Using machine learning techniques in complex multi-agent domains

Martin Riedmiller and Artur Merke

Institut für Logik, Komplexität und Deduktionssyteme
University of Karlsruhe, 76131 Karlsruhe, FRG

Abstract. Machine Learning methods have been widely accepted for their usefulness in many application domains, such as pattern recognition, prediction and control. Although today many algorithms are available for a wide range of different problem types, still a lot of interesting research questions are open. One such area is the application of machine learning techniques in increasingly complex domains.

The RoboCup federation offers a challenging and complex testbed for all kinds of scientific methods by organising international competitions in robotic soccer. The following article describes the appproach of the Karlsruhe Brainstormer soccer agent - a complex software system that in main parts is based on modules realized by machine learning techniques. In particular, Reinforcement Learning methods are applied. These learning methods allow 'autonomous learnig', i.e. system improvement by mere goal specification.

1 Introduction

1.1 Intelligent Agents

A challenging current research direction is the design of intelligent software systems - 'agents' - that are able to autonomously solve certain tasks within their environment. Application areas of software agents can be found in robotics, as for example agents that control robots to rescue people in dangerous environments, and also in virtual worlds as electronic markets, where intelligent agents have to compete against other market participants, that pursue their own goals.

An 'intelligent' agent as we understand it here, is a software system, that acquires parts or all of its behaviour autonomously by observing reactions from its environment. Methods from machine learning can be used to realize such adaptive behaviour. Machine Learning methods have been widely accepted for their usefulness in many application domains, such as pattern recognition, prediction and control. Although today many algorithms are available for a wide range of different problem types, still a lot of interesting research questions are open. One such direction is the application of machine learning techniques in increasingly complex domains, i.e. to examine questions of scalabiltity (what happens when applications get increasingly complex?), of time-efficiency and data-efficiency (how to reduce the time and data required for learning?), and so on.

As a further problem, when facing a complex real world task, it is often not clear, which machine learning algorithm to use for which parts of the problem. Often, the best solution will be one that combines different techniques - learning and non-learning - in a hybrid approach. Currently, no systematic approach for 'intelligent software system design' is known. To collect experience, it is important to tackle complex application problems and to compare different approaches in a fair benchmark. With respect to these points, robotic soccer offers an ideal testbed.

Another aspect is central to the robotic soccer domain: the question of cooperation among a team of individual agents. We will show how a team of individually acting intelligent software agents is able to cooperate to achieve a common goal. In principal, the goal in robotic soccer is easy to define: 'win the match'. At the current stage of our development we break this overall goal in two subgoals: score as many goals as possible and get as few goals as possible. Clearly, this is just an approximation to the final goal definition of winning the match, but it has the advantage, that the defined subgoals are more precisely defined and therefore better accessbile for today's learning techniques. An example of learning how to score goals is given in section 4.

In the following we will describe the appproach of the Karlsruhe Brainstormer soccer agent - a complex software system that in main parts is based on modules realized by machine learning techniques. In particular, Reinforcement Learning methods are applied. These learning methods allow 'autonomous learnig', i.e. system improvement by mere goal specification.

1.2 Robocup - challenges of robotic soccer

The official longterm goal in the robotic soccer initiative RoboCup is that in 2050, a robotic team shall be able to beat the human world champion in soccer. It currently is not clear, whether this goal is realistic or not, and opinions of researchers in this field range from '20 years is already enough time' to scepticism of those who remember the unrealistic promises of artificial intelligence research some decades ago. What can be definitely said, is that even by trying to reach that longterm goal a lot of scientific progress will be made along that way. Contributions will be made in two important fields: the development of novel robotic hardware and its use, i.e. sensor and actuator devices, and in the development of novel 'intelligent' software techniques.

Robotic soccer offers a special type of benchmark, with typical requirements for future hard- and software systems. Among them are the requirement for real-time sensor evaluation and decision making, acting in complex and highly dynamic domains, consideration of stochastic and noisy effects, and acting against a team of opponents in a highly competitive domain.

One of our goals is to formulate our methods for 'solving' the problem of playing robotic soccer within a general mathematical framework, namely Markovian Decision Processes (see section 2.2). This will allow to bring the (learning) methods developed for robotic soccer to new application areas - as for example dispatching jobs in factories among a 'team' of production machines with one common goal - to maximize profit.

Besides the variety of typical application aspects, robotic soccer offers another advantage - the criteria of success is very simple to decide: the team that marked the most goals, wins.

1.3 The four different leagues

Since the overall problem of playing soccer has many different aspects, current competitions take part in 4 categories or 'leagues', each showing special features. The mid-size league is a league where 4 real robots with a maximum diameter of about 50 cm play on a soccer-field of size 9m x 5m. They are completely autonomous, that means that they both have all their sensors (typical a camera system) and their computation devices (typically a laptop) on board. In the small-size league two teams of 5 robots play against each other. The sensor processing is done centrally, which means, there is a single camera that observes the whole scene. The robots in this league are much cheaper than those of the mid-size league. The legged-league is the league of the Sony AIbos. These dog-like robots allow the examination of legged movement. Like in the mid-size league, all information processing (sensing and decision-making) is done on board. To be somewhat indepenent of the development of real hardware and to be able to directly start with research on higher level control algorithms, the simulation league focuses on control problems of a team of 11 virtual robots. Since the simulation league is the ideal league for intelligent software development, it is described in more detail below.

1.4 The simulation league

The simulation league is realized as a client/server architecture. The server models the soccer environment and the clients are represented by the corresponding player programs (up to 11 for each team). The server is running in real time updating its state every 100ms. The player programs communicate with the soccer server using the UDP/IP network protocol. This communication is bidirectional: the soccer server sends some sensor information about the world state to the player programs[1] and the player programs can send motion/communication commands to the server (see below). No other communication channels are allowed, in particular every player program runs fully independent of the others.

The soccer environment is bounded by a rectangle $Q = [-57.5, 57.5] \times [-39.0, 39.0]$ in the euclidian plane $\mathbb{E}^2$. In Q the actual soccer field $F = [-52.5, 52.5] \times [-34.0, 34.0]$ is embedded (see figure 1). Each player is represented by a circle with a radius of 0.3m and has control over the ball in a radius of approximately 1m. Collisions between objects are detected and all movements are influenced by friction and small random deviations. To influence the future server state a player has some elementary commands, some of the most important we have listed below

`kick(p,a)`	add a vector $v = f(p, a)$ to the current ball velocity vector, where f is some nonlinear function depending on the ball position.
`dash(p)`	accelerate by factor p in current direction.
`turn(a)`	turn player's body by an angle $a' = g(a)$, where the function g depends on the player's velocity.
`say(s)`	broadcast the message `s` to all players within a radius of 50m.

1.5 General program structure

Through all the leagues - both virtual and real - every robot control programm has to solve the same basic task: based on incoming percepts, taken by the robot's sensors, the programm has to compute an appropriate control action for its effectors, wheels or

[1] The amount and frequency of sensor information depends of the players view angle and quality which are never large enough to cover the whole soccer field.

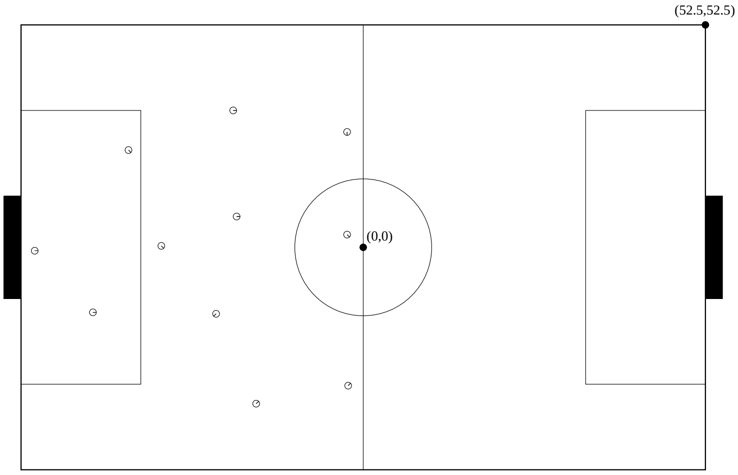

Fig. 1. Soccer field of the simulation league

kicking devices. Such a programm, which is continuously acting embedded in its environment and pursuing a certain goal, is called an agent [Woo99]. The typical structure of an agent in the robot soccer domain consists of two basic components: a module for sensor processing and a module for decision making. The task of the sensor processing module is to estimate the current situation of the world based on the incoming information which may be - and typically is - incomplete, partial and noisy. When different types of sensors are used (e.g. real robots often use cameras and ultrasonic sensors) sensorfusion is an important issue. In the simulation league mainly questions of how to handle past and vague knowledge arise. Also, information from communication has to be integrated. As will be discussed in more detail in section 2, decision making is reasonbly based on the environmet's state estimation computed by the sensor processing module. The task of the decision module is thus to compute a (control) action on the basis of the current (estimated) state. The decision module is the part where currently the research focus of many teams (including ours) lies. It therefore is discussed in more detail in the following section.

1.6 Decision making

For decision making, a broad range of different methods are applied. Deliberate approaches use some kind of rules and logic calculus, imitating aspects of human reasoning. The Robolog team for example uses a Prolog based rule system in order to express tactic rules [SOMB99]. The rules itself are given by a human expert - the role of the system is to deduce which action to take in which situation. Magma Freiburg's tactical decision are made on the basis of a behaviour net. Again, the rules of the behaviour net is designed and tuned by a human expert [Dor99]. The approach pursued by the team of the Humboldt University Berlin is based on the Belief-Desire-Intention (BDI) framework [BHW98]. Besides those knowledge based approaches, where the basic rules have to be formulated by human experts, adaptive and learning approaches play an important role. In [Luk98] a genetic algorithm was used to learn the behaviour of a robot soccer team from scratch. Although some progress in playing has been observed, the team stayed on the level of 'school boy soccer'. An example of how supervised learning techniques could be used was demonstrated in [SV98a]: To learn a simple ball intercept routine, the agent tries various initial turn angles. In a trial-and-error manner, the succesful examples are extracted and used for training a neural network. Reinforcement Learning (which is also our research focus) was used in [And98] for learning of positioning. Stone used a table-based RL approach to learn a multi-agent policy based on simple features [SV98b]. The learned policy was demonstrated to beat some benchmark policies, but its overall performance was too premature to be used in the competition team. In [SSS00] RL is used to learn to play 3-vs-2-keepaway and report drastical improvement over benchmark policy. In their competition team 2000 a hand-coded version of 11-vs-11 keepaway was used.

1.7 The research goal of the 'Karlsruhe Brainstormers'

Reinforcement Learning describes the situation of a machine learning system, where the only training signal provided by the environment is that of success or failure of the agent [BT96a,RS98,Rie00]. The main motivation behind the Karlsruhe Brainstormer's effort in the soccer domain is to investigate Reinforcement Learning (RL) methods in complex domains and to develop new variants and practical algorithms. We consider it important, that we not only demonstrate the principal feasiblity of RL, but actually do apply learned behaviour in our competition team. Our long term goal is a team of learning agents, where we only plug in 'Win the match' - and our agents learn to generate the appropriate behaviour.

The soccer domain allows more than $(108 \times 50)^{23}$ different positionings of the 22 players and the ball - the complete state space considering object velocities and player's stamina is magnitudes larger. In every cycle, even a non-ball-holding agent can choose between more than 300 basic commands (parametrized turns and dashes), which makes a choice of 300^{11} joint actions for the team *per cycle*. This complexity is a big challenge for today's RL methods; in the Brainstormer's project we are investigating methods to practically handle learning problems of such size.

In the following, we will first introduce a useful description of the soccer domain - it is modelled as a Markov Decision Process. Then we will discuss our architecture and especially where we deviate from the proper theoretic modelling in order to gain practical feasibilty.

2 Basic methodolgy

2.1 Principles of reinforcement learning

Reinforcement Learning comprises situations, where a learning system has to learn how to act in an unknown environment. The only training signal given is that of success or failure, after the system has acted over a sequence of decision cycles. This learning problem can be formulated as a Markov Decision Process (MDP) within the framework of Dynamic Programming [BT96b], [SB98]. An MDP is a tuple $M := [S, A, r, p]$, where S denotes the set of states of the environment, A denotes the set of actions available to the system. $p(s'|s, a)$ describes the environtment's behaviour by denoting the probabilty of the environment taking the new state s', if in state s action a is taken. Finally, $r()$

denotes immediate costs that occur each time, when a new decision is made. Now, an arbitrary policy $\pi : S \rightarrow A$ can be characterized by its expected accumulated costs, $J^\pi(s) = E\{\sum_t r(s_t, \pi(s_t))|s_0 = s\}$. The original learning problem then can be expressed as finding the optimal policy $\pi^* : S \rightarrow A$ that receives the minimum expected cumulated costs, i.e. $\forall s : J^{\pi^*}(s) = \min_\pi J^\pi(s)$.

To find such an optimal policy, dynamic programming methods can be applied. A typical method is value iteration, which is based on an iterative refinement of estimates of the optimal accumulated costs for each state. The iteration $\forall s : J_{k+1}(s) := \min_a\{r(s,a) + \sum_{s'}(p(j|s,a)\, J_k(s'))\}$ can be proved to converge in the limit to the optimal cost function J^* under mild assumptions [BT96b]. Once the optimal value function is found, also an optimal policy is defined, by greedily exploiting J^*: $\pi^*(s) := \arg min_a\{r(s,a)+\sum_{s'}(p(j|s,a)\, J^*(s'))\}$. Reinforcement learning methods extend these well-known optimization principles to situations, where large state spaces and/ or unknown transition probabilities hinder the strict application of the pure value iteration method. For example, Real Time Dynamic Programming methods propose the iteration of the value function on real trajectories rather than on the complete state space in order to tackle problems with large or infinite number of states [BSW89]. The algorithm shown in section 3 is an example of such a procedure. When the behaviour of the enivironment as described by the transition probabilities $p()$ is unknown, a variant of value iteration called Q-learning can be used to learn optimal policies by observing the enviroment [Wat89].

2.2 Problem Formulation

The following describes the theoretical formulation of robotic soccer as a **M**ulti-agent **M**arkov **D**ecision **P**rocesses (MMDP) to show the principal difficulties one encounters in this domain[2]. We discuss the framework of discrete time stationary systems as this scenario corresponds to the soccer server environment. A MMDP (or an n-agent MDP) can be defined as a tuple

$$M_n := [S, A, r, p].$$

Here S is the space of all states, A is a cartesian product of action sets $A = A_1 \times \ldots \times A_n$ and p denotes the state transfer probabilities i.e $p(\cdot|s,a)$ is a probability measure on S depending on the current state and the cumulative action $a = (a_1, \ldots, a_n)$.

So far there is no difference to a conventional MDP. The action $a = (a_1, \ldots, a_n)$ could also be produced by a single agent. The difference comes with the reinforcement function r. In conventional MDPs this is a function from $S \times A$ to $\mathbb{R}$. That means that the reward is a scalar value as we have a single agent.

In MMDPs this changes because every agent can receive his own reinforcement signals, and r becomes

$$r : S \times A \rightarrow \mathbb{R}^n.$$

This can also be seen as a tuple of maps $r_i : S \times A \rightarrow \mathbb{R}$ and $r = (r_1, \ldots, r_n)$. If in state s every agent selects an action a_i the system performs an update to the next state s' with probability $p(s'|s,a)$ and produces the reinforcement signal $r_i(s,a)$ for each agent i. So the reinforcement of a single agent i can depend on the actions of all the other agents.

Given our MMDP framework it is possible to discuss optimality of agents behaviours. We will not discuss the most general case in which all agents get unrelated

[2] A more extensive presentation can be found in [Mer99].

reinforcements r_i. The reason is the lack of an universal optimality notion for this case. We will concentrate on two special cases

- cooperative MMDPs
- competitive zero sum (two agents) MMDPs

and will combine them to characterise the robotic soccer environment.

In the cooperative MMDP case all agents get the same reinforcement i.e. $r_1 = \ldots = r_n$. The main difference to a conventional MDP is that the actions a_i are chosen in a distributed way. In the zero sum MMDP we have two agents with reversed reinforcements, $r_1 = -r_2$. So the increase of reward of agent 1 means automaticly an equal decrease of agent's 2 reward and vice versa.

The main advantage of these two special MMDP types is that we can define what optimal behaviour means. To do it in a formal way we need the notion of a total policy and it's value. A total deterministic policy π is a mapping

$$\pi : S \rightarrow A = A_1 \times \ldots \times A_n$$

It is important to see that this definition concerns the behaviour of all agents simultaneously and that the policy of a single agent i can be seen as a projection $\pi_i : S \rightarrow A_i$ in $\pi = (\pi_1, \ldots, \pi_n)$.

It is clear that every agent will concern the value $v_i[\pi]$ of a total policy with respect to his reinforcement signals. There are different possibilities of doing this (cf.[Put94,BT96c]). For our purposes it is just important that such a value is defined and we let the definition a little bit informal

$$v_i[\pi](s) = v_i[\pi_1, \ldots, \pi_n](s) = E\left[\sum_k r_i(s_k, \pi(s_k)) \mid s_0 = s\right].$$

Now concerning the cooperative MMDP case in which all agents get the same reinforcement signals r_i we also have $v_1[\pi] = \ldots = v_n[\pi]$ for all π. All we need is a total policy $\pi^\star$ which has the maximal value[3]

$$\pi^\star = \arg\max_{\pi} v_i(\pi).$$

We denote $\pi^\star$ as an optimal total policy which existence is guaranteed (cf.[Put94]). It is important to see that this definition doesn't say anything about how such a $\pi^\star$ can be computed especially when more then one agent are concerned. Since every agent decides just on one a_i of the whole action a there must be some coordination to assure the choice of the right total action a. We will discuss this peculiarity later on.

In the two agent zero sum MMDP we have a total policy $\pi = (\pi_1, \pi_2)$. Borrowing from game theory we can define a minimax policy for each agent. We look for a total policy $(\pi_1^\star, \pi_2^\star)$ which fulfils

$$v_1[\pi_1^\star, \pi_2^\star] = \max_{\pi_1}\min_{\pi_2} v_1[\pi_1, \pi_2] = \min_{\pi_1}\max_{\pi_2} v_1[\pi_1, \pi_2] = -v_2[\pi_1^\star, \pi_2^\star]$$

Existence of such pair $(\pi_1^\star, \pi_2^\star)$ can be always assured (cf.[FV97]). Such policies are in general no longer deterministic i.e. their map from S to a probability distribution over A.

[3] The maximum is taken over all states simultaneously

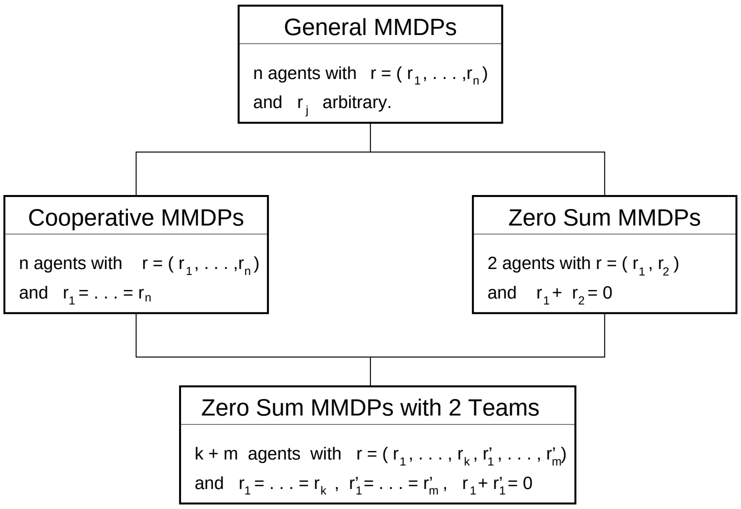

Fig. 2. Hierarchy of MMDP problems.

Let us now turn our attention to robotic soccer. Here we have two teams which perform a zero sum game. Each agent in the same team has the same goal: to score more goals then the agents from the opposite team. This can be expressed as zero sum MMDP of 2 cooperative teams

$$M = (S, A = B_1 \times \ldots \times B_m \times C_1 \times \ldots \times C_n, r, p)$$

where $r = (r_b, \ldots, r_b, r_c, \ldots, r_c)$ and $r_b = -r_c$ express the team respectively zero sum character. Using our previous optimality definitions it is now a straight forward task to define a total optimal policy in the case of two competitive teams. We speak of an optimal total policy $(\pi_1^\star, \ldots, \pi_m^\star, \phi_1^\star, \ldots, \phi_n^\star)$ if

$$v_b[\pi_1^\star, \ldots, \pi_m^\star, \phi_1^\star, \ldots, \phi_n^\star] = \max_{(\pi_1, \ldots, \pi_m)} \min_{(\phi_1, \ldots, \phi_n)} v_b[\pi_1, \ldots, \pi_m, \phi_1, \ldots, \phi_n]$$

The relationship between all introduced models can be seen in figure 2.

2.3 A hierarchical decomposition

To deal with the complexity of a decision making task, a common approach is to decompose the overall task in feasible subtasks. Although there is no general method to break a problem in suitable subproblems, robotic soccer offers a quite natural partitioning: Decision making can be distinguished in decisions implementing individual capabilities (corresponding to skills of a human) and decisions that uses those basic skills to enable team play on a tactical level.

As individual skills, a player needs appropriate procedures to intercept the ball, to dribble with the ball, to properly pass and so on. Each of these procedures emits a sequence of basic commands (turn, dash, kick), until its specific goal is fulfilled. The

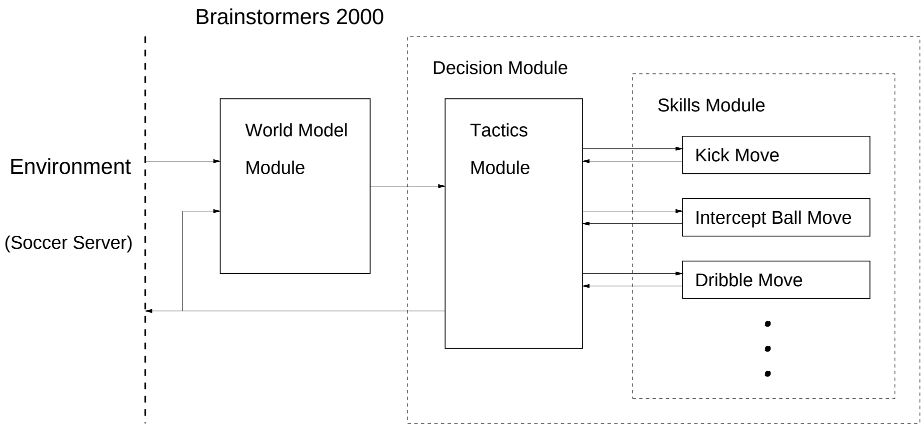

Fig. 3. Architecture of Brainstormers 2000 Agent

procedures implementing the basic skills are called 'moves' in our approach. Characteristically, each move lasts a certain number of cycles and has a clearly defined precondition and a clearly defined goal.

Tactical decisions in our agent are implemented by the policy of our agent. The policy has to select the most appropriate move out of the set of currently available moves. The task of the policy is simply to shoot goals and to avoid getting goals. The current set of moves available depends on the current situation of the agent. In the soccer domain, the tactical decisions have to be done by each agent individually. This raises the problem of learning in distributed systems. Since all the players have a common goal, robotic soccer is a cooperative multi-agent system. In the MDP-framework, the cooperation aspect can be modelled by giving each agent the same transition costs $r()$. The restricted communication ability causes a further difficulty: No agent knows what action is taken by its teammate. It can only observe the resulting state of the joint action. This scenario is sometimes called an 'independent learners' (IL) -scenario. In case of a deterministic environment, we recently proposed a distributed learning algorithm for a team of IL-agents [LR00]. An extension to stochastic domains is currently under development. This will provide the theoretical foundations for learning in a team of individually acting soccer agents. Some emprical results based on heuristic modifications of single-agent Q-learning have already been carried out (section 4.3).

The proposed hierarchical concept has several important aspects, which eventually makes it possible to apply learning methods in this complex domain.

- Each move has a clearly defined pre-conditions and a clearly defined goal. This directly allows the application of Reinforcement Learning methods.
- Since the number and type of moves are restricted to only those skills considered useful for soccer playing, the number of available decisions to the (tactical) policy of the agent is drastically reduced.
- Since each move lasts a certain number of cycles, the number of decisions that has to be taken by the policy is much smaller than the number of cycles per game. This again reduces the complexity of decision making.

The latter two points will allow to apply RL methods also for the tactical level of decision making.

3 Moves: Learning of basic skills

3.1 Motivation

The task of a move is to transform any intitial state that is in accordance with the preconditions for that move into a goal state. Therefore, a move is a procedure that produces a sequence of basic commands by mapping the current state of the cycle to a basic command (turn, dash, kick). It is terminated, when the goal state is reached. As an example, the move *intercept-ball* terminates if the ball is within the player's kick range.

In most of the teams that use a move-related concept to realize basic skills, the skills are implemented using conventional programming. Beside the enormous programming effort that has to be spent to develop a working procedure, it is very difficult to implement an optimal or near optimal procedure, since the environment is stochastic. Thererfore, typically some parameters are added, that have to be manually tuned until the player shows good behaviour. This typically is a very tedious process, which has to be repeated every time the environment changes. Beside, no guarantee for an optimal behaviour exists.

The problems encountered by hand-coding those basic skills was the reason for applying RL methods in the Karlsruhe Brainstormer agent.

3.2 Basic Methodology

Learning of a move is forumulated as a dynamic optimization problem as introduced in section 2. The task is to find a policy, that gets minimum accumulated costs for any initial starting state. Within the RL framework, this is modelled as follows: In each cycle the agent chooses an action. This causes immediate costs $r(s,u) > 0$. When the agent reaches a desired goal state $\mathcal{S}^+$, the sequence is terminated and final costs of 0 are given. For example, when learning to intercept the ball, the sequence terminates with 0 costs as soon as the ball is within the kick range of the player. Typically, the immediate costs are set to a constant value larger than zero, i.e. $c(s,u) = c > 0$. This expresses our desire to learn a time-optimal behaviour. Some moves also have constraints, that means, states that should be avoided. For example, the dribble-move must avoid, that the ball gets into the kick-range of an opponent. Within our framework, this is modelled by a set of negative terminal states, $\mathcal{S}^-$. Whenever the system encounters such a state, the sequence is terminated and 'high' terminal costs arise. These terminal costs for failure can be chosen arbitrarily. Since we are using a neural network with a logistic output function to represent the value function, failure costs of 1.0 are convenient for us. However, care should be taken that no sequence that reaches the positive goal state gets accumulated costs larger than 1.0. This can be provided by setting the immedidate costs to an appropriately small value. Typically we set $c = 0.01$ in our experiments. As already mentioned, a feedforward neural network is used to represent the value function. This is done for two reasons: the state space is continuos and therefore has in principle infinitely many states and secondly, we expect to exploit the generalization abilities of a neural network to accelerate learning [Rie00].

The principle learning algorithm is shown in figure 4. A sequence is started by setting a random starting state. Then, the move selects a basic command - either according to its current policy or randomly to do some exploration - and applies it to the environment. Afterwards, it observes the resulting successor state and performs an update to its current value function. Since in the soccer game we have a model of the physics of the environment when a basic command is applied, model-based learning is done. That means, we learn a value-function of the states. For action selection, in each cycle, for each candidate action the resulting successor state is computed and evaluated. The action leading to the successor state with the lowest estimated accumulated costs is eventually selected.

Init system $s_0 \in \mathcal{S}^0$
Repeat {
   select action
   apply action
   update J }
Until $(\mathbf{s_t} \in \mathcal{S}^+)$
$J_{k+1}(s_t) := 0$

Fig. 4. Basic learning procedure.

3.3 Example: Learning to kick

Finding a good kick routine is a very tedious job. Within our RL framework, this job is done by the agent itself. It is provided with 500 parametrized instances of the kick command (direction is discretized in 100 steps, power is discretized in 5 steps) plus 36 instances of the turn command. This makes an overall of 536 actions, from which the agent can choose one per cycle. The learning goal is to find a sequence of these kicks and turns, such that eventually the ball leaves the player in a certain target direction with a certain target velocity. This target defines the 'positive' States, $\mathcal{S}^+$. If such a state is reached 0 costs occur and the sequence terminates. A failure situation $\mathcal{S}^-$ occurs, if the player looses the ball during a sequence. This results in maximum costs of 1. Since a reasonable optimization goal is to use as few commands as possible for a successful sequence, each intermediate transition causes low constant costs $r(s, u) = 0.01$ [Rie00]. The most challenging thing a kick routine can do is to acclerate the ball up to a maximum speed of 2.7 m/ cycle (this is a restriction of the soccer environment). To generate high speeds of the ball, a highly optimized policy is required.

After about 2 hours of learning on a 500 Mhz Pentium PC, the resulting policies were quite sophisticated - much more efficient than in handcoded heuristic kick routines, the agent learned to pull the ball back, turn itself (if necessary) and to accelerate the ball several times in order to yield high speeds. As shown in figure 5 the neural network based RL module was able to learn to accelerate the ball to speeds up to 2.5 m/cycle successfully in 95% of the situations. The remaining failures presumably occur due to the stochasticity of the environment. Alltogether, three different neural nets were trained, one for target velocities up to 1.0 m/cycle, one for target velocities up to 2.0

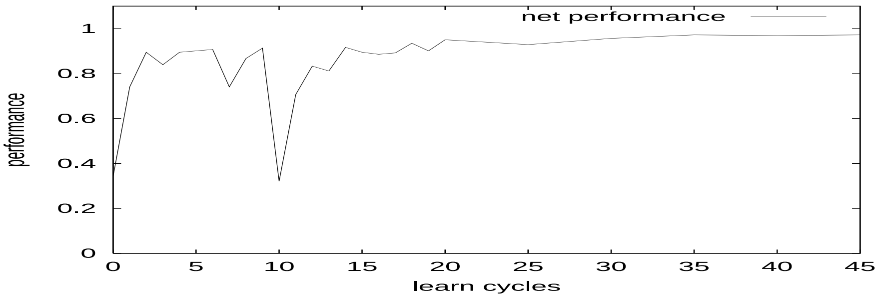

Fig. 5. An example of the performance during learning to kick the ball with maximal velocity 2.5 m/s. Each learning cycle comprises 2000 start states with different ball positions and velocities and the corresponding traversed trajectories. The net stabilizes after approximately 150000 individual updates and reaches a performance of approx. 95%.

and one for target velocities up to 2.5 m/cycle (each of them using 4 inputs, 20 hidden and 1 output neuron).

3.4 Overview of the results

In our current competition teams, all of the basic skills are learned by RL methods. In every case we found that the performance of the previously used hand-coded routines could be drastically improved. For some of the learned skills, no comparable hand-coded routine was available. For example, the learned kick routine was able to accelerate the ball to a speed of 2.5 m/s in 95% of all tested situations, whereas the handcoded routine achieved this only in rare cases. In summary, we learned the following basic skills:

- kicking the ball in arbitrary directions with arbitrary speeds
- intercepting the ball
- going to a position avoiding obstacles
- holding ball against an opponent
- dribbling against an opponent

4 Learning of tactical decisions

4.1 Basic principles

In section 2.2 we discussed how an optimal policy could be defined. What we did not discuss were the difficulties to find such a policy in a distributed way. We will demonstrate this in the cooperative MMDP case considering the solution in the zero sum MMDP with two teams as even harder to reach.

To illustrate the problem of action choice coordination imagine a small 1 state (deterministic) system with 2 cooperative agents, each having two actions:

$$M_2 = [\{s\}, \{a_1, a_2\} \times \{b_1, b_2\}, r = (r_1, r_2), p]$$

The reward function is for both agents equal $r_1 = r_2$ and given by $r_1(s,(a_1,b_1)) = r_1(s,(a_2,b_2)) = 2$ and equal 0 in connection with the remaining actions. It is easy to see that we have two optimal policies $\pi^\star(s) = (a_1,b_1)$ and $\hat{\pi}^\star(s) = (a_2,b_2)$. But if agent 1 decides to take the projection $\pi_1^\star(s) = a_1$ with respect to $\pi^\star$ and agent 2 takes the projection $\hat{\pi}_2^\star(s) = b_2$ with respect to $\hat{\pi}^\star$ we get a total policy $\pi = (\pi_1^\star, \hat{\pi}_2^\star)$ which isn't optimal anymore.

The problem lies in the lack of coordination. In the rest of this section we will give a deeper look to this problem and demonstrate some inherent limitations in multi-agent coordination. To this end we will distinguish different types of multi-agent learning, depending on the amount of information available about other agents during the learning. We will distinguish three main agent types

- White Box Agents (WBA) also called Joint Action Learners (see [CB99]) are agents which have knowledge of all the joint actions $a = (a_1, \ldots, a_n)$ performed in every step.
- Black Box Agents (BBA) also called Independent Learners are agents which just know about their own actions. They aren't really aware of the other agents, all the influence through actions of other agents could also be interpreted as environmental noise.
- Gray Box Agents (GBA) are Black Box Agents which can communicate with other agents. It is no further specified how much information can be exchanged. If no communication takes place we remain having BBAs, but if every agent communicates his action we get the special case of WBAs. But we can do even more with communicating agents, we can exchange information about their future intentions.

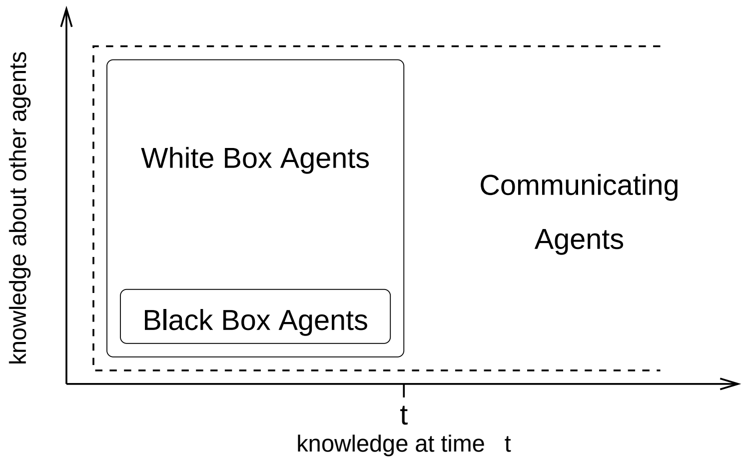

Fig. 6. Illustration of potential knowledge during learning in the different models.

In figure 6 we can see the connections between the agents models. The WBA case can be identified with the single agent MDP situation. We will demonstrate this using the terminology of q-learning (see [WD92,BT96c]). To this end we define

$$q^\star(s,a) := r(s,a) + \sum_{j \in S} p(j \mid s,a) v[\pi^\star](j)$$

in which $\pi^\star$ in an optimal total policy. The value of $q^\star(s,a)$ represents the expected reward[4] for using action a in state s and pursuing an optimal total policy afterwards. The knowledge of $q^\star$ amounts to knowing all optimal policies as we have

$$\pi^\star(s) \in \arg\max_{a \in A} q^\star(s,a)$$

for an arbitrary optimal policy $\pi^\star$. Using Q-learning we are able to compute the $q^\star$ function provided we get access to enough tuples

$$\langle s^t, a^t, r^t, s^{t+1} \rangle$$

where s^{t+1} is the successor state after using the joint action $a^t = (a_1^t, \ldots, a_n^t)$ in state s^t while getting reward $r^t = r(s^t, a^t)$. Using Q-learning every WBA can build up his own $q^\star$ function. The problem of choosing a unique optimal total policy $\pi^\star$ remains, so that each agent can use the projection $\pi_i^\star$ of the same optimal total policy. We will not discuss the possibilities of attaining this goal but we just mention that theoretically sorting all optimal total policies and taking the first will do the job.

The major drawback using WBA and joint actions $a = (a_1, \ldots, a_n)$ comes from the fact that the number of such actions grows exponentially with the number of agents. Furthermore such learning is inflexible with respect of changing the number of participating agents. This reasons are the main motivation of using the considerably weaker model of BBAs. But in this model all that we can hope for is to be able to compute

$$q_i^\star(s,b) = \max_{a \in A, a_i = b} q^\star(s,a)$$

for each agent i. This is the maximal possible expected reward if agent i is using action b in state s. If for example $\{b, b'\} = \arg\max q_i^\star$, so we have more then one action to choose, and we have somehow to decide which of this two we'll actually take. Both problems can be solved if the underlying MMDP is deterministic (see [LR00]) and isn't solvable in the pure BBA scenario when there are probabilistic state transitions (see [Mer99]).

There remains the question of how to deploy communication or some other coordination scheme to solve both problem using BBA agents. As far as we know there has no such algorithm been published yet. In the next section we will present some empirical work, which uses some sort of coordination while working with Black Box Agents.

4.2 Realisation

On the tactical or team behaviour level we didn't use reinforcement learning until very recently. In particular our last year team used search of depth 4 for players with the ball and a priority list of moves for players without the ball ([RMM$^+$00] for more details). As we already mentioned we work with higher level moves on the tactical level. As the first step of developing a whole team policy with RL we started with the attack behaviour.

[4] We don't distinguish different r_i here because all agents get the same rewards.

4.3 Example: Learning to attack

The task of a succesful attack policy can be easily defined as scoring a goal. In contrast to the situation of learning basic skills, where a single agent can solve the task on its own, here the policy must show social or team capabilities. In general, a succesful sequence will involve several players that cleverly cooperate together. This means a policy has to decide where an ideal position for the agent with respect to its teammates is, if a pass to a teammate is better than a selfish dribble or goalshot and so on. Coding an efficient and succesful policy by hand is magnitudes more difficult than implementing basic skills. But finally, succesful team play decides upon the success or failure of a soccer team.

The principle approach of Reinforcement Learning on the tactics level is closely related to the one presented to learn basic skills. However, there are also some main differences: An action is not a basic command, but a complete move. The actions on the tactics level are going to a position, intercepting the ball, dribbling, holding the ball, or kicking and passing. The success state is, when the team marks a goal, a failure state occurs every time when the opponent team gets the ball. Like before, success is rewarded by final costs of 0, failure is punished by final costs of 1. To enforce the agents to cooperate, in each cycle every agent is given the same immediate costs c. Like before, this is a small constant value, resulting in a time-optimal policy.

In a first approach, we tested the principle feasibilty of an adapted version of Q-learning. Each agent's decision is based on the Q-values which are iteratively computed for each possible move. The feasiblity could be shown on learning a cooperative attack policy for two attackers against two defenders [RMM$^+$00]. Using their learned policies, the attackers even played double passes to score against the two defenders. However, when we tried to train with more than those 2 attackers, learning did no longer converge. This was not suprising since each agent has a large choice of possible decisions and the state space grows exponentially when more actors are introduced.

Keeping our goal in mind that we are trying to learn useful policies, which are applicable in our competition team, we were looking for another approach that sensefully restricts the degrees of freedom of the learning agent in order to allow learning in more complex situations. A typical attack situation takes part in the opponent's half, involving mid-field players and attackers or defenders respectively. This consideration leads to the idea of learning 7 versus 7+goalie. For a candidate for learning we first consider the positioning procedure, which applies to situations, where the agent neither has the ball nor has to go for it. This is a very important part in the tactics policy: A clever positioning allows passes to be played and rooms to be opened. Since the ball-possesing part of our policy already works fine, this part was fixed and not learned here. Although, the decision for going to the ball was kept simple and rule-based. This is an example of a hybrid approach, where learned and fixed parts are integrated to a composite policy. In the ideal case, the learned parts are adapted to fit the fixed parts such that the overall optimization goal is fulfilled. In our initial experiments our expectations were confirmed. Playing against our considerably strong defense, the attackers scored or nearly scored in about 29 of 100 cases. In contrast to that, our original attack policy (which was part of our vice-worldchampion team) scored only in 18 out of 100 cases (see table 4.3). This is an improvement by a factor of 1.6!

	Brainstormers2000	RL attack
success rate($Avg.sequencelength$)	18.0%(129.4)	29.0%(118.0)

5 Summary

5.1 Conclusions and future work

The soccer domain can be modelled as a multi-agent Markov Decision Process (MMDP). In this work we provide an overview how such a modelling can be done and provide definitions for optimality of agents behaviours. We also illuminate the principal difficulties one encounters in the robotic soccer and other multi agent domains. Other properties characterising the robotic soccer testbed are the high dimensional, continuous state space, the huge number of basic commands, huge number of policies and the requirement for cooperative independent Reinforcement Learning. The current Karlsruhe Brainstormers approach tries to tackle the complexity by using moves, which themselves are learned by RL methods and define the skills of an agent. To deal with the multi-agent aspects, we pursue both the investigation of theoretically founded distributed RL algorithms plus the empirical/ heuristically motivated way of modified single-agent Q-learning. We report very promising results in using learning of a coordinated offensive behaviour by 7 attackers against 7 defense players and a goalie. Still a lot of research questions are open, for example the dealing with partial observability of state information, the definition of theoretically founded and efficient distributed learning algorithms, the ideal representation of the value function, the search for appropriate features and the theoretical justification to use them, the automatic finding of appropriate moves.

The current version is an intermediate step within our Brainstormers' concept of a learning agent. The final goal is to have an agent, which has learned its fundamental decision behaviour by reinforcement learning. However, until then a lot of work has to be done in the field of multi-agent RL, on Semi- Markov Decision Processes, partially observable domains (POMDPs) and on large-scale RL problems. Some of very recent RL ideas have already been successfully realized. For example, the moves-concept is closely related to Sutton's et.al 'options'-framework [SPS99]. Therefore our work can be regarded as realizing and testing some conceptual ideas in a practical environment.

5.2 Acknowledgements

We would like to thank our students (A. Hoffman, A. Sinner, O. Thate) for their active involvement in the development of our competition team 'Karlsruhe Brainstormers'. Also many thanks go to former members of our development team (S.Buck, S. Dilger, R. Ehrmann, D. Meier).

References

[And98] Tomohito Andou. Refinement of soccer agent's position using reinforcement learning. In H. Kitano, editor, *RoboCup-97: Robot Soccer World Cup I.* Springer Buch Verlag, 1998.

[BHW98] H.-D. Burkhard, M. Hannebauer, and J. Wendler. Belief-desire-intention deliberation in artificial soccer. *AI Magazine*, 19(3):87–93, 1998.

[BSW89] A. G. Barto, R. S. Sutton, and C. J. C. H. Watkins. Learning and sequential decision making. Technical Report COINS TR 89-95, Department of Computer and Information Science, University of Massachusetts, Amherst, September 1989.

[BT96a] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro Dynamic Programming*. Athena Scientific, Belmont, Massachusetts, 1996.

[BT96b] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro Dynamic Programming*. Athena Scientific, Belmont, Massachusetts, 1996.

[BT96c] Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-dynamic programming*. Optimization and neural computation series ; 3. Athena Scientific, 1996.

[CB99] Caroline Claus and Craig Boutilier. The Dynamics of Reinforcement Learning in Cooperative Multiagent Systems. In *IJCAI*, 1999.

[Dor99] Klaus Dorer. Behavior networks for continuous domains using situation-dependent motivations. In *Proceedings of IJCAI '99*, pages 1233–1238, Stockholm, Sweden, 1999.

[FV97] Jerzy Filar and Koos Vrieze. *Competitive Markov decision processes*. Springer, 1997.

[LR00] M. Lauer and M. Riedmiller. An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In *Proceedings of International Conference on Machine Learning, ICML '00*, pages 535–542, Stanford, CA, 2000.

[Luk98] Sean Luke. Genetic programming produced competitive soccer softbot teams for robocup97. In *Proceedings of the Third Annual Genetic Programming Conference (GP98)*, pages 204–222, San Francisco, CA, 1998.

[Mer99] A. Merke. Reinforcement Lernen in Multiagentensystemen. Master's thesis, Universität Karlsruhe, 1999.

[Put94] Martin L. Puterman. *Markov decision processes : discrete stochastic dynamic programming*. Wiley series in probability and mathematical statistics : Applied probability and statistics. Wiley, 1994.

[Rie00] M. Riedmiller. Concepts and facilities of a neural reinforcement learning control architecture for technical process control. *Journal of Neural Computing and Application*, 8:323–338, 2000.

[RMM+00] M. Riedmiller, A. Merke, D. Meier, A. Hoffmann, A. Sinner, O. Thate, C. Kill, and R. Ehrmann. Karlsruhe brainstormers - a reinforcement learning way to robotic soccer. In A. Jennings and P. Stone, editors, *RoboCup-2000: Robot Soccer World Cup IV, LNCS*. Springer, 2000.

[RS98] A. Barto R. Sutton. *Reinforcement Learning*. MIT Press, Cambridge, Massachusetts, 1998.

[SB98] R. S. Sutton and A. G. Barto. *Reinforcement Learning*. MIT Press, Cambridge, MA, 1998.

[SOMB99] Frieder Stolzenburg, Oliver Obst, Jan Murray, and Björn Bremer. Spatial agents implemented in a logical expressible language. In Manuela M. Veloso, editor, *Proceedings of the 3rd International Workshop on RoboCup in Conjunction with 16th Joint International Conference on Artificial Intelligence*, pages 205–210, Stockholm, 1999. IJCAI press.

[SPS99] R. S. Sutton, D. Precup, and S. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 1999. to appear.

[SSS00] Peter Stone, Richard Sutton, and Satinder Singh. Reinforcement learning for 3 vs. 2 keepaway. In P. Stone, T. Balch, and G. Kreatzschmarr, editors, *RoboCup-00: Robot Soccer World Cup IV*. Springer Buch Verlag, 2000.

[SV98a] Peter Stone and Manuela Veloso. A layered approach to learning client behaviours in the robocup soccer server. *Applied Artificial Intelligence*, 12:165–188, 1998.

[SV98b] Peter Stone and Manuela Veloso. Team-partitioned, opaque-transition reinforcement learning. In M. Asada and H. Kitano, editors, *RoboCup-98: Robot Soccer World Cup II*. Springer Buch Verlag, 1998.

[Wat89] C. J. Watkins. *Learning from Delayed Rewards*. Phd thesis, Cambridge University, 1989.

[WD92] Christopher J.C.H Watkins and Peter Dean. Technical Note: Q-Learning. *Machine Leaning*, 8:279–292, 1992.

[Woo99] Michael Woolridge. Intelligent agents. In Gerhard Weiss, editor, *Multi Agent Systems*. MIT Press, 1999.