Learning a Partial Behavior for a Competitive Robotic Soccer Agent

Thomas Gabel, Martin Riedmiller

Neuroinformatics Group, University of Osnabrück, 49069 Osnabrück

Robotic soccer is a highly competitive domain. Accordingly, the use of learnt behaviors in this application field presumes not only learning algorithms that are known to converge and produce stable results, but also imposes the wish for obtaining optimal or at least near-optimal behaviors, even when working within high-dimensional and continuous state/action spaces. This paper deals with the continuous amelioration of adaptive soccer playing skills in robotic soccer simulation, documenting and presenting results of our hunt for optimal policies. We show that not too much effort is necessary to realize straightforward Reinforcement Learning algorithms in this domain, but that a heavy load of work is required when tweaking them towards competitiveness.

1 Introduction

One of the main reasons for the attractiveness of Reinforcement Learning (RL) approaches is their applicability in unknown environments with unidentified dynamics, where an agent acquires its behavior by repeated interaction with the environment on the basis of success and failure signals. Numerous applications of RL techniques in various scenarios have shown that RL works and can be employed for tasks where finding analytical solutions represents a rather difficult and cumbersome challenge.

In practice, it is often impossible to meet all presumptions for an RL algorithm to converge, especially when trying to learn a policy for a more complex task with a continuous state/action space where the use of a function approximator to represent the value function is indispensable. The use of a function approximator, however, induces some noise, i.e. some generalization error on its output predictions. That noise may, when greedily exploiting a state or state-action value function, lead to an overestimation of the successive state's value and hence prevent the finding of an optimal policy. Our investigations to be presented in the scope of this paper are driven by a goal-oriented search for optimal policies. The motivation behind this works stems from a highly competitive application scenario, robotic soccer simulation, where the availability of optimal or at least near-optimal policies may decide between victory or defeat. The main message we want to convey with this paper is that state of the art RL algorithms can be tuned to work in combination with a function approximator in highly complex domains, but that yielding optimal policies still represents a laborious problem.

In Section 2 we motivate our work and present the intercept ball task, one of the most important fundamental skills required in robotic soccer simulation, which we will use as a benchmark throughout this paper. Section 3 discusses existing (adaptive as well as non-adaptive) algorithms to solve that problem. In Section 4 we report on our steps towards a learnt, near-optimal policy for ball interception. In so doing, we focus on the need for function approximation, introduce a number of techniques to improve the learning algorithm, and also point out to inherent difficulties a learning agent faces in the soccer simulation environment.

2 Brainstormers Approach

During the past years, robotic soccer has turned out to be an excellent test-bed for machine learning and, in particular, for Reinforcement Learning tasks. RoboCup is a research initiative to advance AI and intelligent robotics research. Annually, there are championship tournaments in several leagues – ranging from real soccer-playing robots to simulated ones. The context for this paper is RoboCup's 2D Simulation League, where two teams of simulated soccer-playing agents compete against one another using the Soccer Server [10], a real-time soccer simulation system.

With our competition team, *Brainstormers*, we have been participating in the RoboCup [15] championship tournaments for several years, whereupon our main research effort is to realize a growing part of the soccer-playing agent's behavior by machine learning techniques. The complexity of the soccer domain, however, makes the learning of optimal policies a challenging task. Often, a skillfully hand-crafted solution can be superior to a learnt one that is implemented as a proof of concept only. Consequently, having achieved a number of successes at international robotic soccer competitions[1], we aim at learning *competitive* behavior policies.

Several research groups have dealt with the task of learning parts of a soccer-playing agent's behavior autonomously [7, 8]. The focus of this paper is on learning a soccer player's basic behaviors, its so-called *skills*. One of the most important fundamental capabilities of a soccer player is to intercept a running ball as quickly as possible. Since a match's course of action can be influenced significantly, if a team is in ball possession, this skill is crucial for being competitive.

[1] e.g., multiple World Vice Champion titles and World Champion at RoboCup 2005 in Osaka

Intercept Ball Task

The start situation for the task of ball interception is represented by a player moving at velocity $\vec{v_p}$, and the ball at a distance of d_{bp} moving at velocity $\vec{v_b}$. Further, the relative angle between the player's orientation and the ball is denoted as α_{bp}. The environment determines that both the player's and ball's velocity decay at a certain rate from one time step to the next. The player is permitted to decide for one action per time step sending it to the Soccer Server. To approach and intercept the ball the following actions are of relevance: Turn actions, $turn(x)$, with $x \in (-180°, 180°]$, change the player's orientation by an angle x. Dash actions, $dash(y)$, with $y \in [-100, 100]$, generate an acceleration along the player's current orientation. The action's parameter y specifies the relative dash power.

The player's task is to find a sequence of actions which enables him to intercept the ball as quickly as possible: The agent must approach the ball so that it is within the player's kickable area – a region around the player in which he has control over the ball. At a first glance the solution to the task at hand appears trivial: Simply compute the best interception point, use a turn command to aim into that direction, and dash at maximum speed towards that point. Unfortunately, the choice of the word "simply" is inappropriate. On the one hand, the Soccer Server's simulation is time discrete so a closed form solution for computing the interception point is not feasible without compromise in abstracting the environment. On the other hand, one may imitate the server and simulate the environment for a number of time steps ahead. However, this is computationally costly and does not rid us of the crucial question under which circumstances turn actions are to be preferred over dashes. Further, the simulation environment is superposed with a certain level of noise created by the Soccer Server so that a running ball in general does not move along a line.

The crucial factor for intercepting a ball really quickly is the correct decision for turn actions. However, marginal deviations in the turn angle may make further turns necessary, which are "expensive" in terms of interception time as no dash action can be performed concurrently and hence the player slows down. Figure 1 (left) illustrates the problem: At $t = 0$ the player is stationary with a body orientation towards the right, while the ball moves with $|\vec{v_b}| = 1.5$. Obviously, the first action required is a $turn(\alpha)$. The figure shows that the ball can be intercepted within 3 time steps when choosing $\alpha \in [27°, 34.5°]$. Using turn angles beyond that interval, 4 or more cycles are needed. In the figure's right part a more extreme situation is shown: At $t = 0$ the ball starts with $|\vec{v_b}| = 2.7$. The ball position at $t = 3$ is the same as before, so with $\alpha \in [27°, 34.5°]$ the ball might have been intercepted within 3 steps, too. For an initial $turn(\alpha)$ with $\alpha \in (34.5°, 65°]$ the ball would pass the player and at least 5 steps would be required for ball interception. Choosing α beyond that interval and within $(65°, 81°]$ (like the agent in Figure 1 which decides for $\alpha = 70°$) it is again possible to intercept the ball faster, i.e. within 4 time steps. Discontinuities and non-monotonies like the one described exist throughout the entire space of possible intercept starting situations. So, the particular challenge in solving the intercept problem is represented by a high-dimensional, continuous state space which is covered with discontinuities where small changes in one of the state variables (e.g. ball velocity or player orientation) require very different actions to be chosen by the agent.

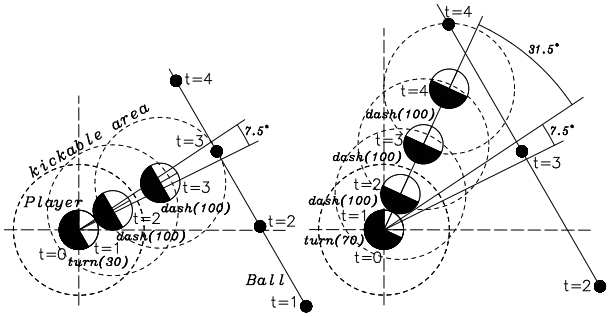

Figure 1: Difficulties in Ball Interception

3 Ball Interception Methods

In its first version our team made use of the ball interception routine from Carnegie Mellon's team CMUnited98 [13]. Intending to apply Reinforcement Learning in a competitive domain like robotic soccer, we soon realized a straightforward RL approach based on value iteration and state value function approximation with a neural network. The resulting behavior, called neuro intercept (NI'02) subsequently, outperformed the original routine clearly and was employed successfully during RoboCup tournaments from 2000 to 2002.

3.1 Numerical Solution Approach

To remain competitive steady improvements to the agents' implementation are required. This also refers to the task of ball interception: Other authors have presented very efficient numerical algorithms for computing the time t it takes a player to intercept a moving ball [12]. They show that there is no closed form for calculating t and therefore make use of Newton's method to numerically find a near-optimal t. However, their algorithm makes two simplifying assumptions about the soccer simulation environment. First, it is presumed that players run at a fixed (maximal) velocity. In practice, players need to accelerate until reaching their maximal speed for a number of time steps and may slow down when, for example, turning. Second, time is assumed to be continuous – which it is not in this environment as the Soccer Server's simulation is done in discrete time steps. Due to these assumptions the algorithm's prediction for intercept times/points is a near-optimal approximation only.

3.2 Model-Based Solution Approach

The time-discrete dynamics of the soccer simulation environment are known [10]. Therefore, it is possible for a player

to predict the next state he will find himself in when having taken a specific action. Based on that knowledge we realized a computationally intensive interception algorithm that simulates the environment, models the results of the player's actions and in so doing finds the optimal interception point (under the assumption the Soccer Server adds no noise to ball and player movements). This hand-coded technique was integrated into our competition team in 2003, replacing algorithm NI'02 and outperforming it by significantly faster ball interception sequences (see also Section 3.3).

As can be seen in Algorithm 1 the main load of this algorithm lies in step $2b$: Here, the player's subsequent decisions as determined by steps 3 and 4 are modelled for a number of time steps ahead to calculate the minimal number of steps required to intercept the ball. If the environment is noise-free, i.e. all ball and player movements are accurately predictable, the intercept point $\vec{p_b}$ can be forecast exactly (parameter ε becomes redundant): The player will head into the optimal intercept direction after its first action. In a noisy environment ε must be set to an appropriate value trading off between exact heading towards the intercept point computed and taking as few additional turn actions as possible.

1. **set** $t := 0$
2. **do**
   (a) **set** $t := t + 1$
   (b) **model** own behavior and ball movement for maximally t time steps
   while ball cannot be intercepted in t steps
3. **set** $\vec{p_b}$:= ball's position in t time steps
   set $\vec{p_p}$:= player's current position
   set $\vec{\alpha_p}$:= current player orientation
   set $\vec{\beta}$:= angle between $\vec{p_b} - \vec{p_p}$ and x-axis
4. **if** $|\alpha_p - \beta| > \varepsilon$
   then $TURN$ towards β
   else $DASH$ at maximal speed

Algorithm 1: Model-Based (MB) Ball Interception

3.3 Reinforcement Learning Approach

We now want to investigate the use of RL for the ball interception problem in more detail, using the accurate solutions provided by algorithm MB as a benchmark.

An RL problem is usually formalized as a Markov Decision Process [1], where an MDP is a 4-tuple $[S, A, r, p]$ with S as the set of states, A the set of actions, and $r : S \times A \to \mathbb{R}$ the function of immediate rewards $r(s, a)$ that arise when taking action a in state s. 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. The intercept task's formalization as an MDP comprises a continuous, 6-dimensional state space $S = \{s = (v_{b,x}, v_{b,y}, v_{p,x}, v_{p,y}, d_{bp}, \alpha_{bp})\}$ 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 turn and dash commands. After successful interception, the player gets a positive reward, to create time-optimal behavior each action incurs a little negative reward.

The agent needs to differentiate between the desirability of successor states, in order to decide for a good action. A common way to rank states is by computing a state value function $V^\pi : S \to \mathbb{R}$ which estimates the future rewards that can be expected when starting in a specific state s and taking actions determined by policy $\pi : S \to A$, i.e. $\pi(s)$, 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 RL 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 $(s_0, s_1, \ldots, s_N)$ – as we do here – starting in some state s_0 and ending up in some terminal state $s_N \in G$. So, learning can be performed online, i.e. the processes of collecting (simulated) experience[2] and learning the value function run in parallel. In this work we update the value function's estimates according to the $TD(1)$ update rule [14], 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 (see Algorithm 2).

1. initialize value function V arbitrarily, let policy π be given by greedy exploitation of $V =: V^\pi$
2. **repeat**
   (a) generate random start situation s_0 for current episode, set $k := 0$
   (b) **while** $s_k \notin G$ **and** $k < maxEpisLeng$ **do**
      i. choose next action a_k by exploiting V greedily according to $a_k := argmax_{a \in A}(r(s_k, a) + \sum_{s' \in S} p(s_k, a_k, s') \cdot V(s'))$
      ii. perform a_k, entering state s_{k+1} and perceiving immediate reward $r(s_k, a_k)$
   (c) **for all** steps s_k in episode $(s_1, \ldots, s_N)$
      i. $ret(s_k) := \sum_{j=k}^{N-1} r(s_j, a_j) + r(s_N)$
      ii. $V(s_k) := (1 - \alpha) \cdot V(s_k) + \alpha \cdot ret(s_k)$ with α as learning rate
   until stop criterion becomes true

Algorithm 2: Episode-Based TD(1) Learning

Grid- and Memory-Based Value Functions

For the TD algorithm (Algorithm 2) convergence to the optimum in a finite state and action space is guaranteed, if each state's value is updated an infinite number of times and if the step size parameter α diminishes towards zero at a suitable rate. Unfortunately, due to the continuous state space

[2]Exploration is omitted in Algorithm 2, value function updates are made for greedy episode parts, only.

Approach	$10k$	$100k$	$500k$
TD_{grid5k}	70.40 ± 5.34	64.30 ± 5.30	65.21 ± 4.97
$TD_{grid100k}$	60.72 ± 1.83	43.30 ± 2.24	40.55 ± 1.78
$TD_{grid600k}$	68.44 ± 1.97	51.07 ± 1.77	40.19 ± 2.18
TD_{CB500}	46.18 ± 11.2	32.12 ± 5.22	27.93 ± 1.97
TD_{CB2k}	27.88 ± 4.02	22.64 ± 2.16	21.15 ± 1.76
$NI'02$	11.06		
MB	**9.73**		

Table 1: Interception Using a Grid- and Memory-Based Function Approximator: The table summarizes the average number of steps to intercept the ball for a set of 1000 random start situations (noise-free environment). Columns reflect the quality of learnt policies after different numbers of learning episodes experienced. So, the rightmost corresponds to approximately ten million state value backups.

covered by the intercept task, these prerequisites cannot be fulfilled here.

Therefore, in a first attempt we discretized the state space S along each of its dimensions. Choosing different levels of discretization we reduced S to $5k$, $100k$ and $600k$ abstract states distributed nearly equidistantly over S and applied our TD(1) learning algorithm directly. Compared to the reference method MB the results obtained were far from optimal (Table 1, lines 1-3). This is not surprising when considering the complexity of the underlying problem (cf. Section 2) and the presumably complex shape of $V^\star$.

Using a specialized memory-based approach which remembers states and belonging state values explicitly, stores them in a case base CB, uses suitable routines for case base management and thus for disassociating from some of its experience from time to time, and which predicts other state values using k-nearest neighbor regression, we could achieve clear improvements [3]. With much less memory consumption ($|CB| = 2000$) we succeeded in making it half the way to the optimum. Certainly, the numbers summarized in Table 1 (lines 4-5) could even have been improved, if we had increased the amount of stored experience. Yet, it must be admitted that then a real-time application of the system would have been prohibited since the time required for nearest-neighbor retrieval/regression grows at least logarithmically with the number of examples stored. Nevertheless, it is worth noting that using this approach average-quality policies can be generated within very short time, i.e. with a limited number of training episodes.

4 Learning to Be Competitive

Our investigations in the previous section have shown empirically that an RL implementation with straightforward value function approximation, using a grid- and memory-based representation, are insufficient to obtain a competitive intercept policy. In the following, we will present an ensemble of three learning techniques whose interplay enables us to learn a near-optimal ball interception behavior.

4.1 Batch Mode Learning

In Section 2 we have shown that the underlying task's state space is full of regions where small deviations in a chosen turn angle cause drastic changes in achievable ball interception times. In order to be able to generalize and capture the highly non-linear and partially non-continuous state value function present for the problem at hand we are in need of a more powerful function approximation mechanism. Feed-forward neural networks are known to be capable of approximating arbitrarily closely any function $f : S \rightarrow \mathbb{R}$ that is continuous on a bounded set S [6]. As we will argue in the following, the usage of multi-layer perceptrons brings us nearer to the target on our way to an optimal policy and allows us to gain insights into the problem structure.

We perform neural network training in batch mode: Repeatedly a number of training episodes is simulated and in so doing a set of representative states $\tilde{S} \subset S$ is built up where for each $s \in \tilde{S}$ we have an estimated value $V(s)$ as calculated by Algorithm 2. So, our learning algorithm is in the spirit of fitted value iteration [4] and other current off-policy RL approaches (e.g. [2]).

Let the state value function approximation provided by the net be denoted as $\tilde{V}(s, w)$ where w corresponds to a vector of tunable parameters, i.e. the net's connection weights. Then, the actual training means determining w by solving the least squares optimization problem $min_w \sum_{s \in \tilde{S}} (\tilde{V}(s, w) - V(s))^2$. For the minimization we rely on the back-propagation variant *RPROP* [11].

Making use of a neural function approximator with 24 sigmoidal neurons in its single hidden layer (best among several net architectures tested) we got ahead a significant step: After $500k$ training episodes – which corresponds to approximately ten million state value backups – the agent using the learnt intercept policy managed to intercept a ball on average within 10.57 ± 0.08 cycles (using the same test set as above), which tops each of our learnt approaches drastically and also outperforms the old NI'02 algorithm (11.06 steps on average), but fails to reach the reference algorithm MB by 0.84 time steps necessary on average for a ball interception.

4.2 Adaptive Shaping

Having in mind that the ball intercept behavior gained so far features clearly improved performance when compared against the other function approximators tested and knowing that an average intercept sequence takes only 0.84 steps longer than model-based ball interception, the results obtained can be considered as a success and the corresponding policy as quite competitive. In this respect the aim to catch up with the optimal performance provided by algorithm MB represents an even more challenging task.

It is not difficult to figure out, why our learnt behavior needs slightly more steps to approach the ball: It always manages to intercept the ball successfully, but often does not find the most "aggressive" way to the ball and executes too many turns. To combat the problem of "avoidable turns" we now pursue a reward shaping approach [9]: Turn actions shall incur higher immediate costs than dashes. In the soc-

cer simulation context this is not intuitive since here turns are free of charge whereas dash actions reduce the player's stamina. Note, that it is not appropriate to simply derogate turn actions with static immediate costs two or three times as high as for dashes. Doing that would destabilize and slow down the learning process as – apart from explorative actions – almost only dashes would be executed at the beginning of learning (when V is not yet rudimentarily approximated) and thus almost no successful episodes would be collected. Therefore, we make use of an adaptive approach, introduce a turn cost factor (initialized to 1.0) which is steadily increased during learning each time the quality of the policy improves. With this adaptive shaping approach we obtained convincingly good results; average ball interceptions (in the test scenario from above) take 10.23 ± 0.12 time steps.

4.3 Active Learning

Aiming at a further improvement of the policy learnt, we tried to find out in which regions of the state space S the value function is approximated insufficiently and directly compared the learnt intercept behavior to our reference method MB for selected parts of S. We recognized that during learning those regions of S are particularly difficult to capture in which – at the moment of (optimal) ball interception – the ball's and the player's velocity vectors are nearly orthogonal and high in velocity. Figure 2 covers a subset $\tilde{S} = \{v_{b,x}, v_{b,y}, 0, 0, 5, 0 | v_{b,x}, v_{b,y} \in [\frac{-v_{b,max}}{2}, \frac{v_{b,max}}{2}]\}$ of S and shows the performance of the learnt policy versus model-based interception. Obviously, for balls that approach the player ($v_{b,x} \ll 0$) and do not move directly towards him ($|v_{b,y}| \gg 0$) algorithm MB outperforms the learnt policy evidently: White regions indicate that MB needed less steps to intercept the ball than the learnt policy, whereas black regions indicate equal performance.

With this knowledge it is plausible to continue the training with focus on problematic parts of S using a non-equal distribution of start situations. Let t denote the number of time steps required for interception as predicted by algorithm MB, $\vec{v_b}(t)$ the ball's and $\vec{v_p}(t)$ the player's velocity at the point/moment of interception. Then, the area of the parallelogram spanned by these vectors

$$A = sin\left(acos\left(\frac{\vec{v_b}(t) \cdot \vec{v_p}(t)}{|\vec{v_b}(t)||\vec{v_p}(t)|}\right)\right)|\vec{v_b}(t)||\vec{v_p}(t)|$$

is a measure for the difficulty of the interception situation. With this measure it is straightforward to preferably generate difficult intercept start situations and learn on the basis of those. The right part of Figure 2 shows the resulting distribution of generated situations for the scenario using $\tilde{S}$.

This approach may be easily extended to an active learning algorithm [5], where the neural network reflects about its competence, knows about the areas of the input space where its approximation is insufficient, and tries to improve its performance especially there. Closing that active learning loop and enabling fully automatic amelioration is left for future work. We here report on results only, we obtained using the manually adapted start situation generation described above. After $500k$ further training episodes with the optimized start situation generation procedure average interception times could be reduced to 10.01 ± 0.05 for the evaluation setting used throughout this paper.

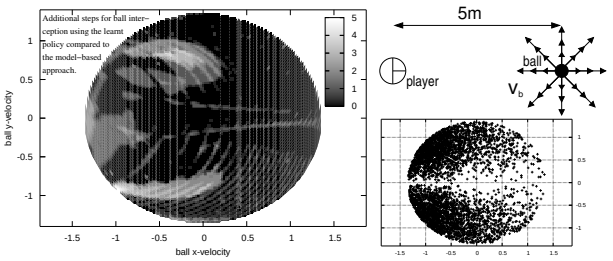

Figure 2: Quantitative Difference in Interception Capability between the Learnt and the Model-Based (MB) Approach for a Specific Set of Start Situations (left) and Adapted Start Situation Distribution (right)

4.4 Discussion

Ball interception on the basis of the learnt behavior takes on average 0.28 steps longer than the theoretical optimum in a noise-free environment[3]. So, we are in possession of a high-performance intercept behavior as this implies that the agent "loses" only one cycle within about thirty compared to model-based interception. We also clearly outperformed our previous learnt intercept routine NI'02.

We have indicated the difficulties for learning in the simulated soccer environment in Section 2 and shall finish our hunt for an optimal intercept policy by returning to that issue: The state space S for the ball interception problem is crowded with discontinuities. These are regions $D \subset S$ where marginal changes to one of the six state variables of an $s \in D$ cause a very different action to be the optimal one, e.g. a turn instead of a dash or a turn to a very different angle. In Figure 1 we have considered that problem from an analytical perspective; Figure 3 illustrates it from a learning perspective for two different subsets of S.

In Figure 3a) comparatively simple intercept situations are regarded – the player's velocity is $\vec{v_p} = \binom{0}{0}$, the ball's $\vec{v_b} = \binom{0}{-1}$, i.e. the ball is departing from the player starting from different positions (the ball offset x is varied from -5 to 5 while the ball-player distance along the y-axis is constant). The chart gives an impression of the state value function V approximated by the neural net from Section 4.2. It shows the value of the successive state after having made a (initial) turn action a subject to the turn angle used when performing a. Light shades of gray indicate high expected rewards, dark ones mean low expected rewards. The thick-lined graph refers to the maximal successor state value for each $x \in [-5, 5]$ and thus denotes the turn angle chosen when exploiting V greedily. The thin-lined graph reflects the optimal turn angle computed by algorithm MB (thus

[3] An additional evaluation has shown, the relative difference between the learnt policy's and MB's performance remains quite the same when the Soccer Server's noise is present during learning.

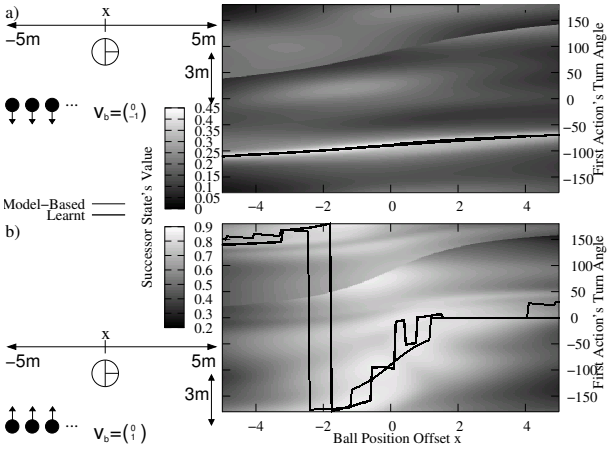

Figure 3: Variance in Determination of the Initial Turn Angle for Ball Interception: (a) for simple interceptions and (b) for more difficult ones

corresponding to $V^\star$). Obviously, in Figure 3a) both graphs are hard to differentiate, i.e. the learnt policy is optimal for this region of S. The situation is, unfortunately, different for the scenario sketched in Figure 3b), where the player faces more "difficult" situations with an approaching ball. Here, the graph for model-based, calculated initial turn angles is not as continuous as in Figure 3a), instead the turn angle for the agent's first action subject to x features numerous points of discontinuity. Capturing all those discontinuities has not been accomplished entirely by the neural-net based value function so that for several situations slightly sub-optimal turn angles will be preferred, which result in intercept trajectories that last one or more cycles longer than necessary.

5 Conclusion

In this paper we have dealt with a sub-task of robotic soccer simulation, the problem of intercepting moving balls as quickly as possible, which is one of the most important skills needed in robotic soccer. Using a Reinforcement Learning approach, we have aimed at learning a behavior policy for that task that is competitive, i.e. that nearly reaches the quality of a hand-coded reference method which is known to provide optimal results in a noise-free environment.

For high-dimensional, continuous state spaces, as the one considered in the scope of this work, the application of a function approximation mechanism is indispensable. Therefore, we have applied and comparatively evaluated several techniques for function approximation, from which we selected neural networks to be best suited. Moreover, we successfully devised several strategies to enhance the learning process and to improve neural network training.

The final intercept policy learnt clearly outperforms our earlier attempts to learn a ball intercept behavior. Further, it is almost as good as the model-based reference approach, which is known to provide optimal results in a noise-free environment. The remaining gap in quality can be put down to the difficulty in capturing a highly non-continuous state value function using function approximation. Summing up, we could show that the behavior policy gained by applying a Reinforcement Learning approach for the intercept task, can be almost as competitive as hand-coded and analytic solutions. Moreover, it became clear that a significant amount of effort must be invested when trying to advance over straightforward learn approaches and aiming at gaining near-optimal policies.

References

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

[2] Damien Ernst, Pierre Geurts, and Louis Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6:503–556, 2005.

[3] T. Gabel and M. Riedmiller. CBR for State Value Function Approximation in Reinforcement Learning. In *Proceedings of the Fifth International Conference on Case-Based Reasoning*, Chicago, USA, 2005. Springer.

[4] G. Gordon. *Approximate Solutions to Markov Decision Processes*. Ph.D. thesis, Carnegie Mellon University, 1999.

[5] M. Hasenjäger and H. Ritter. *Active Learning in Neural Networks*, pages 137–169. Physica-Verlag GmbH, 2002.

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

[7] G. Kuhlmann and P. Stone. Progress in Learning 3 vs. 2 Keepaway. In *RoboCup-2003: Robot Soccer World Cup VII*, Berlin, 2004. Springer Verlag.

[8] A. Merke and M. Riedmiller. Karlsruhe Brainstormers – A Reinforcement Learning Way to Robotic Soccer II. In A. Birk et al., editor, *RoboCup2001: Robot Soccer World Cup*. Springer, 2001.

[9] 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 ICML*, Slovenia, 1999. Morgan Kaufmann.

[10] I. Noda, H. Matsubara, K. Hiraki, and I. Frank. Soccer Server: A Tool for Research on Multi-Agent Systems. *Applied Artificial Intelligence*, 12(2-3):233–250, 1998.

[11] M. Riedmiller and H. Braun. A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm. In *Proceedings of the IEEE International Conference on Neural Networks (ICNN)*, 1993.

[12] F. Stolzenburg, O. Obst, and J. Murray. Qualitative Velocity and Ball Interception. In *Advances in AI, 25th German Conference on AI*, 2002.

[13] P. Stone, P. Riley, and M. Veloso. The CMUnited-99 Champion Simulator Team. In Veloso, Pagello, and Kitano, editors, *RoboCup-99: Robot Soccer World Cup III*, Berlin. Springer.

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

[15] M. Veloso, T. Balch, and P. Stone. RoboCup 2001: The Fifth Robotic Soccer World Championships. *AI Magazine*, 1(23):55–68, 2002.