Proceedings of the 2007 IEEE Symposium on
Computational Intelligence and Games (CIG 2007)

On Experiences in a Complex and Competitive Gaming Domain: Reinforcement Learning Meets RoboCup

Martin Riedmiller and Thomas Gabel
Neuroinformatics Group
Department of Mathematics and Computer Science
Institute of Cognitive Science
University of Osnabrück
49069 Osnabrück, Germany
Email: martin.riedmiller@uos.de, thomas.gabel@uos.de

Abstract— **RoboCup soccer simulation features the challenges of a fully distributed multi-agent domain with continuous state and action spaces, partial observability, as well as noisy perception and action execution. While the application of machine learning techniques in this domain represents a promising idea in itself, the competitive character of RoboCup also evokes the desire to head for the development of learning algorithms that are more than just a proof of concept. In this paper, we report on our experiences and achievements in applying Reinforcement Learning (RL) methods in the scope of our Brainstormers competition team within the Simulation League of RoboCup during the past years.**

Keywords— **reinforcement learning, neural networks, robotic soccer simulation, RoboCup, single- and multi-agent learning**

I. Introduction

The RoboCup robotic soccer initiative was founded to establish a fair and competitive testbed for the development of intelligent and autonomous agent designs [1]. The simulation league in particular aims at the development of intelligent control architectures that are able to tackle questions of both agent individual skills and team cooperation behavior.

From its very first beginnings in 1998, the aim of the *Brainstormers* project was to develop machine learning techniques for a competitive soccer playing robot. In particular, Reinforcement Learning methods [2], that are able to autonomously learn from the only information of success or failure, are in the center of our interest. However, the challenges of such a complex domain as robotic soccer are far beyond the problems typically tackled by Reinforcement Learning methods: continuous states, large amount of actions, considerably delayed rewards, and the requirement of a highly competitive solution.

Of course, solving such a complex problem needs some external structure. In our case, we distinguish between individual skills (like intercepting a ball, going to a certain position, or kicking) and team behavior skills (like for example playing a coordinated attack). Starting with the application of neural Reinforcement Learning methods to individual skills, we meanwhile developed a wide range of learned behaviors that have been or still are actively used within our competition agent. Of course, being actually competitive requires much more than a simple 'proof of concept': learned behaviors must be continuously evaluated against alternative solutions. This leads to an overall architecture, where hand-coded and learned modules work together side by side and, moreover, have to be integrated into a harmonized overall concept.

Our effort has been accompanied by repeated placings among the top-three teams, including the World Champion title at RoboCup 2005 in Osaka. The paper at hand gives an overview over approaches and experiences with Reinforcement Learning methods applied in our Brainstormers competition team within the Simulation League of RoboCup during the years 2000 to 2006. RoboCup is an extremely competitive domain where rapid progress is sometimes achieved within very little time. Accordingly, for a solution based on artificial/computational intelligence to prevail in such an environment, continuous further-developments and performance monitoring are required. Thus, as far as game competitions are considered, a CI-based approach leading to a drop-out in a preliminary round may be interesting from a scientific point of view, but not likely to reside further in a competition team. So, our depictions in the following also point into that direction.

The investigation of machine learning, and Reinforcement Learning methods in the context of robotic soccer depicts an active research area. For example, in the simulation league Stone & Veloso [3] proposed a hierarchical learning paradigm, layered learning. More recently, keepaway soccer, as a special task within soccer simulation, has been established as a benchmark for machine learning [4]. Focusing on agent role assignments, Kok et al. [5] investigate context-specific decompositions of certain learning tasks into smaller sub-problems using coordination graphs that enable an agent to predict the actions of other agents. Also in other RoboCup leagues, such as the MidSize league, the usefulness of Reinforcement Learning methods for acquiring competitive behavior is explored (see, for example [6] and [7]).

1-4244-0709-5/07/$20.00 ©2007 IEEE

II. Reinforcement Learning in Simulated Soccer

Soccer simulation [8] represents the league of the RoboCup robotic soccer initiative where the investigation and application of computational intelligence approaches is most prevalent. The robotic Soccer Simulation (2D) environment of RoboCup features the challenges of a fully distributed multi-agent domain and includes partial observability as well as noisy perception and action execution: Here, the Soccer Server [9], a real-time soccer simulation system, allows autonomous software agents written in an arbitrary programming language to play soccer in a client/server-based style: The server simulates the playing field, communication, the environment and its dynamics, while the clients – eleven agents per team – are permitted to send their intended actions (e.g. a parameterized kick or dash command) once per simulation cycle to the server via UDP. Then, the server takes all agents' actions into account, computes the subsequent world state and provides all agents with (partial) information about their environment via appropriate messages over UDP. The course of action during a match can be visualized using an additional program, the Soccer Monitor (Figure 1).

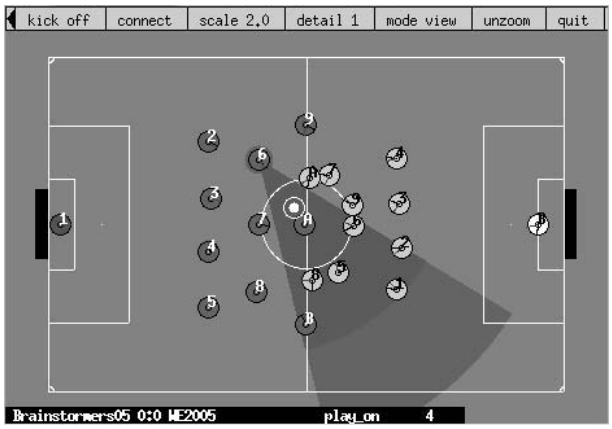

Fig. 1. Screenshot of a match in RoboCup's Soccer Simulation 2D, visualized by the Soccer Monitor. The partial view of player 6 is highlighted.

With our team, Brainstormers, we have been participating in the annual RoboCup tournaments since 1999, focusing our main research effort on realizing a substantial part of the soccer-playing agents' capabilities by applying machine learning and Reinforcement Learning methods. A thorough description of our agents' architecture is beyond the scope of this paper and can be found elsewhere [10]. We note, however, that it is inspired by behavior-based robot architectures: Behaviors of low abstraction level are responsible for basic player capabilities (also termed *skills*) like kicking or ball interception, whereas high-level behaviors are relevant for team-play and strategic decision making. We succeeded in successfully applying Reinforcement Learning approaches to the less as well as to the more abstract behaviors. By also deploying the results of learning during competitions we could prove the case for the scalability of Reinforcement Learning toward RoboCup soccer competitions.

As follows, we exemplarily consider two representatives of learned player behaviors: First, we focus on the difficulties when using Reinforcement Learning to learn the soccer skills (ball interception and ball kicking), second, we examine the use of Reinforcement Learning for team cooperation. Further details on the learning techniques we pursued can be found in related papers describing our team, e.g. [11], [12], [10].

A. Neural Individual Soccer Skills

The Brainstormers' approach to skill learning—no matter if learning to go to a specified position, to kick the ball, to intercept a ball, to do dribble, or another skill—is to model the environment as a Markov decision process (MDP, [13]). An MDP is a 4-tuple $M = [S, A, r, p]$ where S denotes the set of environmental states, A the set of actions the agent can perform, and $r : S \times A \to \mathbb{R}$ the function of immediate rewards $r(s, a)$ (sometimes the notion of costs is used which correspond to negative rewards) that arise when taking action a in state s. The function $p : S \times A \times S \to [0; 1]$ depicts a probability distribution $p(s, a, s')$ that tells how likely it is to end up in state s', when performing action a in state s.

Given an MDP we apply model-based temporal difference Reinforcement Learning using function approximation based on multi-layer perceptron neural networks. In the intercept task example, the formalization as an MDP comprises a continuous, 6-dimensional state space $S = \{s = (v_{bx}, v_{by}, v_{px}, v_{py}, d_{bp}, \alpha_{bp})\}$ where $\vec{v}_b$ is the ball's and $\vec{v}_p$ the player's velocity, d_{bp} the distance and α_{bp} the relative angle between ball and player. Actions for the player are, as determined by the Soccer Server, parameterized turns and dashs. After successful interception, the player gets a positive reward, to create time-optimal behavior each action incurs a little negative reward. To name another learned example skill, the learning of optimal kicking involves a 5-dimensional state space $S = \{s = (v_{bx}, v_{by}, d_{bp}, \alpha_{bp}, \alpha_{pt})\}$ where $\vec{v}_b$, d_{bp}, and α_{bp} are as before and where α_{pt} describes the relative angle between ball and target kick direction from the agent's point of view. Here, the range of possible actions is particularly large since the player may not just take turns (parametrized with one parameter), but also perform kicks which are parameterized by two parameters (kick intensity and kick direction). To handle real-valued action parameters we apply straightforward discretizations (see Table II for the number of discretized actions considered). If the resulting kick fulfills certain quality criteria (ball leaves the area around the player where the ball is kickable, the so-called "kickable area", with the specified velocity and going into the specified target direction), the corresponding episode is considered successful. Otherwise, or if the ball collides with the player, the episode is a failure resulting in a large negative reward.

In search of an optimal behavior in an unknown environment, the agent must differentiate between the desirability of possible successor states, in order to decide for a good action.

Since the transition model in the soccer domain is known[1] we can compute a state value function $V^\pi : S \rightarrow \mathbb{R}$ that estimates the future rewards that can be expected when starting in a specific state s and taking actions determined by policy $\pi : S \rightarrow A$ from then on. Thus, it holds

$$V^\pi(s) = E[\sum_{t=0}^{\infty} r(s_t, \pi(s_t)|s_0 = s)]$$

where $E[\cdot]$ denotes the expected value. If we are in possession of an 'optimal' state value function $V^\star$, it is easy to infer the corresponding optimal behavior policy by exploiting that value function greedily according to

$$\pi^\star(s) := \arg\max_{a \in A}\{r(s,a) + \sum_{s \in S} p(s,a,s') \cdot V^\star(s')\}$$

Temporal difference (TD) methods comprise a set of Reinforcement Learning algorithms that incrementally update state value functions $V(s)$ after each transition (from state s to s') the agent has gone through. This is particularly useful when learning along trajectories $(s_0, s_1, \ldots, s_N)$ starting in some state s_0 and ending up in some terminal state $s_N \in G$. Here, learning can be performed online, i.e. the processes of collecting (simulated) experience and learning the value function run in parallel. For learning to intercept the ball we made use of the $TD(1)$ algorithm where the new estimate for $V(s_k)$ is calculated as

$$V(s_k) := (1-\alpha) \cdot V(s_k) + \alpha \cdot ret(s_k)$$

with $ret(s_k) = \sum_{j=k}^{N} r(s_k, \pi(s_k))$ indicating the summed rewards following state s_k and α as a decaying learning rate [14]. In order to be able to capture the potentially highly non-linear and partially non-continuous state value function for the respective skill learning problem, we (mainly) employ feed-forward neural networks to approximate V, since those are known to be capable of approximating arbitrarily closely any function $f : S \rightarrow \mathbb{R}$ that is continuous on a bounded set S [15]. For more details on neuro-dynamic programming and approximate RL we refer to [16] and for the actual implementation of the learning algorithm and detailed learning results using different mechanisms to approximate the value function we refer to Gabel & Riedmiller [17].

Learned Ball Interception: Learning to intercept a rolling ball is less straightforward than it may intuitively seem. Smallest deviations in the turn angle of a player's turn action (usually, a ball interception sequence has to be started by an initial turn action) may cause drastic changes in the number of steps required to intercept the ball. Accordingly, the optimal state value function $V^\star$ to be learned is characterized by infinitely many points of discontinuity which are inherently difficult to capture by any function approximation mechanism.

When we started to learn basic soccer skill using Reinforcement Learning, we soon realized a neural behavior for ball interception (NIntercept), that clearly outperformed our former hand-coded interception routine and which was used successfully in RoboCup tournaments until 2003. In that year we also started working on a new analytic, model-based interception technique (MBIntercept). MBIntercept extensively simulates the environment many time steps ahead, therefor it is rather computationally expensive, but is able to find the optimal interception point when the Soccer Server adds no noise to ball and player movements. For an intercept benchmark set that we designed, MBIntercept needs on average 9.73 steps for a ball interception, as opposed to 11.02 steps that NIntercept requires on average (noise-free environment). Since the difference in performance of both methods is quite the same when the Soccer Server's noise is present (as it is during competitions), we switched to using MBIntercept during competitions in 2004.

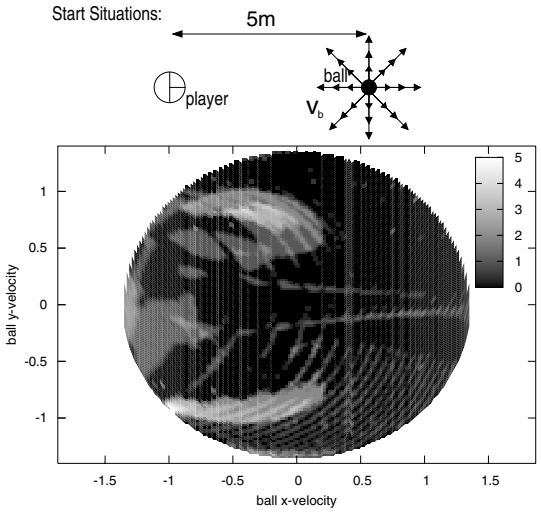

Fig. 2. Quantitative difference in interception capability between the learned and the model-based reference algorithm MBIntercept for a set $S = \{(\vec{v}_b, \vec{v}_p, d_{bp}, \alpha_{bp}) || \vec{v}_b | \in [0, v_{max}], \vec{v}_p = 0, d_{bp} = 5, \alpha_{bp} = 0\}$ of intercept start situations (initially stationary player and varying initial ball velocities, constant distance between ball and player). Bright colors correspond to additional steps required for a ball interception using the learned policy when compared to the optimal solution provided by MBIntercept.

Despite that, in 2005 we started working on the neural ball interception again. Recently, we have proposed a data-efficient neural learning algorithm (Neural Fitted Q Iteration, NFQ, see Riedmiller [18]) that performs off-line (also termed batch-mode) Reinforcement Learning. By employing a modification of NFQ to the intercept task we finally achieved average ball interceptions taking 10.57 steps—that result is visualized in Figure 2. Apparently, the learning agent has particular difficulties to quickly intercept the ball in situations, where the ball is approaching and eventually passing by the player at high speed.

Therefore, to come even closer to the optimum, a further optimization of the learning process was necessary: With a specialized form of adaptive reward shaping [19] and applying an active learning approach [20], we finally arrived at average

[1]During training our agents, we usually turn off the noise generated by the Soccer Server environment. Hence, we know the successor state $s' \in S$, given some state $s \in S$ and action $a \in A$, with $p(s,a,s') = 1.0$.

ball interception times of steps 10.23 per episode (for the test benchmark mentioned above). Although the remaining gap in performance of the learned and hand-coded skill is extremely small, for the competition since 2005 we decided to rely on the latter.

Learned Ball Kicking: The particularities of the Soccer Server environment make it necessary that harder kicks must be composed of a number of elementary kick commands. Thus, to really kick hard, as required for long passes or for scoring, usually an elaborated sequence of n kick commands must be applied, where during the first $n-1$ kicks the ball is moved and accelerated within the player's kickable area and where the nth kick is used to bring the ball to its final desired velocity. Of course, time matters: The shorter the kick sequence, the better, as otherwise opponent players may more easily interfere. The Soccer Server adds a substantial amount of noise to ball movements (up to 5% per time step) and to kicks (up to 10% per kick). Therefore, in what follows, a kick sequence is considered successful, if the resulting kick velocity differs less than $0.2\frac{m}{s}$ from the desired velocity and if the angle of the resulting ball movement differs less than $\frac{\pi}{12}$ from the specified kick target direction.

Our learned kick routine (developed in 2000/2001) outperformed existing heuristic kicking approaches clearly and was not replaced by a classical or analytical solution method during the subsequent years. Despite its reliability and good performance during tournaments we uncovered a weakness of our neural-net based kicking behavior NKick at the end of 2004: When executing kick sequences to reach a final kick velocity of approximately 75% of the maximal kick velocity[2], NKick featured slightly reduced performance. To counteract we re-implemented the kicking learning algorithm making use of $TD(1)$ Reinforcement Learning and applying the same learning methodology as in the case of learning to intercept a ball (see above). The resulting kick behavior relies on five neural networks that are specialized to different target kick velocities. Testing our learning results we confirmed that the performance gap mentioned could be closed while at the same time the kick accuracy could be increased by up to 6% (see Figure 3).

B. Reinforcement Learning for Team Cooperation

Applying Reinforcement Learning to multi-agent systems is problematic due to several nasty properties of multi-agent domains: exponential growth of the decision problem with increasing number of acting agents, the increasing number of state dimensions, the requirement for distributed and individual acting without communication (fortunately, at least during the learning phase, we have the possibility of agent communication here, making the learning problem easier at least in that respect).

Our first approach was a model-free one, where we used a variant of Q-learning [21]. Though this was quite successful

[2]Given current standard parameter settings of the Soccer Server, the maximal ball velocity, and hence the maximal velocity a kicked ball may reach, is $2.7\frac{m}{s}$.

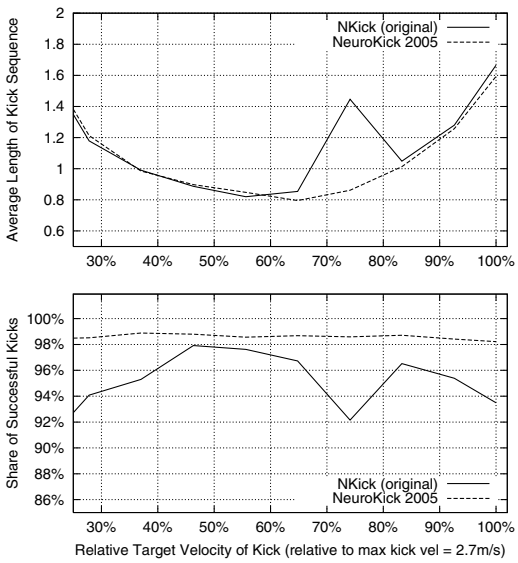

Fig. 3. The charts oppose the kicking capabilities of our learned NKick behavior (used 2000-04) and the re-learned neural kicking routine (used 2005-06) for a test suite of 5000 situations starting from which the agent had to perform kicks with different target velocities.

for a limited number of agents (they even learned double passes), it did not scale to more than 2 teammates vs. 2 opponents.

We felt that for scaling to situations relevant for competition, we should necessarily provide any knowledge that we possibly have. So one of our ideas was to make use of an (approximate) model that predicts the outcome of a certain action. Of course, in the multi-agent domain faced here, we do not have proper knowledge of such a model because:

- we do not have the probabilities with which the actions result in a certain outcome,
- we do not know, what our teammates do (no communication allowed), and
- we do not know, what our opponents do (this differs from team to team).

To deal with the second and third point, we either worked with computing worst-case/best-case scenarios (comparable to max-min search) or—for efficiency reasons—simply assume that other agents do not act at all. To deal with the unknown transition probabilities we employed the following simplification: First, for every possible action, we compute an estimation of its safety. If it is above some (very high) threshold, we consider this action as successful. The resulting state is the one, that will result from a successful application of the action. For example, if we consider a pass to a certain teammate, we first verify that the teammate is the first player to the ball (pass is safe) and then we compute, where he will get the ball. Actions not considered successful are not regarded for selection at all. Note, that the model we use is now quasi-deterministic—but at the same time only a (rough) guess to the actual resulting situation.

Decision-making works as follows. For every situation:

1) compute all potentially successful actions
2) compute all (approximate) resulting states
3) evaluate all resulting states
4) select action with best resulting state

To estimate the value of a state, a neural multi-layer perceptron is used. The number of inputs corresponds to two times the number of teammates + the number of opponents (x and y coordinate) plus 4 inputs for ball position/velocity. It has a certain number of hidden neurons (we mostly used 10 neurons, but the number was not particularly important) and one output neuron—which is the evaluation of the state at input.

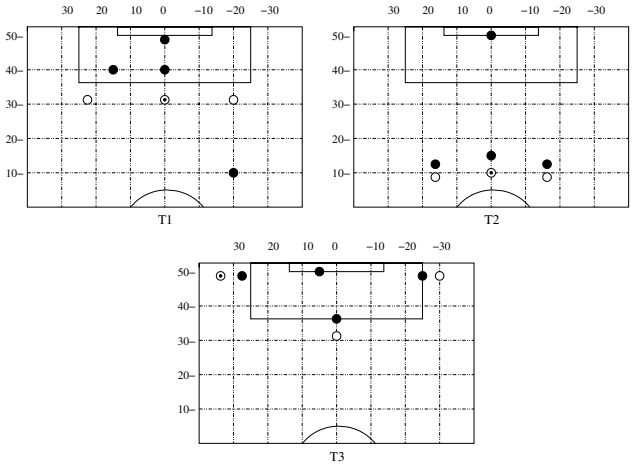

Fig. 4. Initial position of 3 vs 4 attack. Dotted player owns the ball.

Learning is done by making the agents act greedily with respect to their current neural value function. All episodes are recorded (player positions and ball position and velocities). All experiences are added to an 'experience set' E with one exception: episodes that result in a ball loss are not recorded (since this is not the fault of the learning module, as we assume, that only successful actions are provided). After some time of experience collection (e.g. until a certain number of successful episodes was performed), the experience set is evaluated and respective target values are computed. This corresponds to a policy evaluation step. The optimization goal for the dynamic programming part is the minimization of expected time until success, therefore every decision implies small constant transition costs ($c_{trans} = 0.01$), whereas a goal is 'rewarded' with costs 0 and an undesired 'stuck' situation is punished with costs of 1.

To train the neural network, the fast learning procedure Rprop [22] is used. After training (typically for 5000 epochs), the net is distributed to all agents, which restart to sample new experience with the new value function. In its pure form, this corresponds to a policy iteration step which necessarily leads to an improvement of the policy until the optimal policy is found. Due to the approximations our approach comprises the theoretically guaranteed improvement of the policy during each iteration may no longer be effective. However, as shown in Table I quite effective cooperative policies can be learned despite all approximations. The results, which were mainly achieved in 2001 and 2002, show drastic improvements over our attack in our strong 2000 competition team, which was the runner-up in the 2000 RoboCup world championship.

TABLE I

PERFORMANCE OF LEARNED ATTACK AND HAND-CODED ATTACK OF OUR 2000 COMPETITION TEAM AGAINST A POWERFUL DEFENSE. THE DEFENSE POLICY IS DIFFERENT FROM THE ONE USED FOR TRAINING AGAINST. THE TABLE SHOWS BOTH LEARNED SITUATIONS (L1 - L3) AND NEW SITUATIONS (T1 - T3).

	RL attack		BS 2000 attack	
	goal	stuck	goal	stuck
L1	0.645	0.03	0.0	0.97
L2	0.225	0.145	0.01	0.505
L3	0.45	0.04	0.0	0.965
T1	0.655	0.01	0.31	0.205
T2	0.39	0.035	0.14	0.0
T3	0.445	0.05	0.145	0.415

Actions considered on this multi-agent level are made up of individual skills: intercepting, going to one of 8 positions, dribbling, passing to a teammate, scoring. For the competition team several further details are incorporated: some decisions are pre-wired (e.g. the fastest player intercepts the ball) and constraints are formulated (e.g. to prevent the players from running away too far or to crash into each other). The final neural attack module which we applied in our competition team considered 7 attackers and 8 opponents.

III. OVERVIEW

Over the years a considerable part of decision making within our Brainstormers agent has been solved by neural Reinforcement Learning methods. One of the earliest successes was the learning of a powerful neural kicking routine. At that time (in 2000), this was a real breakthrough since good kicking requires an elaborated sequence of appropriate basic commands. Many teams developed heuristics to solve the problem, but our neural kicking (NKick) routine not only kicked more reliably, but also did this in a minimum number of time steps. Additionally, NKick was not designed at all, but completely learned its behavior from scratch—by the experience of success or failure.

Most of the individual skills were developed for our 2000 competition team, and many of them were refined and re-trained in the following years. E.g. the neural kicking routine initially used 54 neural nets, which could be reduced to 5 neural nets by exploiting symmetries in subsequent versions.

At the time of their development, all of the learned skills worked considerably better than the hand-coded routines we had so far. This was the reason why they made their way into our competition team. However, during years of tough competition, many new ideas for solving tasks come up, and although

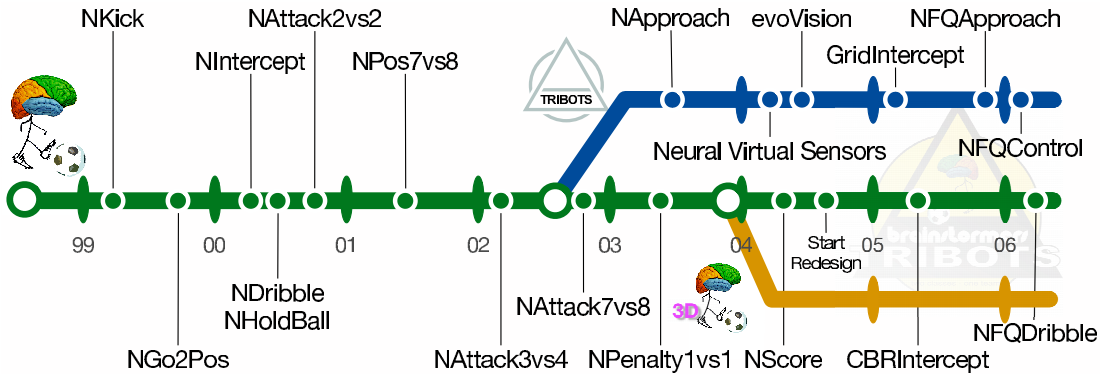

Fig. 5. This time bar shows the milestones in the development of learned behaviors in the Brainstormers' teams participating in the 2D and 3D simulation league as well as in the MidSize-League (Brainstormers Tribots).

the learned skills still perform very well, some of them had to be replaced by improved hand-coded skills (like e.g. the neural intercept routine [20]). Here, the gap between the learned skill and the fine-tuned hand-coded skill is considerably small (less than half of a decision cycle in average), but for competition, this already can make a difference.

The peak amount of learned skills used in the competition team was in the years 2001–2004, where we used many learned individual skills and also had the complete (cooperative multi-agent) attack behavior learned by neural Reinforcement Learning methods. The network guiding our attack play in 2004/6 for instance had 34 continuous inputs (denoting 7 teammates' positions, 8 opponents' positions, and ball position and velocity). After having been so close to the title so many times, without ever getting it, we started a redesign of our agent in 2004 and 2005. Some of the learned skills were replaced by novel, more effective hand-coded skills, other capabilities (like e.g. the multi-agent attack play) were temporarily suspended in our competition team, since they did not match exactly any more the new design. Still, they work and perform pretty well (e.g. the Brainstormers NeuroAgent 2005 can clearly beat the Brainstormers Agent 2004[3]), but to be really competitive, a complete re-learning, adapted to the new design, was needed. Continuing this re-learning while achieving further improvements of the competition agent is currently ongoing work.

Table II provides a comprehensive overview on the Brainstormers' behaviors learned by neural Reinforcement Learning methods. The upper part of the table shows the individual skills, the lower part shows the multi-agent skills. Filled circles ('•') denote the years, where the learned skill was actually used in the Brainstormers competition team at the world championships of RoboCup. Empty circles ('○') denote the years, where a certain skill was developed or improved, but not used in the competition team. The state space dimensions and action space cardinalities of the problems show that the tasks to be learned are far beyond trivial. The final row shows the ranking. The Brainstormers were always among the best three teams of the world during the last 7 years.

[3]A neural net is involved in decision-making of a NeuroAgent 2005 defender / sweeper / midfielder / attacker on average in 56.8%/73.0%/84.4%/82.6% of its total number of actions. The average score against the BS 2004 agent is 2.58:0.33.

TABLE II
OVERVIEW ON BEHAVIORS LEARNED BY NEURAL REINFORCEMENT LEARNING METHODS (SEE THE TEXT FOR EXPLANATIONS).

	dim(S)	card(A)	2000	2001	2002	2003	2004	2005	2006
NKick	5	1204	•	•	•	•	•	•	•
NIntercept	6	76	•	•	•	•		○	○
NGo2Pos	6	76	•	•	•	•	•		
NDribble	11	282	○						○
NHoldBall	8	360	•	•					
NAttack2vs2	14	13	○						
NPos7vs8	34	10		•					
NAttack3vs4	18	14			○				
NAttack7vs8	34	18			•	•	•		•
NPenalty1vs1	8	11				•	•	•	•
NScore	18	14					•		
Rank			2	2	3	3	2	1	2

Demo videos of the learned behaviors and the learning process can be found at our website: **www.ni.uos.de/brainstormers**. At the end of 2005 we also released our team's source code, including numerous high- and low-level behaviors, numerous learned behaviors using Reinforcement Learning and the corresponding value functions represented by neural networks, as well as accompanying libraries. That source code is made publicly available under the terms of the GNU General Public License (GPL) and can be retrieved from our website, too.

IV. CONCLUDING REMARKS

The aim of the Brainstormers project is to show the usefulness of machine learning techniques, especially neural Reinforcement Learning methods, in a highly complex, dynamic, and competitive domain. Ever since the beginning, we were eager not only to show that Reinforcement Learning

methods work in principle, but we were always aiming at their actual use in our competition team at places very essential for success. At the time of their introduction into the team, all the learned skills improved significantly over previously used hand-coded behaviors. In the course of time, some of the learned skills are meanwhile again replaced by hand-coded routines. However, we do not regard this as a step back, but as a natural development in a competitive environment where sometimes one approach is superior and sometimes another.

The tasks solved within our competition team are far beyond the benchmark problems typically regarded in Reinforcement Learning: Here, we have a considerable amount of input dimensions (from 5 to 34), the inputs are continuous, the number of actions is high (10 to more than 1000), the distance to the rewarding goal (with respect to the number of cycles) is considerably large, and optimality of the solution really matters.

In 2002, we started a MidSize team, the Brainstormers Tribots (see Figure 5), that shall stress the usefulness of learning techniques in a real robot team. Currently, a lot of interesting learning tasks have already been solved on the real robot (see [7] and [23]). It is one of our mid-term goals to actually employ them in the Brainstormers Tribots' competition team.

To summarize, successful application of Reinforcement Learning in competitive domains requires

- to find the points, where Reinforcement Learning can be fruitfully applied: better, faster, with less effort,
- to find the right level of abstraction: difficulty of learning vs. loss of optimality, as well as
- to integrate hand-coded and learned modules as suitably as possible.

REFERENCES

[1] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, E. Osawa, and H. Matsubara, "RoboCup: A Challenge Problem for AI," *AI Magazine*, vol. 18, no. 1, pp. 73–85, 1997.

[2] R. Sutton and A. Barto, *Reinforcement Learning. An Introduction*. Cambridge, USA: MIT Press/A Bradford Book, 1998.

[3] P. Stone and M. Veloso, "Layered Learning," in *Machine Learning: ECML 2000. Proceedings of the 11th European Conference on Machine Learning*. Barcelona, Spain: Springer, 2000, pp. 369–381.

[4] P. Stone, R. Sutton, and G. Kuhlmann, "Reinforcement Learning for RoboCup-Soccer Keepaway," *Adaptive Behavior*, vol. 13, no. 3, pp. 165–188, 2005.

[5] J. Kok, M. Spaan, and N. Vlassis, "Non-Communicative Multi-Robot Coordination in Dynamic Environments," *Robotics and Autonomous Systems*, vol. 50, no. 2-3, pp. 99–114, 2005.

[6] Y.Takahashi, K. Edazawa, K. Noma, and M. Asada, "Simultaneous Learning to Acquire Competitive Behaviors in Multi-Agent System based on a Modular Learning System," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2005)*. IEEE Computer Society, 2005, pp. 153–159.

[7] T. Gabel, R. Hafner, S. Lange, M. Lauer, and M. Riedmiller, "Bridging the Gap: Learning in the RoboCup Simulation and Midsize League," in *Proceedings of the 7th Portuguese Conference on Automatic Control (Controlo 2006)*. Porto, Portugal: Portuguese Society of Automatic Control, 2006.

[8] M. Veloso, T. Balch, and P. Stone, "RoboCup 2001: The Fifth Robotic Soccer World Championships," *AI Magazine*, vol. 1, no. 23, pp. 55–68, 2002.

[9] I. Noda, H. Matsubara, K. Hiraki, and I. Frank, "Soccer Server: A Tool for Research on Multi-Agent Systems," *Applied Artificial Intelligence*, vol. 12, no. 2-3, pp. 233–250, 1998.

[10] M. Riedmiller, A. Merke, and W. Nowak, "Brainstormers 2003—Team Description," in *RoboCup 2003: Robot Soccer World Cup VII, LNCS*. Springer, 2003.

[11] A. Merke and M. Riedmiller, "Karlsruhe Brainstormers—A Reinforcement Learning Way to Robotic Soccer II," in *RoboCup-2001, LNCS*. Springer, 2001.

[12] M. Riedmiller, T. Gabel, J. Knabe, and H. Strasdat, "Brainstormers 2D—Team Description 2005," in *RoboCup 2005: Robot Soccer World Cup IX (CD)*. Springer, 2005.

[13] M. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. USA: Wiley-Interscience, 2005.

[14] R. S. Sutton, "Learning to Predict by the Methods of Temporal Differences," *Machine Learning*, vol. 3, pp. 9–44, 1988.

[15] K. Hornick, M. Stinchcombe, and H. White, "Multilayer Feedforward Networks Are Universal Approximators," *Neural Networks*, vol. 2, pp. 359–366, 1989.

[16] D. Bertsekas and J. Tsitsiklis, *Neuro Dynamic Programming*. Belmont, USA: Athena Scientific, 1996.

[17] T. Gabel and M. Riedmiller, "CBR for State Value Function Approximation in Reinforcement Learning," in *Proceedings of the 6th International Conference on Case-Based Reasoning (ICCBR 2005)*. Chicago: Springer, 2005, pp. 206–221.

[18] M. Riedmiller, "Neural Fitted Q Iteration—First Experiences with a Data Efficient Neural Reinforcement Learning Method," in *Machine Learning: ECML 2005. Proceedings of the 16th European Conference on Machine Learning*. Porto, Portugal: Springer, 2005.

[19] A. Y. Ng, D. Harada, and S. J. Russell, "Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping," in *Proceedings of the 16th International Conference on Machine Learning (ICML 1999)*, I. Bratko and S. Dzeroski, Eds. Bled, Slovenia: Morgan Kaufmann, 1999, pp. 278–287.

[20] T. Gabel and M. Riedmiller, "Learning a Partial Behavior for a Competitive Robotic Soccer Agent," *KI Zeitschrift*, vol. 20, no. 2.

[21] C. Watkins and P. Dayan, "Q-Learning," *Machine Learning*, vol. 8, pp. 279–292, 1992.

[22] M. Riedmiller and H. Braun, "A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm," in *Proceedings of the International Conference on Neural Networks (ICNN 1993)*, San Francisco, USA, 1993, pp. 586–591.

[23] M. Lauer, "Ego-Motion Estimation and Collision Detection for Omnidirectional Robots," in *In RoboCup 2006: Robot Soccer World Cup X, LNCS*. Bremen, Germany: Springer, 2006.