play Ball Wide	
	Play Ball To Mover	
	Drop Side Own Half	
	Offer at Center	
	Turn Back	
	Long Ball On Deep Mover	
	Dribble To Goal	
BU Own Keeper	Play Ball Back To X	
	Move Deep From Side	
	Hold Ball	
	Field Large Ball Side	
	Move To Pole 2	
	Give Long Ball	
STP	Attack-Assist	
	Open-Passage	
	Move-to-X	
Defend		
Game Situation	Action	Remark
Disturb Buildup	Hold-up	
	Prevent Deep Pass	
	Force Wide Pass	
STP	Block-Passage	
	Move-to-Intercept	
	Mark-Agent-X	
	Defend-Line-X	
	Defend-Area-X	
Switch		
Game Situation	Action	Remark
Disturb Buildup	Constant Cover	
	Prevent Deep Pass	
	Pressure Player Ball	

Table 9: *Examples of Agent Action Types, derived from the KNVB Futsal Coaching Manual. This needs to be further refined and coordinated with STP Actions.*

Ball Actions		
Action	Class	Remark
Dribble	0	Keep the ball within 2 grid cells
Pass	1	Shoots in direction of team mate
Receive	2	Detects the ball within its VOF
Shoot	3	Shoots the ball towards the goal

Table 10: *Coding ball actions into 4 classes.*

of the network is the most complicated and therefore we will first concentrate on the agent actions, which are easier to realize and are more in accordance with existing technology. Together with the Game Situation Classification it will form the baseline of our approach.

7. The Neural Networks

Many different network architectures exist to realize Machine Learning applications. Our research is directed towards an application that will learn to correctly classify game situations, predict individual agent's moves and learn to recognize the intentions of individual players as well as the entire team. For this we will need a multitude of Neural Network approaches. In this section we describe the ones we will most likely be using.

7.1. Network Types

We are studying two different network types to find out which one best represents the information that needs to be learned. A Deep Recurrent Q-Learning Network (DRQN) can learn sequences of up to 100 moves per game situation that capture the temporal properties of the robot behavior. This will learn to plan ahead for 100 steps of 10 ms, thus spanning a period of up to 10 seconds.

The action sequences in such a network are learned by a recurrent cell at the end of a convolutional network, that transforms the Game Situation Images (GSI) into input data to the recurrent cell and let it learn which actions are usually taken by agents of a team, when confronted with a given game situation.

7.2. Recurrent Networks

Learning the behavior of the players of a soccer team is a problem of associating actions taken during a game with the results.

Recurrent Neural Networks (RNN) can be implemented in different ways. Long Short Term Memory cells (LSTM) and Gated Recurrent Units (GRU) perform the same function, but have a different architecture. Bidirectional LSTMs are a variation to this theme, that generally perform better than their unidirectional counterparts. We will test the various approaches in a neural network that will learn sequences of actions from the logfiles and be used to predict which action each agent of a team takes during a game.

The digital log files that we use contain low level information and we concentrate on learning game actions that maintain ball possession and regaining the ball, once lost.

7.3. Attention Networks

Recurrent Neural Networks (RNN) with LSTM or GRU cells only learn dependencies on previous time steps. In our case we expect more dependencies to be relevant, like the position of nearby and opponent agents, the current game situation, the position of the ball. More complex dependencies are better represented by Conditional Random Fields and Attention Networks.

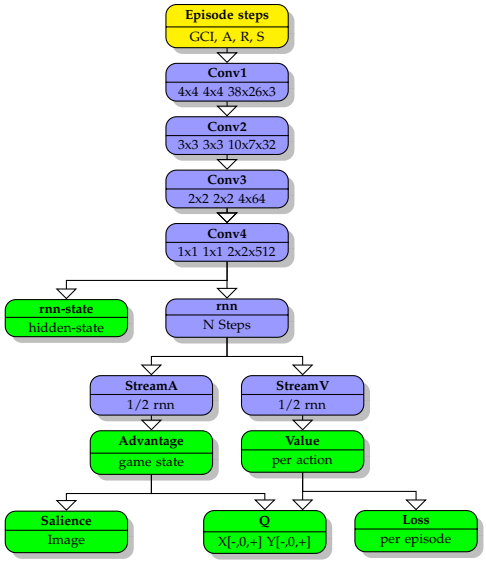

Figure 16: *Architecture of the DRQN network*

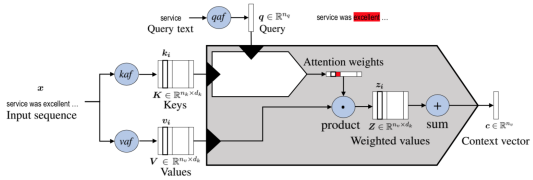

Figure 17: *Unified Attention Model.*

We expect that a combination of a Hierarachy of Attention Networks (HAN) with a Deep Q-Learning Network (DQN) will be the best approach to learn the behavior of agents as part of a Set-Piece.

In order to learn the intentions of every agent during an episode, we will need to learn sequences of actions and classify them according to a set of standard behaviors as described in Table 9.

7.4. Loss and Rewards

In a system, where a DRQN network is used to learn to improve performance, the Q value is calculated based on how well the agent performs. So in case of a robot soccer match, a reward is given if the team keeps or seizes the ball, moves the ball toward the opponent's goal, shoots at the goal or scores a goal. In our case

we need to measure how well the network predicts the next move. This can easily be done, because our captured data tells us exactly what happened during a competition. In fact, it is very important to properly represent the opponent's weaknesses, so these can be exploited during a simulation. Therefore the reward system should award points when predicting the next move correctly and give penalties when the next move differs from the prediction.

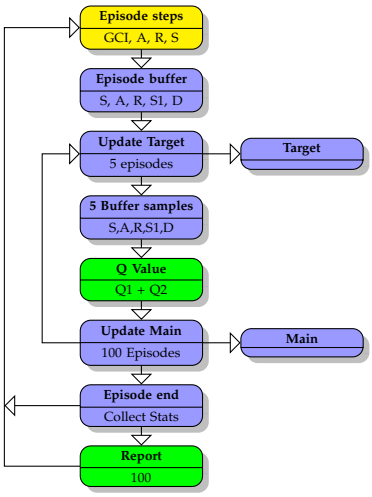

Figure 18: *Training the DRQN Network*

7.5. Training the Neural Network

We are now in a position to let the network learn the different Game Situations. Our intention is that the network will find the most important Turnover Point during a game that leads to a certain event. In Table 7 we listed the different Game Situations in order of importance. Scoring a goal is the most important event, but several other events, preceding a goal must prepare for this. We expect that the log files contain sufficient information to let the network learn when a given Game Situation may lead to a better (or worse) situation.

7.6. Using the network in Simulation

In this project we deal with two simulators: the simple simulator that is part of this project and the TURTLE GreenField Simulator. Our simple simulator accepts single position updates for each time-step and visualizes the contents of the logfiles. When using the learned Neural Network as input, it will receive the position predictions for each agent and show this on the simulated field.

The TURTLE simulator can also visualize a logfile, but in addition it will take the position information and use the actual strategy code implementation to determine the next move for our team. The learned network needs to output position information for the competing team, based on the current world model and therefore needs a Game Situation Image (GSI) as input for the network.

Because we have both a symbolic and sub-symbolic level in the learning environment, we need to decide which one is leading. Somehow, the symbolic (intentional) level is closely related to the step generator and there will always be two (or even three) outputs on every time-step:

1. Position information for every agent and the ball
2. Symbolic information for explanation purposes
3. Game Advantage / Threat information

For the position information, the use case is simple. It will be supplied to the simulator and is used to update the visualization. The other two are a bit more complicated, since they provide information that is of no use to the simulator itself but is used to explain what the competitor would be doing. However, some additional information could be provided as well. The Advantage / Threat information can supply information about the current situation and could be used as coaching information, to advice the team on the strategy to follow. It could also give an indication of weak and strong points in the strategy of both teams.

However the information in the Set-Pieces could also be used to show alternatives or help in analyzing how other teams would react to a similar situation. We need to explore these possibilities further, when the development is in a more advanced stage.

8. Testing various Set-Pieces

This work was originally started as an attempt to learn to play soccer strategies. At first we intended to learn from the training manuals for Futsal, as published by FIFA and UEFA. Later we got in touch with Max Tjaden, the Dutch KNVB Bondscoach for Futsal, who gave us the KNVB Futsal Training Manual. We implemented the set-pieces included in that work with the idea of training a network with these examples and let the system learn to improve on that.

Although we still maintain that idea, it's implementation is a rather big step, so we decided to first concentrate on a less ambitious goal and work on learning to predict the behavior of our opponent teams.

The current work will concentrate on classifying and predicting action sequences, collected in the MSL logfiles that we collected over the past 9 years. We will use the KNVB Futsal Training data to find out how different teams react to the situations, described in these exercises, as well as those from the FIFA and UEFA coaching manuals.

9. Conclusions and further work

References

[1] Symbolic reasoning (symbolic ai) and machine learning skymind.ai/wiki/symbolic-reasoning.

[2] MachineTalk.org 2019. Create the transformer with tensorflow 2.0 machinetalk.org/2019/04/29/create-the-transformer-with-tensorflow. 2019.

[3] Rubikscode.net 2019. Introduction to transformers architecture rubikscode.net/2019/07/29/introduction-to-transformers-architecture. 2019.

[4] Albrecht2018. Not yet summarized.

[5] Andre1998. Not yet summarized.

[6] Kai Arulkumaran, Antoine Cully, and Julian Togelius. Alphastar: An evolutionary computation perspective.

[7] Timur Bagautdinov, Alexandre Alahi, Francois Fleuret1, Pascal Fua1, and Silvio Savarese. End-to-end multi-person and action localization and collective activity. 1431.

[8] Balch1998. Not yet summarized.

[9] Adrien Bennetot, Jean-Luc Laurent, Raja Chatila, and Natalia Díaz-Rodríguez. Towards explainable neural-symbolic visual reasoning. Technical Report is, 1909.

[10] Birnsted1999. Not yet summarized.

[11] Joydeep Biswas, Juan P. Mendoza, Danny Zhu, Benjamin Choi, Steven Klee, and Manuela Veloso. Opponent-Driven Planning and Execution for Pass, Attack, and Defense in a Multi-Robot Soccer Team. Technical report, 2014.

[12] Brett Browning, James Bruce, Michael Bowling, and Manuela Veloso. STP: Skills, tactics and plays for multi-robot control in adversarial environments. 2004.

[13] David Carmel and Shaul Markovitch. Incorporating Opponent Models into Adversary Search. 2000.

[14] Carlos Celemin, Rodrigo Perez, Javier Ruiz del Solar, and Manuela Veloso. Interactive Machine Learning Applied to Dribble a Ball. 2017.

[15] Zheqian Chen, Rongqin Yang, Zhou Zhao†, Deng Cai, and Xiaofei He. Dialogue Act Recognition via CRF-Attentive Structured Network. 1711.

[16] Guangchun Cheng, Yiwen Wan, Abdullah N. Saudagar, Kamesh, Namuduri, and Bill P. Buckles. Advances in human action recognition: A survey. 1501.

[17] Francois Chollet. Deep Learning with Python.

[18] Philip Cooksey, Devin Schwab, Rui Silva, Rishub Jain, Yifeng Zhu, and Manuela Veloso. CMus 2018 Team Description. Technical Report years., 2018.

[19] Eugenio Culurciello. The fall of RNN / LSTM. 2018.

[20] Lotte de Koning, Juan Pablo Mendoza, Manuela Veloso, and René van de Molengraft. Skills, Tactics and Plays for Distributed multi-robot control in Adversarial environments. 2017.

[21] Henk de Swaan-Arons and Peter van Lith. *Expert Systemen*. Academic Service, 1984.

[22] DeepMind2019. Learning explanatory rules from noisy data deepmind.com/blog/article/learning-explanatory-rules-noisy-data.

[23] Daniel Dennett. Intentional Systems Theory. 1971.

[24] Artur d'Avila Garcez, Marco Gori andLuis C. Lamb andLuciano Serafini, Michael Spranger, and Son N. Tran. Neural-symbolic computing: An effective methodology for principled integration of machine learning and reasoning. 1905.

[25] Echen. Introduction to Conditional Random Fields. 2012.

[26] Sean R Eddy. What is a hidden Markov model? 1004.

[27] Panna Felsen, Patrick Lucey, and Sujoy Ganguly. Where Will They Go? Predicting Fine-Grained Adversarial Multi-Agent Motion using Conditional Variational Autoencoders. 2018.

[28] FIFA. FIFA Futsal Coaching manual.

[29] Foerster2018. Not yet summarized.

[30] FotsalExpert.com. 6 Amazing Futsal Formations And Team Setups. 2018.

[31] Andrea Galassi, Marco Lippi, and Paolo Torroni. Attention please! a critical review of neural attention models in natural language and processing. 1902.

[32] Marta Garnelo and Murray Shanahan. Reconciling deep learning with symbolic artificial intelligence: representing objects and relations. *Current Opinion in Behavioral Sciences*, 29:17–23, oct 2019.

[33] Zoubin Ghahramani. An Introduction to Hidden Markov Models and Bayesian Networks. 2001.

[34] Robit Ghosh. Deep Learning for Videos: A 2018 Guide to Action Recognition. 2018.

[35] Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous Deep Q-Learning with Model-based Acceleration. 1603.

[36] Danijar Hafner. Variable Sequence Lengths in TensorFlow.

[37] Roland Hafner, Sascha Lange, Martin Riedmiller, and Stefan Welker. Brainstormers Tribots Team Description. Technical Report award., 2002.

[38] Harnad. Grounding Symbols in the Analog World with Neural Nets. Think 2: 12-78 (Special and Issue on Connectionism versus Symbolism D.M.W. Powers and P.A. Flach and eds.), 1993.

[39] Matthew Hausknecht and Peter Stone. Deep Reinforcement Learning in Parameterized Action Space. 2016.

[40] Matthew Hausknecht, Peter Stone, Department of Computer, Science, The University, of Texas, and at Austin. Deep Recurrent Q-Learning for Partially Observable MDPs. 2015.

[41] He He. Opponent Modeling in Deep Reinforcement Learning. 2016.

[42] Pablo Hernandez-Leal, Bilal Kartal, and Matthew E. Taylor. Is multiagent deep reinforcement learning the answer or the question? A brief survey. University of Alberta CCIS 3-232 Edmonton and Canada, 2018.

[43] Patrick Hohenecker and Thomas Lukasiewicz. Ontology reasoning with deep neural networks. 1808.

[44] ICLR2018. Composable planning with attributes. 2018.

[45] José Antonio Iglesias, Juan Antonio Fernandez, Ignatio Ramon Villena, Agapito Ledezma, and Araceli Sanchis. The Winning Advantage: Using Opponent Models in Robot Soccer. 2009.

[46] José Antonio Iglesias, Agapito Ledezma, and Araceli Sanchis. A Comparing Method of Two Team Behaviours in the Simulation Coach Competition. In V. Torra et al., editors, *MDAI 2006*, volume 3885 of *LNAI*, page 117 – 128. Springer, 2006.

[47] Sina Iravanian and Sahar Araghi. *Grid Soccer Simulator 1.0: User's Manual*, 2011.

[48] Arthur Juliani. Simple Reinforcement Learning with Tensoflow: Part 3 - Model-Based RL.

[49] Andrej Karpathy. The Unreasonable Effectiveness of Recurrent Neural Networks. karpathy.github.io/2015/05/21/rnn-effectiveness/2015, 2015.

[50] Konda2000. Not yet summarized.

[51] Andrey Kostyushko. Turn-based football. How is that possible. 2015.

[52] Mateusz Kurek. Deep Reinforcement Learning in Keepaway Soccer. 2015.

[53] Hoang M. Le, 1 Yisong, Yue, Peter Carr, 2 Patrick, and Lucey. Coordinated Multi-Agent Imitation Learning. 2017.

[54] Agapito Ledezma, R. Aler, Araceli Sanchis, and D. Borrajo. Predicting Opponent Actions in the RoboSoccer. 2002.

[55] Hector J. Levesque. *Common Sense, The Turing Test and the Quest for Real AI*. MIT Press, 2017.

[56] Ming Liang and Xiaolin Hu. Recurrent Convolutional Neural Network for Object Recognition. State Key Laboratory of Intelligent and Technology and Systems, 2015.

[57] Q. Liao and T.A. Poggio. Symbolic Reasoning (Symbolic AI) and Machine Learning skymind.ai/wiki/symbolic-reasoning.

[58] Qianli Liao and Tomaso Poggio. Object-oriented deep and learning and by. 2017.

[59] Qianli Liao and Tomaso Poggio. Object-oriented deep learning. 2017.

[60] Timothy Lillicrap, Jonathan Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra at Google Deepmind. Continuous Control with Deep Reinforcement Learning. 2016.

[61] Littman1994. Not yet summarized.

[62] Ziyin Liu, Junxiang Chen, Paul Pu Liang, and Masahito Ueda. Relational Attention Networks via Fully-Connected Conditional Random Fields. 2018.

[63] Jason Brownlee Machinelearningmastery.com. How does attention work in encoder-decoder recurrent neural networks. 2017.

[64] Gary Marcus, New York, and University. Deep learning: A critical appraisal. 2017.

[65] Jan Martinovic, Vaclav Snasel, Eliska Ochodkova, Lucie Zolta, Jie Wu, and Ajith Abraham. Robot Soccer and -Strategy Description and Game Analysis. 2010.

[66] Matsubara1999. Not yet summarized.

[67] L.A. Meerhoff, A. de Leeuw, F.R. Goes, and A. Knobbe. Mining Soccer and Data: Discovering patterns of tactics in tracking.

[68] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, and Ioannis Antonoglou. Playing Atari with Deep Reinforcement Learning. 2013.

[69] Volodymyr Mnih, Adrià Puigdomènech, Mehdi Mirza, Alex Graves, Tim Harley, Timothy P. Lillicrap, David Silver, and Koray Kavukcuoglu. Asynchronous Methods for Deep Reinforcement Learning. 1602.

[70] Heiko Muller, Martin Lauer, Roland Hafner, Sascha Lange, Artur Merke, and Martin Riedmiller. Making a Robot Learn to Play Soccer Using Reward and Punishment. 2006.

[71] Suraq Nair. A Simple Alpha(Go) Zero Tutorial.

[72] Ehsan Nazerfard, Barnan Das, Barnan Das, and Diane J. Cook. Conditional Random Fields for Activity Recognition in Smart Environments. 2010.

[73] Christopher Olah. Understanding LSTM Networks. colah.github.io/posts/2015-08-Understanding-LSTMs, 2015.

[74] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep Exploration via Bootstrapped DQN. 1602.

[75] Matthias Plappert. keras-rl reinforcement learning library. `https://github.com/keras-rl/keras-rl`, 2016.

[76] Shookofeh Pourmehr and Chitra Dadkhah. An Overview on Opponent Modeling in RoboCup. 2010.

[77] Paul Power, Hector Ruiz, Xinyu Wei, and Patrick Lucey. "not all passes are created equal:" objectively measuring the risk and reward of passes in soccer from tracking data.

[78] Hossein Rahmani, Ajmal Mian, and Mubarak Shah. Learning a deep and model for human and action and recognition from novel and viewpoints. 1602.

[79] RhythmLynx. Definitely Real Football. `https://rhythmlynx.itch.io/definitely-real-football`, 2017.

[80] Martin Riedmiller, Thomas Gabel, Roland Hafner, and Sascha Lange. Reinforcement learning for robot soccer. *Autonomous Robots*, 27(1):55–73, may 2009.

[81] Martin Riedmiller, Roland Hafner, Sascha Lange, Martin Lauer, Dept. of Mathematics, and Informatics. Learning to Dribble on a Real Robot by Success and Failure. In *2008 IEEE International Conference on Robotics and Automation Pasadena, CA, USA, May 19-23, 2008*. IEEE, 2008.

[82] Sahota1994. Not yet summarized.

[83] N.J. Sales and R.G. Evans. An Approach to Solving the Symbol Grounding Problem: Neural Networks for Object Naming and Retrieval.

[84] Daniel Sanchez Santolaya. Using Recurrent Neural Networks to Predict Custom Behavior from Interaction Data - Master Thesis 2017 UvA.

[85] Tom Schaul, John Quan, Joannis Antonoglou, and David Silver. Prioritized Experience Replay. Published as a conference paper at ICLR, 2016.

[86] M.C.W. Schouten. Internship report: The Implementation of Setplays to the RoboCup middle-size League. 2018.

[87] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov, and OpenAI. Proximal Policy Optimization Algorithms. 1707.

[88] By Selene and Báez Santamaría. Predicting opponent team activity in a RoboCup environment. 1997.

[89] David Silver, Thomas Hubert, and Julian Schrittwieser. A general reinforcement learning algorithm that masters Chess, Shogi and Go through self-play. 2018.

[90] Kunwar Yashraj Singh, Nicholas Davis, Chih-Pin Hsiao, Mikhail Jacob, Krunal Patel, and Brian Magerko. Recognizing actions in motion trajectories using deep neural networks. Technical Report of, 2016.

[91] Luc Steels. The Symbol Grounding Problem has been solved. So what's next. 2005.

[92] Timo Steffens. Feature-Based Declarative Opponent-Modeling. In D. Polani et al., editors, *RoboCup 2003*, volume 3020 of *LNAI*, page 125–136. Springer, 2004.

[93] Peter Stone, Richard S. Sutton, and Gregory Kuhlmann. Reinforcement Learning for RoboCup Soccer. 2005.

[94] Peter Stone and Manuela Veloso. Multiagent Systems: A Survey from a Machine Learning Perspective. AT&T Labs and — Research and Computer Science and Department and 180 Park Ave and room A273 Carnegie Mellon University and Florham Park and NJ 07932 Pittsburgh and PA, 2000.

[95] Sunehag2018. Not yet summarized.

[96] Istvan Szita and Andras Lorincz. Learning Tetris Using the Noisy Cross-Entropy Method. 2006.

[97] Max Tjaden. KNVB Futsal Coaching manual.

[98] Felipe W. Trevizan and Manuela M. Veloso. Learning Opponent's Strategies in the RoboCup Small Size League. 2010.

[99] UEFA. UEFA Futsal Coaching manual.

[100] Lecture Notes Stratchclyde University. Making Better Decisions - Opponent Modeling.

[101] Hado van Hasselt, Arthur Guez, and David Silver. Deep Reinforcement Learning with Double Q-learning. 1509.

[102] Tim van Kasteren, Athanasios Noulas andGwenn Englebienne, and Ben Kröse. Accurate Activity Recognition in a Home Setting. 2008.

[103] Peter van Lith. *Kunstmatige Intelligentie*. CACI, 1982.

[104] Peter van Lith, Marinus van de Molengraft, Gijs Dubbelman, and Martin Plantinga. A Minimalistic Approach to Identify and Localize Robots in RoboCup MSL Soccer. 2018.

[105] M.E. van 't Klooster. Deep Learning for Opponent Action Prediction in Robot Soccer Middle Size League, Master Thesis TU Eindhoven. 2018.

[106] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Google Brain, Google Brain, Google Research, and Google Research. Attention is all you need. 1706.

[107] Veloso1996. Not yet summarized.

[108] Veloso1997. Not yet summarized.

[109] Marco Verch. What Are Saliency Maps in Deep Learning?

[110] Phongtharin Vinayavekhin, Subhajit Chaudhury andAsim Munawar, Don Joven Agravante, Giovanni De Magistris, Daiki Kimura, and Ryuki Tachibana. Learning and Understanding using Attention, Focusing on What is Relevant: Time-Series. 1806.

[111] Ubbo Visser, Christian Druc̈ker, Sebastian Hu̇bner, Esko Schmidt, and Hans-Georg Weland. Recognizing Formations in Opponent Teams. In P. Stone, T. Balch, and G. Kraetzschmar, editors, *RoboCup 2000*, volume 2019 of *LNAI*, pages 391–396. Springer, 2001.

[112] Hanna M. Wallach. Conditional random fields: An introduction. Technical Report MS-CIS-04-21, 2004.

[113] Heng Wang and Cordelia Schmid. Action Recognition with Improved Trajectories. ICCV - IEEE International Conference on Computer Vision, Dec 2013, Sydney, Australia. pp.3551-3558,10.1109/ICCV.2013.441 . hal-00873267v2, 2013.

[114] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. Dueling Network Architectures for Deep Reinforcement Learning. 1511.

[115] Skymind Wiki. A Beginner's Guide to Attention Mechanisms and Memory Networks. skymind.ai/wiki/attention-mechanism-memory-network.

[116] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive and survey on graph and neural and networks. IEEE, 2019.

[117] Jiechao Xiong, Qing Wang, Zhuoran Yang, Peng Sun, Lei Han, Yang Zheng, Haobo Fu, Tong Zhang, Ji Liu, and Han Liu. Parametrized Deep Q-Networks Learning: Reinforcement Learning with Discrete-Continuous Hybrid Action Space. 2018.

[118] Joyce Xu. Beyond DQN/A3C: A Survey in Advanced Reinforcement Learning.

[119] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. Hierarchical Attention Networks for Document Classification. 2008.

[120] M Yoon, J Bekker, and S Kroon. New reinforcement learning algorithm for robot soccer. *ORiON*, 33(1):1, jun 2017.

[121] Hong-Bo Zhang, Yi-Xiang Zhang, Bineng Zhong, Qing Lei, Lijie Yang, Ji-Xiang Du, and Duan-Sheng Chen. A Comprehensive Survey of Vision-Based Human Action Recognition Methods. 2019.

[122] Qunzhi Zhang and Didier Sornette. Learning like humans. 1707.

10. Appendix

10.1. Previous work

In this section earlier work on learning behaviors with robots is discussed. It forms the basis for the previous work section. (Still to be completed.)

10.2. Tutorials

During the initial investigation a number of tutorials and books have been very helpful in finding the relevant literature. We will describe each of them and list their main contributions that lead to the analysis of the papers mention in the following sections.

10.2.1 Chollet2018 - Deep Learning with Python [17]

This book on using TensorFlow with Keras was used along with Adrian RoseBrock's book on Python. Chollet is the creator of Keras and contributed to the development of TensorFlow. The book has been very informative and helpful during development.

10.2.2 Santolaya2017 - Using Recurrent Neural Networks to Predict Custom Behavior from Interaction Data - Master Thesis 2017 UvA [84]

This Master Thesis gives a good overview of the available literature on Recurrent Networks and provides a tested example of using a recurrent network.

10.2.3 Strathclyde2017 - Lecture Notes - Making Better Decisions - Opponent Modeling [100]

This is a presentation with informative sheets about various Opponent Modeling approaches.

10.2.4 Juliani2019 - Simple Reinforcement Learning with TensorFlow: Part 3 - Model-Based RL [48]

This excellent tutorial with many detailed coding examples for the various approaches for Reinforcement Learning. We are using the DRQN version. At the end there is also a reference to Actor Critic systems.

10.2.5 Hafner2019 - Variable Sequence Lengths in TensorFlow [36]

In our use case, experiences are formed from the Matlab log files. In these files we have variable sequence lengths. This article explains how to implement these in TensorFlow.

10.2.6 Nair2017 - A Simple Alpha(Go) Zero Tutorial [71]

We use this software, along with Juliani as a basis for the Alpha-Zero part of our system. Currently this is not yet in use, since we first concentrate on learning an Opponent Model.

10.2.7 Verch2017 - What Are Saliency Maps in Deep Learning? [109]

In the sample DRQN system of Juliani [48], that we use, saliency images are generated during exploration. This article explains what Saliency is. We use this facility in our visualization software.

10.2.8 Xu2018 - Beyond DQN/A3C: A Survey in Advanced Reinforcement Learning [118]

A good overview of most recent developments in DQN/A3C and attention in Recurrent Networks.

10.2.9 Brownlee2017 - How Does Attention Work in Encoder-Decoder Recurrent Neural Networks [63]

Good introduction on the Attention Mechanism in Encoder/Decoder RNNs.

10.2.10 Plappert2016 - Keras-RL Github Repository with examples (8) [75]

10.3. Papers on General Opponent Modeling

Opponent Modeling is the main technique we are going to use, to learn the behavior of opponent teams. As we do not have detailed information about strategies, we will first learn our own strategies and use our log-files to verify if our learning method is accurate enough. When that works, we use the same approach to learn the strategies of our opponents. The following papers describe what has been done so far on this subject. Most approaches use Implicit modeling, where the actual model is hidden and used to adapt a team's strategy. We are interested in Explicit Modeling, which has been studied much less.

10.3.1 Stone2000 - Multi-agent Systems: A Survey from a Machine Learning Perspective [94]

This is the seminal paper on Multi Agent Systems (MAS). It gives an overview of work done up till

2000 and lays the foundation for the study of Machine Learning in Soccer. In this survey Single- and Multi Agent Systems are analyzed in four different categories, each in two major domains; the pursuit world and robot soccer:

1. Homogeneous non-communicating agents
2. Homogeneous communicating agents
3. Heterogeneous non-communicating agents
4. Heterogeneous communicating agents

For each of these classes of agents, examples are given, as well as references to literature where these systems have been studied. In all cases they are also viewed in the context of learning systems. Another aspect that is described is benevolence against competitive agents. In robotic soccer, the Team Partitioning, Opaque-Transition Reinforcement Learning (TPOT-RL) of Stone (2000) is described. In this approach agent are capable to learn to adapt their behavior based on the opponent's perceived strategy. This has also been studied when one of the teams is learning. But when both teams are learning no stable solution was found.

Agents that cannot communicate are forced to model each other's behavior. In our case our own agents communicate, but there is no information on the opponent, so that behavior has to be modeled. References are made to work, in which the game situations, plans, goals, actions and roles of the opponent are modeled.

In case of a communicating team, members need to agree on the roles they play and there is frequent mention of the locker-room agreement (Stone and Veloso 1999). Agents that cooperate need to make commitments to the team goals. This is described in the belief/desire/intention (BDI) approach, which is a popular technique for modeling other agents (Haddadi1995 [**?**]).

A large section is dedicated to robotic soccer and much of the research is a precursor to the RoboCup initiative, which started in 1996 (Kitano 1996, 1998, 1999, Veloso 2000). Another interesting approach is a system that generates commentaries to observed soccer matches automatically, called SOCCER. This system is based on observing human soccer games and announces important events in an observed game. A ground-breaking development was Dynamite, which was used to develop Reactive Deliberation which was based on a number of hard-wired behaviors (Sahota 1994). It served as the basis and inspired much of Stone's work on robotic soccer. Minimax-Q Learning for Markov games was used in a simulated soccer game (Littman 1994) formed the basis for the later Soccer Simulation Server (Stone and Veloso 1996) . This server was also used in a commentator system ROCCO (Andre 1998) and MIKE (Matsubara 1999) and Byrne (Birnsted 1999).

Gradually more work was done on decomposing soccer tasks into different roles. A system with dynamically changing roles was developed by (Balch 1998) later followed by a RL approach, where an entire team learned effective behaviors instead of individual behaviors. Hierarchical task decomposition became the accepted way to let entire teams learn collective behaviors (Uther and Veloso 1997). It becomes clear that here, the foundations were laid down for later developments like STP (Browning 2004 [12]) and similar approaches.

10.3.2 Carmel2000 - Incorporating Opponent Models into Adversary Search [13]

This is one of the first examples of Opponent Modeling, using the M* algorithm, a modified alpha-beta search algorithm, which tries to find the best response to a given game situation. It introduces swindle and trap approaches, that let the opponent believe some strategy is being followed, that later on proves to be a disadvantage to the opponent. Like most of the early work on Opponent Modeling it is based on a statistical approach and does not return action sequences.

10.3.3 Hernandez-Leal2018 - Is multi-agent deep reinforcement learning the answer or the question? [42]

This survey of Opponent Modeling gives a systematic overview of the various approaches, developed over the past 20 years. It starts with the work of Stone and Veloso on Multi Agent Systems (MAS) [94].

It describes Single Agent Learning, followed by Reinforcement Learning, working toward the latest research on Agent-Critic algorithms. It then describes the work done on Deep Reinforcement Learning, like Q-Learning, of which Value-based methods like DQN are the baseline method. In most of the DQN approaches several input stages are stacked to represent time series, but generally speaking these systems cannot handle more than 4 time steps. Time sequences are better represented by Deep Recurrent Q-Learning Networks (DRQN) in which an LSTM layer learns the time-steps. Some systems have multiple LSTMs to represent different time-scales. Most of these systems work in discrete action spaces, where actions are classified as single actions.

In cases where the action space is continuous, like with soccer, actions are normally related to speed and/or direction and require a different approach, which is handled by Policy-gradient methods, like Deep Deterministic Policy Gradient (DDPG). Actor-Critic systems like A2C and Asynchronous Advantage Actor Critic (A3C) algorithms employ multiple networks that learn different aspects and are able to output continuous values. They depend on an Experience Replay (ER) buffer. In most cases such a buffer is either filled by exploring random possibilities or filled from live examples. In our case, the log files form a natural Experience Replay buffer. When learning new approaches, exploration is a requirement and is used for instance in Alpha-Go and the Atari games. If there are enough samples, the experience buffer is exploited to learn existing behaviors. Many new variants of these approaches have been developed recently.

For Multi-agent Deep Reinforcement Learning (MDRL) learning is more complicated. Instead of learning individual behaviors, such systems also learn team behaviors. Multi-agent research is divided into four categories:

1. Analysis of emergent behavior
2. Learning communication
3. Learning cooperation
4. Agents modeling Agents

We are most interested in the last category, where DRQN is succeeded by Deep Reinforcement Opponent Networks (DRON), where Q-values are learned that find representations of the opponent policy. The first approaches use hand-crafted features, that the network learns to recognize. Another approach is Self Other Modeling (SOM) which concentrates on finding out the opponent's goals based on the known behavior of a team's own agents. Most recent approaches here are Learning With Opponent Awareness (LOLA) and Theory of Mind Network (ToMnet). The main goal of such an approach is to predict the opponent's next action.

The most important aspects of most approaches are the following: The Experience Replay (ER) buffer, where sometimes additional information is added to disambiguate the contents. Parameter sharing is another important aspect, where parts of a network are shared to learn different aspects of the same game state. Recurrent networks are an important part to learn time sequences. Lastly a new network approach of ensembles is used to avoid over-fitting of opponent models.

In most cases hand-crafted features can speed-up the learning process, but more work is being done to let the network find out these features by itself. Self-Play and Monte-Carlo search is extensively used in most modern learning systems. However almost all of these approaches learn opponent policies as a method of learning proper responses to game situations. They all use Implicit Opponent Models, that are input to a learning system that finds new strategies. The most modern approaches even take this learning ability into account and model the fact that the opponent learns from its experience and uses this to coerce the opponent into taking actions that open opportunities for its own benefit.

In our case we are currently more interested in Explicit Opponent Modeling, which is a more static approach and is aimed at best representing the current opponent strategies. Here the use of continuous state- and action spaces is most interesting.

10.3.4 Browning2004 - STP: Skills, Tactics and Plays for multi-robot control in adversarial environments. [12]

In this paper the mechanism of the STP is explained. Our team implemented STP in 2019 and so our 2019 log-files will include data related to this approach, developed for the Small Size League (SSL) as part of the CMU robot team. A range of articles was written on this subject and many improvements have been made to the approach since this paper. The main point is that most of the 'logic' in this approach is coded in the PlayBook and the State Machines that implement the ideas. Only some small fragments of this are described in this paper. Also one of our team members (Koning [20]) has written a paper about the applicability of STP to our robots. A similar paper has been written about the CAMBADA approach, which is comparable to STP. (Schouten [86]) Both approaches use a centralized approach. In the SSL there is a central computer, in the CAMBADA approach, the central Coach computer is used.

10.3.5 Purmehr2010 - An overview on Opponent Modeling in RoboCup Soccer Simulation 2D. [76]

In this overview article the most important approaches for Opponent Modeling used in the Simulation League are described. There are two main streams, sometimes combined; Team Strategy Classification and Agent Action Classification. What is looked for is mostly

Formation, Offense and Defense strategy and Pass graphs. The most important work is based on the Coach Competitions, where a single agent controls an entire team, based on accurate data from the simulator. Information for individual agents is injected with noise in the simulator and makes using this data less useful. The Coach information is noise-free and is therefore preferred for these purposes. Riley and Veloso use a windowing approach to classify Game Plays. This uses some 30-37 hand-crafted classes and achieves an accuracy of 40%. Iglesias uses a similar approach to classify hand-crafted features into a so-called trie structure (pass1to2->dribble2->pass2to10->goal10). Steffens uses 2-13 hand-crafted features in Feature Based Declarative Opponent Modeling in descriptions like "Opponent often does long pass along the left wing" Later on this work uses Case Based Reasoning. To classify agent actions there generally are not enough examples for specific cases, which is increased by fuzzyfication. Most work tries to recognize simple behaviors like intercept, pass, dribble. Another approach uses triangular graphs to create topological structures. In another work an Expert System is used to model the opponent game using heuristic rules. To find team strategies some approaches classify the team formation, like 5:2:3 to describe how the team behaves. In Ayanegui-Santiago such formations are recognized by a neural network. Ledezma classifies individual agent actions into a hierarchical schema. In later work this is stored in a decision tree. The general conclusion of this overview is that opponent modeling could be very useful but that much more research is needed.

10.3.6 Koning2017 - Skills, Tactics and Plays for Distributed multi-robot control in Adversarial environments [20]

This paper explains how STP can be used in the TURTLE environment, where control is distributed. CAMBADA solves this problem by using a central Coach computer. In this proposal all robots take a vote for a certain play and the majority vote is communicated to all robots. Like in the original STP paper, we need to define our own Playbook and Tactics, while our existing skills will be used. Tactics are mainly based on our current Role Assignment algorithms. There is mention of different Playbooks, depending on the opponents. For our project we need to collect all Plays, Tactics and Skills because these may be the features that a Neural Network needs to search for.

10.3.7 Klooster2018 - Deep Learning for Opponent Action Prediction in Robot Soccer MSL. [105]

In this Paper one of our team members developed a Convnet that learns 3 different situations:

1. RefBox situation classification (Kickoff, ThrowIn, FreeKick, Corner, Penalty, GoalKick, DroppedBall)
2. Occurred actions, divided into Pass, Shot on Goal and Other actions
3. Future Opponent Actions.

This last one is the main goal of the work and is achieved by creating images with movement traces for all robots and the ball, in washed-out color depending on how long ago this happened for a period of up to 5 seconds (100 time steps).

This 2-layer network achieved a 98% accuracy on the RefBox task, 82% on the Actions task and 72% on the Opponent prediction task. It uses a stacked input frame approach, contrary to our approach with a Recurrent Network. See the work of Hausknecht and Stone [40] for a description of the differences.

10.3.8 Schouten2018 - The Implementation of Setplays to the RoboCup middle-size League [86]

This paper describes a study made at CAMBADA in 2017/18. It describe the STP equivalent, developed by CAMBADA that is closely integrated with the RTDB, which we are also using in the TURTLEs. It also is a three layered approach, consisting of Setplays, Roles and Behaviors. Condition checking is different from the STP approach and it seems that in some cases (like with a pass), the play is started or continued, even when the conditions are not met, since they are not checked during execution. Whereas in STP the selection of a Play is done by each player and communicated to all other team-members, here this role is assumed by the Coach, a computer that is monitoring all team player activities and decides which roles are being assigned. The system keeps track of when setplays have been selected and if they were successful, so they can be used to improve future usage. There is a nice section on selecting the right angle to shoot at the goal, based on the number of points that are the best option to score. Something that seems a large disadvantage is that different field sizes require the definition or adaptation of set-plays, since location information is absolute. One nice thing is the interactive definition of Setplays using the SPlanner2 GUI. It

allows definition and modification of Setplays and is directly coupled with the agent software to test if a Set-play is executable and is tested before submitting it. We also developed an interactive Setplay editor, based on the KNVB Futsal training manual. We have adopted the term Set Pieces, which is the terminology used in the Futsal documentation of KNVB, FIFA and UEFA. We also have similar information about the training manuals from FIFA and UEFA. We intend to use this as test-cases to see how the various teams handle these standard Set Pieces. Case Based Reasoning is mentioned several times in this paper and also in others and we need to look into this. Also check their TDPs from 2016 and 2017.

10.3.9 Lith2018 - A Minimalistic Approach to Identify and Localize Robots in RoboCup MSL Soccer[104]

In this prelude to our current work, digital images from the omnidirectional camera's of our TURTLE robots were used to recognize robots from our own team and the opponents. These images were then used to create an Agent Situation Imagev(ASI) as an occupancy grid, spanning a circular area with a radius of 6 meters. These images, together with a world model in which all robots are depicted are to be used in the system that is described in this paper. The localization work is not yet finished and still has problems when robots have identical colors, like the blue status lights, that some robots have.

10.3.10 Meerhoff2019 - Mining Soccer and Data: and Discovering patterns of tactics in tracking[67]

This unpublished paper represents the work of the Leiden Institute of Advanced Computer Studies (LIACS) where data-mining techniques are used to find statistical information that identifies weak or strong game situations. This is done by analyzing the positions of all players of a team and find configurations like the spread of the team, the shape of the team layout, distances to peer robots, the ball, the goal etc. The analysis of digital or hand-collected position data during a game generates a large collection of table rows, that are correlated with known game situations to find statistically significant information. This approach was used on our own data to find out if there is any overlap between this and our own work. The results are to be included in our paper.

10.4. Opponent Modeling

10.4.1 Iglesias2009 - The Winning Advantage: Using Opponent Models in Robot Soccer [45]

This paper describes the RoboCup opponent modeling challenge. Reference is made to Iglesias and Ledezma on Caos coach 2006 simulation team in which the basis for this work was created. Kuhlman and Stone (2009) characterize team behavior as a set of features, calculated from statistics collected during game observations. Fathzadeh (2008) describes a rule-based expert system to classify games. Iglesias and Ledezma (2008) presented the CAOS system to model and recognize team behavior. It classifies actions as one of *Pass, Dribble, Intercept, Steal, Goals Missed, Foul and Hold*. They developed a visual tool called Viena in which they create patterns, stored in the CAOS Pattern Library. These patterns are expressed in a language called CLang with descriptions like: *(bowner opp(5) (do our(4) (intercept) (pass-Opp6-Opp8))*

10.4.2 Trevizan2010 - Learning Opponent's Strategies in the RoboCup Small Size League [98]

The CMUDragons team introduces a technique to classify the opponent defense strategy by defining a number of features, like distance from the ball, distance to the goal, distance between the robots and collect these in matrices, organized by episodes. The number of goals scored is taken as a measure to define how well the defense is against the CMDragons. They refer to other work, where decision trees and payoff matrices, using the prisoner's dilemma as examples. Another approach uses conditional random fields to classify the roles of robots. references 1, 14 and 15 may be relevant. It uses a statistical approach to compare strategies by using the Frobenius norm. This might be comparable to the metrics used by LIACS.

10.4.3 Mnih2013 - Playing Atari with Deep Inforcement Learning [68]

This is the seminal DeepMind paper, describing how Atari games were learned from watching screen images of games like Pong and others, learning actions from a convolutional neural net (ConvNet). Where most earlier approaches were based on hand-crafted features and policy representations, this work used raw screen images to learn to play a variety of Atari games with the same architecture, dubbed Deep Q-Learning Networks (DQN). The basis for this work is formed by the Bellman equation, which was extended

to use an experience replay buffer, from which random samples are taken to learn which sequences of actions lead to the highest future reward. it uses four stacked input frames to represent a sequence and learns the Q function in a separate network for each of the discrete actions. These Q values are used in the main network to learn the best action to take, given a certain state. By making the update frequency of the Q-Learning network much lower than that of the main network, a stable learning situation is created, that makes this approach feasible. Many variations to this approach have been tried later on, which improved on this approach, eventually leading to the Alpha-Go and Alpha-Zero systems.

10.4.4 Mnih1602a - Asynchronous methods for Deep Reinforcement Learning [?]

This paper describes asynchronous versions of the four most used RL algorithms. It replaces the Experience Replay buffer by a parallelization approach that leads to faster and better search of policies. Several terms keep coming back in this literature, that need further exploration and explanation:

1. on-policy and off-policy learning
2. Value-based and policy-based RL
3. model-based and model-free learning
4. one-step and n-step Q-learning
5. RSG, RMSProp and RMSProp with shared statistics
6. Discrete actions and Continuous actions

10.4.5 He2016 - Opponent Modeling in Deep Reinforcement Learning [41]

This work describes a DQN approach with a (very) simple soccer simulation, in which implicit modeling is used to learn behaviors based on opponent actions. It tests two strategies: Concatenation and Mixture-Of-Experts (MOE) in which a separate model learns the opponent strategies as a hidden property. So, although the strategy is learned, it is not stored as explicit knowledge. The simulation is based on 2 players on a 6x9 grid, using limited hand-crafted strategies, that the DQN network has to learn. The Q values that are learned are the best actions, based on the game situation in relation to the opponent strategy. It makes references to earlier work in which a separate opponent model is created from Davidson 1999, Ganzfried & Sandholm 2011 and Schadd et al 2007. Need to check these references as well. Related work in Explicit Opponent modeling is found in Uther & Veloso 2003 (decision Trees), Foerster et al 2016 (Neural Network), Hausknecht & Stone 2015 (DQN). There is also a reference to work on DRQN from Eigen et al (2014)

10.4.6 Lillicrap2016 - Continuous Control with Deep Reinforcement Learning [60]

In DQN [68] a number of Atari games were learned by a Deep Q-Learning network, using pixels data as input. This work describes an extension of DQN into a new approach called Deep Deterministic Policy Gradients (DDPG). In this approach the existing DQN is extended to allow learning continuous functions, so that instead of discrete actions now values like forces and directions can be learned. The DQN network is extended with actor-critics that use an Experience Replay buffer (ER) to learn Policy Gradients instead of a Policy. Previously it was thought that this was impossible, since the solution space becomes intractable. Using Batch Normalization in all layers.combined with an Actor Critic made this approach possible. The Target network is not updated regularly by copying all weights, but uses soft target updates. They are also constrained to values between 0 and 1 by batch normalization. They also report that learning from pixels is just as fast as learning from underlying state description values.

10.4.7 Steffens2004 - Feature-Based Declarative Opponent-Modelling (8) - [92]

this paper describes an early approach to define and learn an opponent model, based on specific features in the input of log files. It is based on the Simulation League Coach Server, which keeps information about played games. Its aim is to compare tactics of various teams and to create counter-strategies. The approach is named Feature Based Declarative Object Model (FBDOM) It uses the Standard Coaching Language (CLang) to define rules and actions and learns two distinct models using a Situation-detector and an Action-detector. It does not represent all possible situatins, just a subset in the form of if-then rules. The models were built from just one or two games per team and proved to be distinct for most teams and concentrated on the offensive strategy only. In some cases the strategies of teams seemed very similar. testing of the models was done by building a counter-strategy and then play against this to see how many more goals this strategy scored. It was also tested against a random strategy to make sure the counter-strategy was effec-

tive. Only in the case of teams with similar strategies no significant differences were found. The main point in this approach is the use of a rule-based architecture, which enabled the system to provide explanations for the exhibited behavior. That is very important for our approach.

10.4.8 Ledezma2002 - Predicting Opponent Actions in the RoboSoccer (8) - [54]

In this work a model is built, using a classifier, that identifies the actions taken, given some situations and a separate data structure that quantifies the action, for instance with a speed or an angle. It uses a rule-modeler (c4.5 Quinlan1993) to learn the classification and a regression tree (M5 Quinlan1993) to represent the strategy. Here too this architecture is selected to allow understanding of the generated rules. The system consists of an Action Modeler, a Rule Learner and a Parameter Learner. These are input to a reasoning module, that uses the rules to reason about the strategy and to provide explanations. The number of attributes and action classes is relevant for the recognition performance, which started out as 45% and with additional reconfiguration was reduced from 140 attributes to 32 and from 31 classes to 7, resulting in a 72% recognition capability. By creating an attribute level paired with a class level a hierarchical structure developed, leading to the name of Hierarchical Learning for this approach.

10.5. Strategy and Tactics

In this subsection papers are included that concentrate on Strategic and Tactical methods to control a team.

10.5.1 Martinovic2010 - Robot Soccer - Strategy Description and Game Analysis [65]

In this paper a very simple 2-player soccer simulator is used to teach a DQN to recognize action patterns and learn responses to this. In this case the system learns the implicit strategy of the opponent in an attempt to directly learn to react in response to the opponent strategy. It creates three matrices, one for the ball, one for the opponent and one for the own team. These matrices are like an occupancy grid, in which a 1 represents the presence of an object. The system maintains two different world models, a strategy level and an abstract level. The strategy level is a coarse representation that compresses the field to small dimensions, so that patterns become dense and more similar. The abstract model has real world coordinates and is used to translate between the strategy level and reality. The most important thing is to match different patterns, which is done in a way that is not described very clearly in this paper.

10.5.2 Hausknecht2016 - Deep Reinforcement Learning in Parameterized Action Space [39]

Most DQN approaches learn a set of discrete actions. However in soccer the learned actions are in most cases instances where speed and direction are important properties that quickly become inaccurate, when reduced to discrete actions. In most cases an action will be represented with a description and a quantitative indication like move-forward with speed 80. So it becomes important to have a network that can learn parameterized actions. Standard DQN networks can only output discrete classes. Therefore Deep Deterministic Policy Gradient (DDPG) networks were developed in Lillicrap 2015 in which actions (policies) are expressed as a gradient. This approach is further expanded into a DQN that outputs both an action and its associated parameters. In this work a simplified version of RoboCup simulation is used, in which an agent learns to shoot at the goal. It learns four actions: Dash, Turn, Tackle and Kick, each associated with one or two parameter, giving: Dash(power, direction), Turn(direction), Tackle(direction) and Kick(power, direction). In order to learn, rewards are given, but because scoring a goal results in very sparse rewards, hand-crafted intermediate rewards like Move-to-ball-reward and Kick-to-goal-reward. This reward engineering is necessary to reach convergence. Several earlier attempts have been made to realize modeling action spaces like, decaying traces, re-currency and double Q-learning. Actor/critic systems first decoupled action learning from values functions. It uses a target network to learn Q values separated from the actions.

One problem to be solved by this new approach is to keep the values learned within the bounds of acceptable values, like -180 to 180 degrees for direction and 0 to 100 for speed. This work investigates three possible approaches to this problem by modifying the learned values when they crossed the boundaries: zeroing, squashing and inverting the gradients. It proves that only inverting the gradients works as desired. However this approach is quite process intensive. It takes three days on an NVidia Titan X GPU. It resulted in performing better than the best hand-crafted agents.

10.5.3 Xiong2018 - Parameterized Deep Q-Networks Learning: Reinforcement Learning with Discrete-Continuous Hybrid Action Space [117]

This work extends the work of Hausknecht2016 and introduces a hybrid between DQN and DDPG in a new approach called Parameterized DQN (P-DQN). Instead of a double network in which a critic learns the Q-Values and the target network learns the Actions and associated parameters, this network directly learns the actions in one network and the parameter values in another network. It used an asynchronous approach to speed up the learning process.

10.5.4 Biswas2014 - Opponent-Driven Planning and Execution for Pass, Attack, and Defense in a Multi-Robot Soccer Team [11]

This paper describes new methods in the CMUDragons SSL team architecture namely:

1. Pass-Ahead. Instead of passing to the location of a known teammate, a pass is made to a future location from where the probability of making a direct kick at the goal is the highest.
2. Coerce and Attack. This approach detects opponent team roles and attempts to make them deviate from their plan to create openings that allow a better attack on the opponent.
3. Threat-based defense. Tries to anticipate opponent passes and tries to block them.

In the last two cases, information about the roles of the opponents is required. The work is based on STP, which consists mainly on human-generated plans. To calculate the best passing positions, the 6x4 meter field is represented by a 6x4 grid in which potential fields are used to find the best cells. Passes are generally done as flat passes, a chip pass (lob shot) or a yanking pass, which a backspin effect is given to the ball. For the Coerce and Attack plan, an analysis is made of the opponent roles. They are classified as robot-following or ball-following behaviors, further subdivided into Primary defenders (ball-following), Mark (robot-following) and Wall (also robot-following). The Coerce And Attack Planner (CAP) calculates the probability of each opponent being one of the three defenders by generating an area around each robot for each of the roles and uses this information to estimate the probability of the robot playing that particular role. To analyze the performance, two scores are calculated for every game played, the Offense Ratio and Attack Ratio. Offense ratio is the amount of time that the ball was on the opponent's half, against being on our own half. The Attack ratio is the number of times the team attempted to shoot at the goal, against that of the opponents. We could calculate the same ratios for all our logged games as well. Important references are Ros and Veloso (2006) and Browning and Veloso (2005), Riley and Veloso (2002), Han and Veloso (2000) and Erdogan and Veloso (2010).

10.6. Papers on DQN, DRQN and LSTM

For the following topics, include some papers and references:

1. Hierarchical Learning. Especially with the Game State Transition Diagram
2. DQN and A3C/A2C and newer approaches
3. DDQN, NAF, CEM, DDDQN and PPO
4. Print tutorial on DQN and find papers
5. LSTM and GRU
6. Residual Networks
7. Monte Carlo Tree Search

10.6.1 Hausknecht2015 - Deep Recurrent Q-Learning for Partially Observable MDPs [40]

This paper is a further study on the Atari games. It compares using stacked input frames in a Convnet against using an LSTM in the last layer. It concludes that with many Atari games, stacking the last 4 frames, a POMDP becomes an MDP. Adding an LSTM in the last layer turns a DQN into an DRQN and has similar performance, but seems more robust. This approach uses a target network to stabilize predictions of the Q Value and actually is the precursor to the later Dueling and Double dueling networks that we are using. This work also uses visualization techniques like we used in the Classification of MSL robots version. We will use the same techniques in our work on Explicit Opponent Modeling.

10.6.2 Hasselt1509 - DDQN Deep Reinforcement Learning with Double Q-Learning (8) [101]

The Double Q-Learning approach is an modification of DQN, to handle the generally large overestimation of Q-values that DQN suffers from for several domains. It solves this problem by splitting the output of the last convolutional layer into two separate streams, one for the value and another one for the policy. The more actions a domain has, the larger the overestimation

will be. There are actually two networks, a main network, that learns the functions and a target network, that is updated at a lower frequency than the main network. The main network is used to learn the action (or policy), while the target network is used to learn the value. By decoupling both networks, they will have different error rates and will therefore compensate the overestimation.

10.6.3 Osband1602 - Deep Exploration via Bootstrapped DQN - (8) - [74]

This paper describes Bootstrapped DQN, an extension to standard DQN and also applicable to DDQN, which leads to faster learning and more generalizing function exploration. It uses a number of so-called heads, included in the network, directly after the convolutional layers, that evaluate multiple strategies in parallel. This allows the network perform deep exploration and finds better strategies more quickly. The paper explores other approaches like Thompson DQN and Ensemble DQN. It also mentions other exploration algorithms like PSRL and RLSVL.

10.6.4 Gu1603 - NAF Continuous Deep Q-Learning with Model-based Acceleration (8) [35]

Most Q-Learning methods learn actions or policies as classifications. That is fine for actions like moving forward, backward or sideways. But for functions like kicking the ball, we need a more refined direction, preferably in degrees and additionally a power component as well. Continuous actions are more difficult to learn, given its potentially large action space, which introduces large uncertainties. Deep Deterministic Policy Gradients (DDPG) is able to learn continuous, parameterized action spaces, but are complicated. Normalized Advantage Functions (NAF) is a much simpler approach with comparable results and also learns faster. Although this is a model-free approach, it proposes to build an internal model to facilitate the fast generation of experience-replay examples. In our case that will not be necessary, because we will not be employing exploration and therefore have no experience replay buffer. The examples from the log files are sufficient and actually take over the function of such a model. This approach uses a single network that outputs both the policy and the value. The approach is related to Dueling networks (Wang1511) Like other networks, the value that is learned is the advantage, which is an indication of how good it is to be in a particular state. This function is normalized during training to get a gradient, that can be scaled back to a paramterized value and serves as the Q-Value.

10.6.5 Szita2006 - CEM Learning Tetris Using The Noisy Cross-Entropy Method (5) [96]

All neural networks use an optimizer, to take the loss of each sample and transform that in a value that is used in the back propagation step. Together with the learning rate, this determines not only the speed of learning, but also the amount of overshoot that can occur during training. We are using the ADAM optimizer in most cases, but there are more optimizing strategies. Cross-Entropy is also a popular optimizer. The Noisy Cross Entropy Method, described in this paper claims to be an order or magnitude faster than the standard Cross Entropy Method and we will experiment with it.

10.6.6 Wang1511 - DDDQN Dueling Network Architectures for Deep Reinforcement (8) Learning [114]

DQN and the later Double DQN use two networks, a main network and a target network to learn the Value and Advantage functions. In this approach a single network splits its output stream into a separate Value and Advantage stream and uses a function to combine both into a Q-Value with a state-action pair. The approach is model-free and off-policy. The function that combines the Advantage and Value back into a Q-Value requires good considerations, that are explained well in this paper. The approach described here, can also be applied to the Double Q-Learning Network, resulting is a simpler and more reliable model.

10.6.7 Schulman1707 - PPO Proximal Policy Optimization Algorithms (8) [87]

This paper proposes a new method to calculate Policy Gradients, which are used in networks that learn policies in continuous domains. We have already seen DDPG and NAF, but in this work PPO is compared to Trust Region Policy Optimization (TRPO) CEM (Szita2006) and A2C/A3C. It is a replacement for standard Policy Gradient Algorithms in other approaches and are simpler to implement. It could be interesting for our approach to a continuous parameterized application.

10.6.8 Wu2019 - A Comprehensive and Survey on Graph and Neural and Networks [116]

Almost all work on recognizing patterns and geometric trajectories is based on Concolutional Neural Networks. Although we have detailed positional data about X,Y,Z positions and accelerations, this data is converted into a grid-like image and then fed into a neural network. The questions is if there does not exist a type of network that is using the positional data directlu to learn to recognize shapes or trajectories.

This work is exactly that. and is a survey on so-called Graph Neural Networks. This study gives an overview and categorizes the various Graph Network approaches as Convolutional Graph Networks, Recurrent Graph Networks and AutoEncoder Graph Networks, so basically the same as existing Neural Networks but then applied to graphs.

The curious thing to the entire enterprise however, is the representation of graphs as input to the Neural Network. Instead of a Graph representation, the structures are converted to adjacent cells in a grid, resulting in images, just as we were already using. The conclusion is that at this time there does not seem to exist a direct analog representation to graphs or trajectories to be used in Neural Network. So an image or grid representation is still the dominant technology.

10.6.9 Schaul2016 - Prioritized Experience Replay

Paper is read and annotated but must still be included here.

10.7. Recurrent Networks

10.7.1 Arulkumaran2019 - AlphaStar: An Evolutionary Computation Perspective [6]

(Copy of the abstract) AlphaStar draws on many areas of AI research, including deep learning, reinforcement learning, game theory, and evolutionary computation (EC). In this paper we analyze AlphaStar primarily through the lens of EC, presenting a new look at the system and relating it to many concepts in the field. We highlight some of its most interesting aspects—the use of Lamarckian evolution, competitive co-evolution, and quality diversity. In doing so, we hope to provide a bridge between the wider EC community and one of the most significant AI systems developed in recent times.

10.7.2 Karpathy2015 - The Unreasonable Effectiveness of Recurrent Neural Networks [49]

In this excellent blogpost examples are given of the application of a Recurrent Neural Network (RNN) that analyses single-character documents and draws conclusions from this, using a simple network. The RNN has a hidden internal state, that is communicated to each time-step and is therefore capable of learning the probability of certain actions occurring after earlier actions. In order to do this, the network has one or more Lon Short-Term Memory (LSTM) cells, that do the actual learning of the sequences. It uses Stochastic Gradient Descent (SGD) and an optimizer like RMSProp or Adam.

Many examples are mentioned, like Video Classification,Image Captioning and Question-Answering. The first applications were DeepMind's Neural Turing Machines. A very promising approach are Attention Networks, that are now becoming State-of-the-Art.

10.7.3 Olah2015 - Understanding LSTM Networks [73]

Long Short-Term Memory networks allow a Recurrent Neural Network (RNN) to learn long-term dependencies in time series. They were introduced by Hochreiter and Schmidhuber (1997). This blogpost explains how they work. The important ingredient are the so-called gates, that control what information is processed by the cell. An LSTM has three gates; Forget, Input and Output. Forget determines how much of the previous time steps is passed through the cell. The Input gate determines which new information is stored inside the cell. The Output gate determines how much is sent to the cell output.

There are many variants to this architecture, like the peephole connections or combined forget and input gates. A simpler version of the LSTM is the Gated Recurrent Unit (GRU) which has a single update gate and merges the cell state and hidden state. This results in a simpler and faster cell, that is comparable but not as flexible, but gaining popularity. Here too the conclusion is that Attention Networks are the way of the future.

10.7.4 Culurciello2018 - The fall of RNN / LSTM [19]

Both previous blogposts are about LSTMs and conclude that Attention Networks are the wave of the future. This one concludes that LSTMs are old hat

and that the only way forward is with Attention Networks. The technology is used in Siri, Cortana, Google Assistant and Alexa, so it is used heavily in Natural Language Processing (NLP). One reason that LSTMs became popular is because it solved the so-called Vanishing Gradients Problem, which occurs when very small weights are added to many layers of a neural network. The Stochastic Gradient Descent successively performs dot-product calculations, which often result in weights that become so small, that they get closer to zero with every iteration.

However LSTMs model dependencies on the preceding steps only and therefore cannot deal with dependencies that are farther away in the event chain. For instance it cannot learn to deal with a question mark at the end of a sentence. This can be done with techniques like Conditional Random Fields, but these are not commonly integrated with Recurrent Neural Networks. So in this post the Transformer is featured as a better approach, based on Attention Modules. An even newer approach based on Causal Convolutions outperforms even these Attention Modules. It also refers to a novel approach called Temporal Convolutional Networks (TCN).

The remainder of this post explains Hierarchical Neural Attention Encoders, based on the Attention Module and is similar to a Neural Turing Machine.

10.8. Symbolic Processing

10.8.1 Sales1994- An Approach to Solving the Symbol Grounding Problem: Neural Networks for Object Naming and Retrieval [83]

10.8.2 Harnad1993 - Grounding Symbols in the Analog World with Neural Nets [38]

10.8.3 Zhang1707 - Learning like humans with Deep and Symbolic Networks [122]

We introduce the Deep Symbolic Network (DSN) model, which aims at becoming the white-box version of Deep Neural Networks (DNN). Symbols are connected by links, representing the composition, correlation, causality, or other relationships between them, forming a deep, hierarchical symbolic network structure. The symbols and the links between them are transparent to us, and thus we will know what it has learned or not - which is the key for the security of an AI system. The Deep Symbolic Networks (DSN) model aims at solving the drawback of Deep learning that the obtained models remain black-boxes[3]. Suppose we have already constructed a deep symbolic network as in model with all symbols equipped with their identifying operators. One notices immediately the composition links between the symbols, emerging from the relationships between them. Links by causality, Links by abstraction, Higher order links, Links can also be represented by symbols. Identifying operators, using MNIST as an example. In this conceptual paper, we have introduced a Deep Symbolic Networks (DSN) model, which is inspired by DNN and also comes out from observing the real world - it models the deep, hierarchical structure of the world, with the observations that humans symbolize physical matter, and that singularities isolate symbols and create symbol dictionary naturally for us.

10.8.4 Hohenecker1808 - Ontology Reasoning with Deep Neural Networks [43]

Traditionally, symbolic logic-based methods from the field of knowledge representation and reasoning have been used to equip agents with capabilities that resemble human logical reasoning qualities. In this paper, we employ state-of-the-art methods for training deep neural networks to devise a novel model that is able to learn how to effectively perform logical reasoning in the form of basic ontology reasoning. We derive a novel approach for creating models that are able to learn to reason effectively in a great variety of different scenarios. Interestingly, however, it can be observed that, under certain provisions, even the best reasoning models based on machine learning are still not in a position to compete with their symbolic counterparts. We develop a novel model architecture, called (RRN), which makes use of recent advances in the area of deep neural networks (Bengio, 2009). Most of the KRR formalisms that are used for reasoning today are rooted in symbolic logic, and thus, as mentioned above, employ mathematical proof theory to answer queries about a given problem. Machine learning models are often highly scalable, more resistant to disturbances in the data, and capable of providing predictions even if the formal effort fails. The facts that define such a knowledge graph are usually stated in terms of triples of the form subject, predicate, object and specify either a relation between two individuals and or an individual's subject object membership of a class, in which case refers to an individual, to a special subject predicate membership relation, and object to a class. First it generates vector representations, so-called embeddings, for all individuals that appear in the considered data, and second, it computes predictions for queries solely based on these generated vectors. RRNs are based

on the idea that we can encode all the information that we have about an individual, both specified and inferable, in its embedding. The results presented in this work show that the RRN model is able to learn to effectively reason over diverse ontological knowledge bases, and, in doing so, is the first one to achieve an accuracy that comes very close to the yet unattainable accuracy of symbolic methods, while being distinctly more robust. The RRN is among the very first deep-learning-based approaches to comprehensive ontology reasoning, which is why it is hard to compare to the state-of-the-art of machine learning models for reasoning, whose architectures do not allow for performing the same kind of inferences.

10.8.5 Garcez1905 - Neural-Symbolic Computing: An Effective Methodology for Principled Integration of Machine Learning and Reasoning [24]

Neural-symbolic computing aims at integrating, as foreseen by Valiant, two most fundamental cognitive abilities: the ability to learn from the environment, and the ability to reason from what has been learned. We illustrate the effectiveness of the approach by outlining the main characteristics of the methodology: principled integration of neural learning with symbolic knowledge representation and reasoning allowing for the construction of explainable AI systems. Neural learning and inference under uncertainty may address the brittleness of symbolic systems On the other hand, symbolism provides additional knowledge for learning which may e.g. ameliorate neural network's well-known catastrophic forgetting or difficulty with extrapolating. The Knowledge-Based Artificial Neural Network (KBANN) [49] and the Connectionist inductive learning and logic programming (CILP) [17] systems were some of the most influential models that combine logical reasoning and neural learning. Knowledge representation is the cornerstone of a neural-symbolic system that provides a mapping mechanism between symbolism and connectionism, where logical calculus can be carried out exactly or approximately by a neural network. This way, given a trained neural network, symbolic knowledge can be extracted for explaining and reasoning purposes. The representation approaches can be categorised into three main groups: rule-based, formula-based and embedding. Rule-based Representation, Formula-based Representation, Tensorisation is a class of approaches that embeds first-order logic symbols such as constants, facts and rules into real-valued tensors. Normally, constants are represented as one-hot vectors (first order tensor). Predicates and functions are matrices (second-order tensor) or higher-order tensors. Various attempts have been made to perform reasoning within neural networks, both model-based and theorem proving approaches. In neural-symbolic integration the main focus is the integration of reasoning and learning, so that a model-based approach is preferred. Forward chaining and backward chaining are two popular inference techniques for logic programs and other logical systems. In the case of neural-symbolic systems forward and backward chainings are both in general implemented by feedforward inference. In early work, the demand for solving "black-box" issues of neural networks has motivated a number of rules extraction methods. Most of them are discussed in the surveys [1, 24, 55]. These attempts were to search for logic rules from a trained network based on four criteria: (a) accuracy, (b) fidelity, (c) consistency and (d) comprehensibility [1]. In this paper, we highlighted the key ideas and principles of neural-symbolic computing. In order to do so, we illustrated the main methodological approaches which allow for the integration of effective neural learning with sound symbolic-based, knowledge representation and reasoning methods.

10.8.6 Liao2017 - Object-Oriented Deep Learning [59]

This research that aims at converting neural networks, a class of distributed, connectionist, sub-symbolic models into a symbolic level with the ultimate goal of achieving AI interpretability. wW would like to see to what extent we can convert neural networks, sub-symbolic models, into a symbolic level. In our framework, we aim at making progress on explicitly representing all such symbolic concepts, leading to what we call "object-oriented" deep learning (OODL) How to learn disentangled representations from data using distributed code is however a challenging task. Variational Auto-encoder [1] can do it to some extent, but cannot so far scale to real-world tasks. We call these approaches "neural disentanglement". When an object is detected, its properties (e.g., pose, size, position) are predicted at the same time. Same things happen in every intermediate layer of the network, so that one has full knowledge of the properties of any detected object and its parts, recursively throughout all layers The input to a general OO layer is a (1-D) list of objects, their "signatures" and their properties. A "signature" is a (learned) vector that identifies the object and is invariant to the properties. The properties include features

that are useful for performing intelligent tasks. They include but are not limited to positions (x,y), poses (scale, rotation, etc.), "objectness" (probability of being an object) pointers to its parts (for feature binding), and physical states (e.g., volume, mass, velocity, acceleration, etc.). So far we have experimented positions, poses and objectness (and part-whole relationships in some experiments). The output of the general OO layer is another 1-D list of objects and their properties. A Predicting/Voting OO Layer, A Binding OO Layer. We tried our model on the standard CIFAR-10 dataset.

10.8.7 Mao2019 - The Neuro-Symbolic Concept Learner: Interpreting Scenes, Words and Sentences From Natural Supervision [?]

Our model learns by simply looking at images and reading paired questions and answers. Our model builds an object-based scene representation and translates sentences into executable, symbolic programs. NS-CL has three modules: a neural-based perception module that extracts object-level representations from the scene, a visually-grounded semantic parser for translating questions into executable programs, and a symbolic program executor that reads out the perceptual representation of objects, classifies their attributes/relations, and executes the program to obtain an answer. Visual perception, Concept quantization, DSL and semantic parsing, Quasi-symbolic program execution. Optimization objective, Curriculum visual concept learning. We presented a method that jointly learns visual concepts, words, and semantic parsing of sentences from natural supervision. The proposed framework, NS-CL, learns by looking at images and reading paired questions and answers, without any explicit supervision such as class labels for objects. Our model learns visual concepts with remarkable accuracy. Based upon the learned concepts, our model achieves good results on question answering, and more importantly, generalizes well to new visual compositions, new visual concepts, and new domain specific languages.

10.8.8 Garnelo2019 - Reconciling deep learning with symbolic artificial intelligence: representing objects and relations [32]

A 'vanilla' autoencoder is liable to produce entangled representations (Figure 1b top). But a variational autoencoder with a suitably tuned loss function will generate disentangled representations. In a vanilla autoencoder, each variable in the latent representation is a single number, while in a variational autoencoder each variable is represented by its mean and its variance. A deep neural network capable of learning a mapping from its input data to a multi-object disentangled representation would be a significant step towards a deep learning system that acquires and uses grounded symbolic representations, with all the potential advantages that entails. It was subsequently shown that an architecture based on relation networks could be successfully applied to certain analogical reasoning tasks [39]. A related means of discovering and using relational information is a so-called self-attention mechanism [5]. One way to understand self-attention (which has little to do with attention in the conventional sense) is by comparison to relation networks [41]. We have focused on the question of how a deep network can learn to acquire and use compositional representations whose elements are objects and relations. These are key features of symbolic representations, and the work discussed exemplifies what can be achieved today. But we are still a long way from a satisfying synthesis. Simulation, in the sense of using a model to predict how states unfold over time, is one form of such processing, and a number of recent works describe deep neural networks that carry out such simulations [46,47]

10.8.9 Bennetot1909 - Towards Explainable Neural-Symbolic Visual Reasoning [9]

As opaque machine learning models are increasingly being employed to make important predictions in critical environments, the danger is to create and use decisions that are not justifiable or legitimate. Interpretability is the degree to which an observer can understand the cause of a decision. A last option to achieve an explanation of the model decision would be to directly populate the KB from the data. This would allow to provide an explanation in natural language directly from the black box, emphasizing in the meantime the model's reasoning errors and highlighting possible bias in the dataset or model. We propose creating the reasoning-facilitating KB by performing word-embedding on the black box model labels in order to determine which words are particularly exposed to a risk of errors due to learning priors or biased data collection. Interchangeable words are more likely to be victims of overuse of context [Zhao et al., 2017]. The use of a symbolic basis with a neural network can provide explanations close to the functioning of human reasoning while maintaining the state-of-the-art performance at the same time. As the user or expert external knowledge does not interfere the predictions

in the explanation process, it constitutes a truly explainable model that is faithful to communicate the reasoning behind its output decisions.

10.8.10 Garnelo1609a - Towards Deep Symbolic Reinforcement Learning [?]

In this paper, we propose an end-to-end reinforcement learning architecture comprising a neural back end and a symbolic front end with the potential to overcome each of these shortcomings. They inherit from deep learning the need for very large training sets, which entails that they learn very slowly. They are strictly reactive, meaning that they do not use high-level processes such as planning, causal reasoning, or analogical reasoning to fully exploit the statistical regularities present in the training data. We propose a novel reinforcement learning architecture that addresses all of these issues at once in a principled way by combining neural network learning with aspects of classical symbolic AI. The use of language-like propositional representations to encode knowledge is an important aspect. A major obstacle here is the symbol grounding problem [18, 19]. Hand-crafted representations cannot capture the rich statistics of real world perceptual data. The hybrid neural symbolic reinforcement learning architecture we propose relies on a deep learning solution to the symbol grounding problem. It comprises a deep neural network back end, whose job is to transform raw perceptual data into a symbolic representation, which is fed to a symbolic front end whose task is action selection. The neural back end must learn a compositionally-structured compressed representation of the raw perceptual data, while the symbolic front end must learn a mapping from the resulting symbolic representation to actions that maximise expected reward over time. Conceptual abstraction, Compositional structure, Common sense priors, Causal reasoning. Low-level symbol generation, Object detection and characterisation, Representation building, Spatial proximity, Type transitions, Neighbourhood, Symbolic interactions and dynamics. We have proposed a hybrid neural-symbolic, end-to-end reinforcement learning architecture, and claimed that it addresses a number of drawbacks inherent in the current generation of DRL systems. Moreover, even though the system is only a preliminary proof-of-concept with many limitations (to be discussed shortly), it dramatically outperforms DQN on the most difficult game variant.

10.8.11 Rabinowitz2018 - Machine Theory of Mind - [?]

This article describes an approach to implement a system to learn desires, beliefs and intentions based on observations of sequences of actions. It is also an attempt to make neural networks more explainable. The goal is to make agent human-interpret-able. They construct an observer, who in each episode gets access to a set of behavioral traces of an agent. In this way it finds similarities in behavior between different agents. The system learns to infer the goals of agents. ToMnet consists of three modules: a character net, a mental state net, and a prediction net. The character net parses past episode trajectories, to find out specific actions agents take. The mental state network reasons about the agent during the current episode (i.e. infer its mental state. By learning the agent's policy, it finds out the agent's goal by observing the rewards given. If the agent sought the closest object in a past episode, then the ToMnet was more cautious about whether it would seek the same object again on a new episode. They also note that: hand-crafting a Machine Theory of Mind to parse behaviour based on our human knowledge (e.g. Baker et al., 2011; Nakahashi et al., 2016; Baker et al., 2017; Lake et al., 2017) becomes increasingly intractable. Therefore they learn it with ML, but their examples are form simple grid-world experiments only. The networks are LSTM Q-Learning networks.

10.9. Attention Models

In this section we investigate Hierarchical Systems, the concept of Attention in Neural Networks and Symbolic Processing. We are looking for a way to combine Neural Networks with reasoning mechanisms like we had in Expert Systems as in Good Old Fashioned AI (GOFAI). The Attention Mechanism is especially interesting, since it directs a system's efforts towards information that is semantically or logically related to its inputs.

10.9.1 Marcus2017 - Deep Learning: A Critical Appraisal [64]

Current Deep Learning networks suffer from a number of fundamental problems, as I have argued before. In search for a way to combine the Symbol Level as used in GOFAI and Expert Systems, and Deep Neural Networks, this article sums up these problems and shows ways in which these might be handled in the

future. This work on recognizing opponent team behavior is one attempt at this as well. The following problems are identified:

1. Deep Learning is going to meet a wall. Chollet (2017), Marcus (2012) and Hinton and Sabour (2017) are hinting at this. Deep Learning is not a general solution to AI.
2. DL is very data hungry. We need large numbers of examples to learn. They are not capable of using simple logical inferences as input to learn things that are obvious to us, nor can they generalize beyond the given examples.
3. Transfer Learning is almost impossible. Current models can only transfer features that are similar enough, but may still lead to absurd errors, like recognizing a traffic sign as a weapon.
4. DL vannot represent hierarchical structures. Only adjacency (like in sequences) and positional shift (like in convolutions) can be handled. drawing inferences about intention or causality are beyond current technology. Tehy only work with correlation.
5. DL is a 'Black Box' and cannot explain its reasoning. We need to devise 'White Box' system that contain explanation facilities, like Expert Systems used to have. They work on the basis of labeled examples, but have no way of grounding these labels.
6. When something changes in the outside world, new concepts are needed or structural changes occur, the training must be done from scratch. Learned layers become unusable. Learning new examples even causes older training to be forgotten.
7. There is too much hype and it is not clear to potential users, where the limits are. Many examples suggest that many more types of problems can be solved, which is not true. Only classification problems in an unchanging world can be handled very well. For the remaining problems we need different solutions, that are not yet available. Pieter Thiel's remark: "We wanted flying cars, instead we got 140 characters."
8. We need to concentrate on Unsupervised Learning, where new classes and categorization is done automatically, while solving the Grounding Problem. Action Sequences might be a good example for this, since the relationship between actions and images can be made explicit. This ties in with Symbol manipulation. Action symbols must be grounded in our world and should correspond to the tokens, used by an AI system. This is attempted in systems like NeuroSymbolic Modeling and Neural Programming. Embodied AI is another topic that might be fruitful. So we need a way to combine Reasoning with Concepts and Neural Inferencing.

10.9.2 Qianli2017 - Object-Oriented Deep Learning [58]

In this work a new approach to working with symbols is introduces. Instead of using Tensors and Convolutions, object structures are used as the basic building blocks. In each layer a number of objects is described with properties. So instead of concentrating on features in a convolutional network, here objects and their properties are used to build layers of increasing generalization. On the lowest level random target points are selected, called Voting, to mimic convolutions and to create the first level primitive objects. Each object has a coordinate and orientation along with other properties. In higher layers objects are grouped into larger objects, called Binding to form grounded symbols. Other than convolutions, here the position and orientation is maintained, related to the original picture, creating a hierarchy of generalized objects.

With this method, objects have feature bindings, property associations and signal groupings. These properties relate to the object and not to features or sub-symbolic parts in a layer of a convolutional neural network. The resulting Object Oriented network differs from a Feature Oriented network, in that is maintains a direct relationship with the object and its location in the image.

This approach realizes a disentangled representation of objects in an image, which is very hard to do in Convolutional Neural Network. Currently this is also done by Variational Auto-Encoders, but they do not scale to real-world problems. The Object Oriented approach is a better candidate for such problems.

10.9.3 ICLR2018 - Composable Planning with Attributes [44]

In this work actions of an agent are decomposed into high-level abstractions of a task into elements, called Attributes. Each action is analyzed according to the effect it has on the environment. Recording these cause and effect relationships in Attributes makes it possible, to learn a path to a solution, comparable to a reasoning chain in situation-action pairs.

10.9.4 Garnelo1609 - Towards Deep Symbolic Reinforcement Learning

10.9.5 Zhang1707 - Learning like humans

10.9.6 Liao2018 - Symbolic Reasoning (Symbolic AI) and Machine Learning [57]

10.9.7 Vaswani1706 - Attention Is All You Need [106] (9)

With this paper started the idea of Attention. It introduces the Encoder and Decoder architecture, later integrated into the Transformer. It is an approach to sequence modeling, also known as Seq2Seq models. Developed mainly to overcome problems in Natural Language Processing (NLP) it concentrated on finding long-distance dependencies in pieces of text. Existing LSTM models worked well on near-distance dependencies only. Longer distances resulted in increased memory usage and longer processing times. With a Query mechanism that accesses key-value pairs that learn dependencies, a much better mechanism was achieved. The tow main concepts are self-attention and multi-headed attention. Self-attention finds dependencies within the same sentence. multi-headed dependencies look for relationships with longer distances in other sentences or more distant parts of a text. Whereas linear models or convolutional models look for mappings between inputs and given labels, attention learns mappings between dependencies and is much less data-hungry. Instead of using a RNN, this approach uses standard feed-forward networks to learn mappings between dependencies and could work equally well on strings as pictures.

10.9.8 Brownlee2017 - How Does Attention Work in Encoder-Decoder Recurrent Neural Networks [63]

10.9.9 MachineTalk2019 - Create The Transformer With Tensorflow 2.0 [2]

10.9.10 Rubik2019 - Introduction to Transformers Architecture [3]

10.9.11 DeepMind2019 - Learning explanatory rules from noisy data [22]

10.9.12 SkyMind2018Symb - Symbolic Reasoning (Symbolic AI) and Machine Learning [1]

10.9.13 Vinayavekhin1806 - Learning and Understanding using Attention, Focusing on What is Relevant: Time-Series [110]

10.9.14 Galassi1902 - Attention please! A Critical Review of Neural Attention Models in and Natural Language and Processing [31]

This review article concentrates on the recent developments of Neural Attention Models, specifically designed for Natural Language Processing (NLP).

It presents a generalized model for such networks and emphasizes the ability of these networks to offer explanations of what is going on inside neural networks. For text applications it shows a visualization tool, something similar could be created for image processing. It also offers two references to time-series analysis (Tran2018, Song2018) which we should follow up.

The most important part of the approach is the Attention Function, which learns a probability distribution over all its inputs over time. Other than with LSTM there can be multiple and distant dependencies. In many cases, the input consists of a bi-directional RNN and the output is another RNN. An important example application is RNNsearch (Bahdanau2015). The paper offers a unified Attention Model, which is based on the work of Daniluk2017 and Vaswani2017.

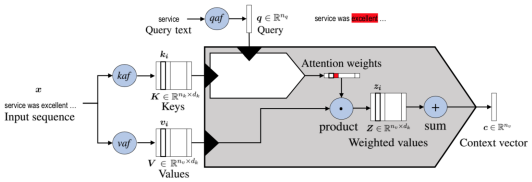

Figure 19: *Unified Attention Model.*

In some other approaches, the input consists of contextual information, extended with background knowl-

edge. In Hierarchical Attention Models this model is extended to mutpile layers, where the Attention Function is often combined with a RNN, a GRU or a CNN. A Hierarchical that analyzes an input sequence and transforms it into an output sequence is called a sequence-to-sequence annotator.

A very interesting approach is to combine an Attention Model with Knowledge. According to LeCun, Bnegio and Hinto (2015) this allows for a combination of sub-symbolic models with symbolic knowledge and may serve as a basis for explaining what a neural network learns. It is also proposed as a new way to allow neural networks to perform complex reasoning tasks. Neural Symbolic Learning is described in Garce, Broda and Gabbay (2012).

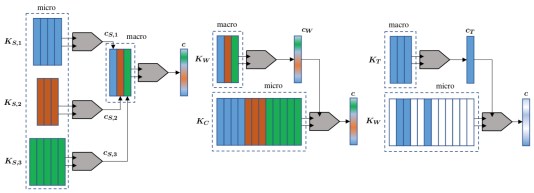

Figure 20: *Hierarchical Attention Model.*

10.9.15 Skymind2019 - A Beginner's Guide to Attention Mechanisms and Memory Networks [115]

Attention Mechanisms are currently state-of-the-art in Natural Language Processing (NLP). Convolutions take a moving window over an input image and finds features in the image that assist in recognizing objects in the image. With NLP, but also with analysis of time series, the determining features are hidden in the differences between succeeding time steps or words in the input. For this a Recurrent Neural Network (RNN) is used, in which a Long Short-Term Memory (LSTM) cell is learning the dependencies between successive steps. However this approach puts an emphasis on events being close together, distant influences cannot be represented by such a network.

An Attention Mechanism makes a table of two events (or sentences) and puts the elements in the rows and columns, like in a confusion matrix. It then learns the relationships between all elements of this table, which makes it find relevant properties of both structures. It finds the strongest relationships between these events and this results in an Attention Unit. These units can be stacked, to form Encoders. Several Encoders together form a Transformer, that results in a smarter representation of the input data.

The most common architecture of a transformer has a Key-Value pair as input See Figure 19. It also has a query as input, that searches over all Keys and finds the values or different associations of that Key and then returns a vector with the most probable meaning of this particular word or event.

10.9.16 Yang2008 - Hierarchical Attention Networks for Document Classification [119]

This paper also concentrates on assigning labels to text. It introduces Hierarchical Attention Networks (HAN) in which for instance one level learns the association between words, while the second learns information about an entire document. Although most papers concentrate on NLP, the same methods can be applied to the analysis of time series. Here a structure is described of a Word Sequence Encoder, connected to a word-level attention layer and a Sentence Encoder with a sentence-level attention layer.

Instead of an LSTM, most Attention Mechanisms are using the simpler Gated Recurrent Unit (GRU). A GRU has two gates; a reset gate and an update gate. The update gate decides how much information is kept and how much new information is added. This work uses a bi-directional GRU (Bahdanau2014). It uses a single layer Multi Layer Perceptron (MLP) to learn the importance of the input. The HAN is compared to other methods like SVM, LSTM, CNN, GRNN and concludes that HAN is more powerful than all of them. Another important feature is that a visualization technique is used to show the relevance of words as part of a piece of text.

10.9.17 Steels2005 - The Symbol Grounding Problem has been solved. So what's next [91]

As we are looking for ways to let an Opponent model find intentions of individual players as explanations of their behavior, we must connect these behaviors to symbols. grounded in the robotic soccer-playing world. This is a symbol-grounding problem and as such this article is very relevant. Luc Steels has been working on language acquisition by robots for many years, and this essay is the result on a workshop on the Symbol Grounding Problem. It is a defense against the claims of Searle in his Chinese Room experiment, in which he claims that symbol grounding is only possible in embodied organisms and is the result of an unknown biochemical or quantum process. This essay shows

that autonomous systems are capable of generating and communicating their own symbols and ground them to their physical world. However he makes the distinction that these symbols are computer symbols and are not related to the human world. These symbols develop in a semiotic network by trial and error between agents and when both agents share a sufficiently similar semiotic network, communication is possible, using grounded symbols. In our case we would be using the already existing semiotic network of human soccer players and connect this with the equivalent symbols in the robotic network, to allow robots to communicate about human grounded symbols. That allows communication and forms the basis for explanation facilities of the proposed Opponent Model.

10.9.18 Dennet1971 - Intentional Systems Theory [23]

This summary of Dennet's 1978 theory explains the differences between first-order and second-order intentional systems. An intentional system has beliefs and desires that lead to predictable behaviors. A second-order intentional system has beliefs about beliefs (and desires) and is capable of reflection. Simple agents like animals and robots are first-order intentional systems.

In the case of a robot, the actual beliefs and desires are derived from those of its creators. In our case taking the intentional stance allows us to talk about desires, beliefs and actions that are grounded in the soccer-playing-robot world. This gives the system explanatory power and allows us to communicate with the agent using the learned model as a semiotic network, shared by the robot and it's users.

This is a rather philosophical approach that ties in closely with the work of Steels about grounding symbols [91].

10.9.19 Chen1711 - Dialogue Act Recognition via CRF-Attentive Structured Network [15]

This work on Dialog Analysis is performed in the field of text analysis, part or Natural Language Processing (NLP). There are a lot of similarities between Dialog Analysis and recognizing action patterns. In both fields, the intention of the actor is used to classify action sequences. In our case these are movements, in the case of NLP they are utterances. So a hierarchy of Intentions, Acts and Roles corresponds to many of the aspects used in NLP.

In this paper a comparison is made with traditional hand-crafted features, used in Recurrent Neural Networks and the approach, called CRF-ASN. A Linear Chain CRF is input to a number of Hierarchical Attentive Encoders, to select the actions that are most relevant, given the current context. In this way an act is interpreted as part of an group of actions, within the context of the dialog, comparable to our Game State. The problem is a typical sequence labeling task.

The paper contains many implementation details and references to relevant literatur and achieves an accuracy of 81.3 to 91.7%, much higher than previous methods. Attention mechanisms were first proposed in Bahdanau2015. The Neural Network involved uses bi-directional GRU's, described in Bahdanau2014.

10.9.20 Liu2018 - Relational Attention Networks via Fully-Connected Conditional Random Fields [62]

10.9.21 Nazerfard2010 - Conditional Random Fields for Activity Recognition in Smart Environments [72]

10.10. Formation Analysis

10.10.1 FutsalExpert2018 - 6 Amazing Futsal Formations And Team Setups (8) [30]

This blogpost provides an overview of 6 Futsal formations. Although the FIFA, UEFA and KNVB coaching manuals also show these formations, the explanation of them are very clear and useful. A summary: Defensive Formations:

The Pyramid (2-1-1)
2 Defenders, 1 midfielder and 1 attacker form a defensive lineup in a half-court pressure play. The midfielder is the link between defense and attack. He is the third man and the only one supporting the attacker during a counterattack.
The Wall (3-0-1)
A super defensive strategy, to adopt when things get hard. It uses 3 defenders and 1 attacker. Useful to prevent the other team from scoring while protecting an advantage.
Neutral Formations:

The square (2-0-2)
A balanced formation for less experienced teams with two defenders and two attackers. Most teams attack or defend with 3 players. Role switching is needed when supporting an attacker or defender.
The diamond (1-2-1)

This is the best formation for a team that has a lot of ball possession.IT uses two midfielders. The midfielders can be trapped and become isolated form the defender.
Attack Formations
The Y (1-1-2) If the other team is not so strong and you are willing to take a risk, this formation puts pressure on the offence. Is good to gain the ball. Position of the midfielder is key to support the defender when the opponent has the ball.
All or Nothing (1-0-3)
This formation is scary. Do not use this formation for a long time and only when your team is at a disadvantage and you are at the end of a life-or-death match.

10.10.2 Power2017 - "Not All Passes Are Created Equal:" Objectively Measuring the Risk and Reward of Passes in Soccer from Tracking Data. [77]

This paper from STATS describes a system to analyze passes differently from human annotators. The usual way is binary, by distinguishing between successful and failed passes. This paper argues that the success of a pass needs to be measured by its risk and reward. By analyzing two seasons of a professional soccer league they show how these two factors are being estimated by a learning system. The best way is to create dangerous situations by passing the ball to create an imbalances between the attack and defense. Taki and Hasegawa [14] looked to measure the dominant region of a player in order to model the probability of a players region intercepting that of the ball. Link et al., [7] hand crafted a set of features using player and ball spatial-temporal data; pressure, density, zone and control to measure the dangerousness of a pass in the attacking 3rd of the field. Gyramati et al [5] proposed a "QPass" method to quantify the quality of a pass As we get closer to the goal, it can be seen that passing gets riskier with passes in the forward third being successfully made 58% of the time, and passes into the penalty area being made 37% of the time. We utilize the player trajectory data to craft soccer specific features which we define as Micro Features:

1. Speed of the player in possession and the intended receiver.
2. Speed of the nearest defender toward the passer and the receiver.
3. Distance of nearest defender to the passer and receiver.
4. Nearest defender angle to the passing line.
5. First time pass.
6. Time from regaining possession.
7. Intended receiver.

The expected receiver membership can therefore be determined as:
Expected Receiver = Distance / Min Distance * Angle / Min Angle

Pass reward is estimating the likelihood that the pass made will result in a shot within the next 10 seconds. We need to capture the tactical features (i.e., game-state) as well as the formation features (i.e., team-structure). Three distinct game-states are used for analysis: i) build-up, ii) counter-attack, iii) unstructured-play. Analysis is done based on formations. We employed a formation clustering method described by Bialkowski et al., [1]. This is important as we can now capture if a pass is being played between the lines of midfielders and defenders or if a pass is attempting to break the final line of defenders. A defensive block can be split into high-block, medium-block, and low-block.Because the number of defenders in front of the ball is higher than our counter attack example, the skill level required to complete a more penetrative pass is larger. Additionally, we define a dangerous pass, which is a pass that is in the top 25th percentile of passes with the highest reward. The ability to play the critical pass that unlocks a defense is one of the most highly sought after skills in soccer. By assessing the reward of each pass during a play, we can objectively assess who is responsible for changing the attacking momentum in a possession. By combining the passing risk and passing reward models we can now learn a new dictionary of objectively measured dangerous or critical passes. We also want to be able to assess the impact the pass will have in creating danger for the opposition. We therefore define a Dangerous Pass (DP) as an attempted pass that has a greater than 6% chance of leading to a shot in the next 10 seconds. The analysis process for a coach is to quickly find patterns a team will use and understand how dangerous these patterns are. We can immediately see that the most dangerous passes occur around the edge of the penalty area. These passes have the highest reward they are also have the highest risk. In this paper, we presented an objective method of estimating the risk and reward of all passes using a supervised learning approach. Adding contextual features improved the prediction performance in addition to giving semantic information to each pass.

10.10.3 Visser2001 - Recognizing Formations in Opponent Teams (7) [111]

This work by the University of Bremen is based on work of CMU and learns team behavior. References are made to the work in the Simulation League. In order to switch between team behaviors, they analyze the team formations. In addition they analyze defensive behaviors like man-to-man marking, zone defense and catching a pass. To learn the different formations, they interviewed human experts. Based on this they define an 8x8 grid. This is then fed into a Neural Net, which classifies the 10 player formations into 16 different formations. The best performance was 72.27%, based on 612 input frames and 68 test frames. There is a reference to a paper on human understanding of team behaviors (Taylor Raines 2000) and adversarial learning (Stone and Veloso 1998).

10.10.4 Disney2017 - Computer algorithm automatically recognizes soccer formations and defensive strategies

This TechXPlore.com weblog explains work, done by Disney in cooperation with CalTech and STATS to recognize formations and team strategies. The technique emplyed is imitation learning. The work, this post is referring to is Coordinated Multi-Agent Imitation Learning (Le2017 [53]).

10.10.5 Le2017 - Coordinated Multi-Agent Imitation Learning

This recent paper brings together Team Sport Analysis and RoboCup approaches to infer the roles of players from sports tracking data. This paper refers to multi-agent imitation learning as a method to learn a model of team behavior based on tracking data. This work, too, emphasizes the importance of establishing player identities, and their role. In the sports analysis community this is known as index-free multi-agent control (Kingston and Egerstedt 2010). They call their approach a Stochastic Variational Inference approach.

This work realizes index-free policy learning via role-learning and role-based indexing, learning an efficient policy per role. The experiments are performed on 45 games of professional soccer in the European League. The training data consists of 7500 sets of trajectories with a total of 1.3 million frames at 10 Hz. The average sequence length is 176 steps. They are using an LSTM trained on overlapping 50-step sequences. The model is learned via stochastic variational inference on a continuous HMM. There is a reference to Generative Adversarial Imitation Learning (Ho2016) and Structured Stochastic Variational Inference (Hoffman2014).

10.10.6 Power2018 - Mythbusting Set-Pieces in Soccer

This paper (coming from Sports Analytics Research) explains the usefulness of Set-Pieces and begins with a list of misconceptions about Set-Pieces. The analysis of the usefulness of Set-Pieces is conducted by comparing the chance of success of 20 Soccer Team in the UK and finding out their strategies when taking a corner. The analysis is done based on hand-crafted features and then visually represented as heatmaps. In order to compare the various tactics, a generalized Set-Piece Grammar is developed, that could be very usefull, although it is only applicable to taking corners in 11x11 soccer.

The analysis is done by a rule-based system, based on information retrieved from human experts. As an example, they did show that an in-swing cross has a 10.81% chance of a goal, compared to a 6.46% chance for an out-swing cross. Other factors are how many defenders are on each goal post. Using the same method they are able to determine the kind of defense is emplyed: man-to-man, zonal or a hybrid. They also concluded that a Neural Network to learn all approaches based on the hand-crafted features is a better approach and tested this with a 46x25 picel input image. Details about the network configuration are also given. The detection accuracy was 82.53% based on 1500 input samples. They also note that many more goals are scored from open play than from Set-Pieces.

The way that they predict an opponent's behavior is through four levels of analysis:

1. Defending strategy (man-to-man, zonal, hybrid).
2. Delivery of Set-Pieces (Short, Long, Flick-On, Location), although not clear what they mean with those.
3. Opponent's strong and weak points.
4. How is the opponent likely to play against our team (How do we defend and how will they deliver the ball).

To compare opponent strategies they use a technique called affinity propagation to cluster average team behavior (ref 12 - Frey and Dueck 2007)

10.10.7 Felsen2018 - Where Will They Go? Predicting Fine-Grained Adversarial Multi-Agent Motion using Conditional Variational AutoEncoders (9) [27]

This is another paper from Sports Analytics by STATS UK, the same group that is working with Disney. It is about predicting opponent behavior and uses a number of AutoEncoders to learn the opponent behavior. Their method is called Conditional Variational AutoEncoder (CVAE) and analyzed 1200 Basketball games, using position data of players and the ball.

Two important ascpects of this work are: the use of Context in the analysis, which in this case means information about the game state and multi-agent alignment, which means that in all cases the identity of the players was normalized to allow comparisons.

They make references to the work of DESIRE (Lee 2017), which used a CVAE RNN Encoder/decoder to learn team behaviors. Another approach used Social LSTM's also using hand-crafted features. Lucey 2013 investigated players roles.

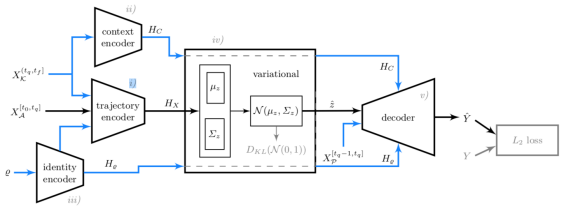

Figure 21: *CVAE Encoder/Decoder (Felsen2018).*

The Model has three input encoders: The Identity Encoder, which identifies the player, Context Encoder, which provides the game situation and the Trajectory Encoder, which provides the path of the agent. The CVAE learns the various movements and the decoder provides the predicted path. There are a number of complex calculations involved, like Kullback-Leibler Divergence and the trajectory decoder to minimize Euclidian distance.

There are implementation details about the used network configurations. The best configuration achieved a 5.8% error rate on trajectory prediction, where a 1 sec history and role assignment with context and team identity were used.

10.10.8 Wagenaar2019 - Using Deep and Convolutional Neural and Networks to Predict and Goal-Scoring Opportunities and in Soccer

This is work done at the University of Groningen and related to Sports Analysis as well. They use a Convolutional Neural Network to learn self-organizing maps to identify defensive and offensive patterns in soccer. They analyzed two different game situations: snapshots before goal-scoring and loss of ball-possession situations. They use a time window of 10 seconds. The input consists of digitized games from the Bundesliga teams. They normalized the data so that all games are played from right to left. GoogLeNet with incpetion is used in both a pre-trained and a fresh trained version as well as a simple 3-layer CNN.

The input consists of 6300 images and the best version was GoogLeNet trained from scratch with enhanced images, which achieved 67.1% accuracy.

10.10.9 Power2016 - Not All Passes Are Created Equal: Objectively Measuring the Risk and Reward of Passes in Soccer from Tracking Data

10.10.10 Le2017a - Data-Driven Ghosting using Deep Imitation Learning

10.10.11 Zheng2016 - Generating Long-term Trajectories Using Deep Hierarchical Networks

10.10.12 Ruiz2017 - The Leicester City Fairytale?: Utilizing New Soccer Analytics Tools to Compare Performance in the 15/16 & 16/17 EPL Seasons

10.11. Action Models

10.11.1 Zhang2019 - A Comprehensive Survey of Vision-Based Human Action Recognition Methods [121]

This recent overview of Human Action Recognition gives a structured summary of all current methods to analyze video and detect and recognize Human Actions. This is relevant, because we want to be able to recognize movements on the field as intentional events, which of course is a lot simpler but could use the same or simpler versions of these approaches. All methods are based on the analysis of time series of data. The most important aspect of the problem is the selection of a good feature representation. Traditionally feature selection was done manually, but advances in Machine Learning have shown that automated feature learning leads to much better results. A very popular method is Spacial Temporal Interest Points (STIP).

The use of depth cameras has stimulated the development of depth and skeleton analysis, where human action feature representation is very important. We must make a distinction between action detection and

action classification. Action prediction is the next step after detection and classification. Further research concentrates on Action Representation and Interaction Recognition. Also single person and multiple person actions are being studied.

For hand-crafted features the methods Boost, Support Vector Machines (SVM) and Probability Map Models are mentioned. To represent actions the methods Motion Energy Image (MEI) and Motion History Image (MHI) are being described. Well-know methods as SIFT and HOG are being used, but all of these approaches seem overkill when applied to our simple, single-point 2D images. Trajectory-based action recognition and the improved Dense Trajectory (IDT) seem more appropriate for our application. (Wang et al [13] [113]).

Many more approaches are being discussed here, but they all concentrate on depth and skeleton approaches, which are dedicated to Human Action Recognition. The Human aspect is relevant if we were to analyze video sequences of humans or humanoids, playing soccer. With simple, wheeled robots the problem is confined to the 2D world, represented as single point trajectories. Interactions with the ball and robots are the only complicating factor here, and could only benefit from these studies in a limited way. Several other articles were found on this subject (see below), but for the moment they do not seem relevant to our work.

10.11.2 Ghosh2018 - Deep Learning for Videos: A 2018 Guide to Action Recognition [34]

This very readable blog post gives a good overview over the developments in Human Action Recognition over the past 4 years. For all major developments the most influential papers are summarized along with their major contributions and the impact that has had on the state-of-the-art. The final results are described in more detail in the previous paper (Zhang2019) and results in a two-stream neural network in which temporal and spatial data is processed separately and optical flow is processed as a preceding step in the learning process.

10.11.3 Wang2013 - Action Recognition with Improved Trajectories - [113]

10.11.4 Cheng1501 - Advances in Human Action Recognition: A Survey [16]

10.11.5 Rahmani1602 - Learning a Deep and Model for Human and Action and Recognition from Novel and Viewpoints [78]

10.11.6 Bagautdinov2014 - End-to-End Multi-Person and Action Localization and Collective Activity [7]

10.11.7 Kasteren2008 - Accurate Activity Recognition in a Home Setting [102]

While most research is done on Human Action Recognition, we need to concentrate on a much simpler 2D domain with single action points and trajectories. This work, performed at the University of Amsterdam analyzes human activity in a house, using simple on/off sensors, placed on doors and spots in rooms. With these sensors the movements of a single person is tracked in a house and from this tracking information, activity patterns are learned like, toileting, washing, cooking etc.

In order to learn associations between sensor readings and activities two different approaches were tested; a Hidden Markov Model (HMM) and Conditional Random Fields (CRF). Both performed well and had only small differences, depending on the way the sequences were interpreted. So as a result we studied the differences between the two approaches a bit in more detail.

The approach, described in this paper is much more in line with our requirements and so we will further investigate the application of HMM and CRF techniques along with a recurrent network, to learn to recognize and classify robot movement patterns.

10.11.8 Eddy1004 - What is a hidden Markov model? [26]

A Hidden Markov Model is a method to make probabilistic models if linear sequence data. HMMs are the standard building blocks for automated sequence analysis. They are used in speech recognition and many other domains. In our case we see them used frequently in Reinforcement Learning systems. A HMM consists of states, state transitions that have a probability and a set of observations. These observations are the difference between a Markov Decision Process (MDP) and a HMM, where the observations are asso-

ciated with the hidden state transitions, that we need to infer from the observations. To efficiently calculate the most probable state, the Viterbi algorithm is used.

HMMs assume that there is only a dependence on the previous state. So in cases where there are multiple dependencies, a HMM is not appropriate.

10.11.9 Echen2012 - Introduction to Conditional Random Fields [25]

Conditional Random Fields (CRF) are similar to HMMs but are more flexible and can handle multiple dependencies. A CRF that is only depending on previous labels is called a linear-chain CRF, which is essentially the same as a HMM.

A CRF defines a number of feature functions that each return a value. The sum of all these values is transformed into a probability. In that respect a CRF is also Logistic Regression and form a log-linear model for sequential labels. So every HMM is also equivalent to a CRF. A CRF can model a much richer set of label distributions.

10.11.10 Ghahramani2001 - An Introduction to Hidden Markov Models and Bayesian Networks [33]

This review paper is a scientific overview of Hidden Markov Models, their aspects and in the context of learning systems.

10.11.11 Wallach2004 - Conditional Random Fields: An Introduction [112]

This scientific paper explains the way Conditional Random Fields work in the contect of Part-Of-Speech (POS) analysis.

10.11.12 Singh2016 - Recognizing Actions in Motion Trajectories Using Deep Neural Networks [90]

This work in the domain of Pretend Play, in which users let cartoon characters describe trajectories that must be recognized as actions by other users, is based on a Deep Convolutional Neural Network (CNN). Instead of using a Recurrent Network to recognize action sequences, the time domain was converted into action traces. The program shows an action and a series of human users are asked to classify the action pattern. These supervised action patterns form the ground truth and a standard dataset for this domain (teh Charades dataset).

The system uses augmentation by flipping, rotating and shearing the input to create more examples and prevent overfitting. Details are given about the network architecture. This approach resulted in 99.5% recognition as opposed to 12.6% with previous attempts. With multiple characters involved at the same time, the accuracy dropped, partly because the time aspect was dropped, partly because processes like following and accompanying could not be represented. The approach is very similar to the work of Klooster2018.

10.12. Team Description Papers

We need to include the TDP's from teams that have developed important facilities, related to our intended system:

1. BrainStormers/Tribots
2. Cambada 2016/17/18
3. CMU 2016/17/18

10.12.1 Hafner2002 - Brainstormers Tribots Team Description Paper [37]

All software for the Tribots is published and we have a copy of it which we can study. The research focus is on Q-learning, then still called Reinforcement Learning. RL is implemented in a number of Behaviors as part of a hierarchy of functions. They have used this approach in SSL, simulation and MSL. It also mentions the TwoBots, their first humanoid team.

10.12.2 Muller2006 - Making a Robot Learn to Play Soccer Using Reward and Punishment [70]

10.12.3 Riedmiller2008 - Learning to Dribble on a Real Robot by Success and Failure [81]

This very short paper describes an early attempt by the Brainstormers to use Q Learning on a real robot. The task is to dribble and rewards are given when a dribble is executed correctly. The criterion for a good dribble is when a sharp turn is made, while keeping the ball at a short distance and making progress into a given direction. The input is the camera image and the actions are split into X and Y movements. Four different combinations of action pairs with different speed vectors are used.

10.12.4 Riedmiller2009 - Reinforcement learning for robot soccer [80]

This paper concentrates on the application of Value/Function based Q-learning strategies using a

Neural Network. The approach works on the basis of time steps in which an agent learns the best actions in response to the current game state. It is based on Markov Decision Process (MDP) using a 4-tuple input M = [S, A, p, c] in which S=State, A=Action, p=probability and c=cost. Cost is expressed as c(s, a, s') denoting the cost of changing state a into state a' using action a. This work concentrates on optimizing the long-term cost, not immediate cost. This paper already shows the contours of later work in batch-mode Q-learning. It collects a series of experiences and finds the best Value/Function for an experience. This repeated use of experience batches is later also used in the DeepMind Atari Games approach. It depends on a world model, which is called a predictive world model in which it anticipates future actions in an attempt to overcome time delays in the input. It also features a hierarchy of models for dribbling, shooting and higher levels dealing with tactics. So there are a number of learn-able sub-tasks that can be combined with hand-crafted tasks. Time steps in this system are 100 ms each. In this model 60 turn actions and 16 dashes are used, while the world model has 9 dimensions, including player positions, ball position and area on the field where the play occurs. It concentrates on Aggressive Defense Behavior (ADB). Learning is done per episode, where rewards and penalties are assigned at the end of an episode. They use predefined starting situations to train certain low-level behaviors. IT uses Monte-Carlo simulation, which we also find later on in the Alpha-Zero approach. An important remark made is that opponents did seem to behave differently in seemingly symmetric situations. Dribbling behavior is described as learning to select from 5 different action triples, consisting of (X, Y, Rot) (2.0, 0.0, 2.0) (2.5, 0.0, 1.5) (2.5, 1.5, 1.5) (3.0, 1.0, 1.0) (3.0,-10.0, 1.0). Apparently these are sufficient to control the dribbling behavior. The Brainstormers used this in the Midsize League in 2006, 7 and 8 and used MotorSpeed, Go2Pos, Intercept and Dribble.

10.12.5 Cooksey2018 - CMUus 2018 Team Description

10.13. Papers to classify and comment

10.13.1 Denzinger1998 - Evolutionary On-line Learning of Cooperative Behavior with Situation-Action-Pairs

10.13.2 Bergkvist2003 - Machine learning in simulated RoboCup and Optimizing the decisions of an Electric and Field agent

10.13.3 Stone2005 - Reinforcement Learning for RoboCup Soccer (Keepaway)

10.13.4 Muller2006 - Making a Robot Learn to Play Soccer Using Reward and Punishment

10.13.5 Soccer2010 - Robot Soccer and -Strategy Description and Game Analysis

10.13.6 Kurek2015 - Deep Reinforcement Learning in Keepaway Soccer

10.13.7 Soccer2017 - Interactive Machine Learning Applied to Dribble a Ball

10.13.8 Caitlin2017 - Learning a Robot to Score a Penalty Minimal Reward Reinforcement Learning

10.13.9 Maeda2017 - Active Incremental Learning of Robot Movement Primitives

10.14. To be included and evaluated

10.14.1 Haddadi1995

10.14.2 Sahota1994

10.14.3 Littman1994

10.14.4 Veloso1996

10.14.5 Andre1998

10.14.6 Matsubara1999

10.14.7 Birnsted1999

10.14.8 Balch1998

10.14.9 Veloso1997

10.14.10 Konda2000

10.14.11 Mnih1602

10.14.12 Foerster2018

10.14.13 Sunehag2018

10.14.14 Silver2016

10.14.15 Silver2017

10.14.16 Albrecht2018

10.14.17 Arulkumaran2019

10.14.18 Bahdanau2014

10.14.19 Bahdanau2015