Karlsruhe Brainstormers - A Reinforcement Learning approach to robotic soccer

M. Riedmiller, A. Merke, D. Meier,
A. Hoffmann, A. Sinner, O. Thate, and R. Ehrmann

Institut für Logkik, Komplexität und Deduktionssyteme
University of Karlsruhe, D-76128 Karlsruhe, FRG

Abstract. Our long-term goal is to build a robot soccer team where the decision making part is based completely on Reinforcement Learning (RL) methods. The paper describes the overall approach pursued by the Karlsruhe Brainstormers simulator league team. Main parts of basic decision making are meanwhile solved using RL techniques. On the tactical level, first empirical results are presented for 2 against 2 attack situations.

1 Introduction

The main motivation behind the Karlsruhe Brainstormer's effort in the robocup soccer domain of the simulator league is to develop and to apply Reinforcement Learning (RL) techniques in complex domains. Our long term goal is a learning system, 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.

2 Robotic Soccer as a Reinforcement Learning Problem

The problem that we face in a robotic soccer game can be formulated as finding an optimal policy π^* in a Markov Decision Problem $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, p(\cdot|\cdot), c(\cdot)\}$. A state s of the state space $\mathcal{S}$ consists of position and velocity of the 22 players and the ball, an action $a \in \mathcal{A}$ is a 22-dimensional vector of basic commands kick/ turn/ dash (one for each player), the world model $p(s_{t+1}|s_t, a_t)$ gives the transition probability that the successor state s_{t+1} is reached when action vector a_t is applied in state s_t. Finally, each transition causes costs that occur, when action a is applied in situation s. Since we are generally interested in controlling one team, we will assume that we can only choose the eleven entries in the action

vector that correspond to our team. For the rest of the paper, we consider the remaining eleven components of the opponent team to be chosen by an arbitrary and unknown, but fixed policy. The task in an MDP is to find an optimal policy $\pi^* : \mathcal{S} \rightarrow \mathcal{A}$, that minimizes the expected costs, accumulated during a trajectory: $J^*(s) = \min_\pi E\{\sum_{t=0} c(s_t, \pi(s_t))|s_0 = s\}$. A central team policy π would have to assign each state an eleven-dimensional command vector.

Reinforcement Learning in principle is designed to find optimal or approximately optimal policies in MDPs where the state space is too large to be tackled by conventional methods. The basic idea is to approximate the optimal value function by iteratively refining estimates for J^*: $J_{k+1}(s_t) \leftarrow \{c(s_t, a_t) + J_k(s_{t+1})\}$. However, the robotic soccer domain is such complex, that currently known RL algorithms would have no chance to find a solution in acceptable time. Therefore, in the following we try to give an analysis of the occuring problems and discuss both solved and open questions of the solution approach pursued by the (human) Karlsruhe Brainstormers. Two main aspects have to be solved: one is the reduction of the complexity, the other is the question of Reinforcement Learning in distributed systems.

2.1 Reduction of the complexity

In principle, there are three sources of complexity: the number of states, the number of actions and the number of decisions that have to be taken until a trajectory is (succesfully) finished. One idea behind RL is to handle complex state spaces by learning on experiences occuring during interaction with the world, thereby considering only the relevant parts of state space. A further reduction might be obtained by learning on features rather than on the state representation directly. This might turn the MDP into a POMDP (partially observable MDP) and is a current research topic. The number of actions can be reduced by arbitrarily constraining the number of commands that the agent can choose. On one extreme, we might allow only a single action, and decision making becomes trivial, but we obviously pay for it by loosing quality of the solution. The trick therefore is to provide as many actions as might be useful for a good policy, but not more. On the level of basic commands it is very difficult to decide, which actions are actually required in a certain situation. However, if we allow whole sequences of actions, the job becomes easier: A player needs a sequence to intercept the ball, to dribble the ball and so on. Such a sequence is called a 'move' here. Each move has a defined subgoal. The concept of moves has the following two important aspects: It considerably reduces the number of actions available to the agent, and being itself a sequence of basic commands, it reduces the number of decisions made by the agent. As we will see below, a move itself can be learned by RL methods.

2.2 Reinforcement Learning in distributed systems

Applying the move-concept, the policy of an agent has to select between a couple of moves instead of basic commands - this can be interpreted as a 'tactical'

decision. In the soccer domain, the tactical decisions have to be done by each agent individually. This raises the problem of learning in distributed scenarios. Since all the players have a common goal, this is a cooperative multi-agent system. In the MDP-framework, the cooperation aspect can be modelled by giving each agent the same transition costs $c()$. 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 [2]. 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 4.1).

3 Moves

A move is a sequence of basic actions, that transforms a current situation $s(0)$ into a new situation $s(t)$ some time steps later. The resulting situation is one of a set of terminal states $\mathcal{S}^f$, which might be either positive/ desired situations $\mathcal{S}^+$ or negative/ undesired situations $\mathcal{S}^-$. The move ends, if either a terminal state is reached $s(t) \in \mathcal{S}^f$, or the time exceeds a certain limit $t > t_{max}$.

For example, the move *intercept-ball* terminates if either the ball is within the player's kick range ($\mathcal{S}^+$) or if the agent encounters a situation, where it is no more possible for the player to reach the ball ($\mathcal{S}^-$).

3.1 Reinforcement Learning of Moves

Since each move has a clearly defined goal, the task is now to find a sequences of basic commands that does the job. This can be done either by conventional programming, or, as it is the case in our approach, by reinforcement learning methods. The general idea of RL is that the agent is only told, what the eventual goal of its acting is and which actions it might use (here: turn/ kick/ dash). Per time step, one command is selected depending on the situation. Learning here means to incrementally improve its decision policy such that the learning goal is fulfilled better and better. Here we apply Real-Time Dynamic Programming methods [1], that solve the learning problem by incrementally approximating an optimal value function by repeated control trials. Since the state space is continuos, a feedforward neural network is used to approximate the value function. For a detailed description of RL using neural networks, the reader is referred to [3].

Example: Learning to kick Finding a good kick routine is a very tedious job. In the 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 this 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 negative 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 $c(s, u) = 0.002$ [3]. After about 2 hours of learning, the resulting policies

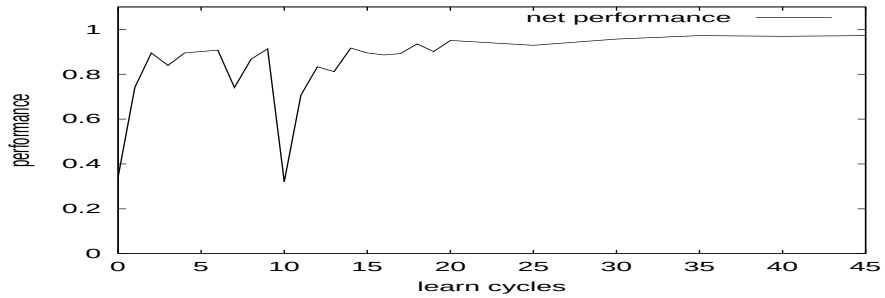

Fig. 1. An example of the performance during learning to kick the ball with maximal velocity 2.5 m/s. Each learning cycle comprises 2000 start states with different ball positions and velocities and the corresponding traversed trajectories. The net stabilizes after approximately 150000 individual updates and reaches a performance of approx. 95%.

were quite sophisticated - much more efficiently 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 (see also figure 1). It was able to learn to accelerate the ball to speeds up to 2.5 m/s. Alltogether, three different neural nets were trained, one for target velocities up to 1.0 m/s, one for target velocities up to 2.0 and one for target velocities up to 2.5 m/s (each of them using 4 inputs, 20 hidden and 1 output neuron).

4 The tactical level

On the tactical level, each agent must decide among the following moves *intercept-ball*, *go in one of 8 directions*, *wait at position*, *pass ball to teammate* (10 choices), *shoot to goal in three variants*, *dribble in one of 8 directions*. This makes an overall of 10 actions in situations, where the agent has no ball and 22 actions in situations where the agent possesses the ball. No agent can win the game on its own; therefore, the agents must learn to cooperate and coordinate their behaviour. A

team strategy can only work, if every agent acts reliably and contributes to the common goal.

4.1 Reinforcement Learning of team strategies

Our initial experiments are carried out with 2 attackers against 1 or 2 defenders. The defenders have a fixed policy: they run to the ball and if they get it, they are directly moved to the attackers goal, where they score. The task for the attackers is to score as fast as possible. If they do so, they are 'rewarded' by final costs of 0; if they loose the ball, they are punished by costs of 1. Every time step the attackers need until they score, they get a small constant cost of 0.002 (meaning that they can need up to 500 steps until their current policy looks as bad as a policy that looses the ball). At every time a new decision is required, the agent can choose between 13 moves if he has the ball (he has only one teammate for passing) and 10 moves if he is without the ball. In contrast to the approach proposed in [4], we used the complete continuous state information as input. Since this state information is continuous, we use neural networks to represent the value function. In the implementation finally used, each move uses an individual net to represent the costs that would occur if this move was actually applied in the respective situation. This allows to use individual features for every move, hence supporting generalization abilities. The following table shows the improvement achieved by a learned policy over an alternative greedy policy, which gets the ball and try to score then.

	greedy	RL trained
2 against 1	35 %	85 %
2 against 2	10 %	55 %

The greedy policy scores against 1 defender in 35% of the situations and against 2 defenders in only 10% of the situations. The 2 learning agents successfully learned to cooperate to score in most of the situations. In case of 1 defender, they score in 85% of the situations, and against 2 defenders they score in 55% of the situations. This is a fairly high rate of success, since in a real soccer game not to score does not mean to get a goal on your own, but just that your team has to regain the ball. We even observed such complex cooperation patterns as 'double passes'!

5 Summary

The soccer domain can be modelled as a complex MDP. Its main properties are the high dimensional, continuous state space, the huge number of basic commands, huge number of policies the requirement for cooperative independent Reinforcement Learning and the problem of the partial observability of the state information.

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. Feedforward neural networks are used for function approximation

in continuous state spaces. To deal with the Multi-agent aspects, we pursue both the investigation of theoretically founded distributed RL algorithms plus the empirical/ heurisically motivated way of modified single-agent Q-learning. 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. However, RL methods have already proven to be very useful in our competition agent.

5.1 Reinforcement Learning in the competition team

In the current competition version of our Brainstormer's agent all basic moves are learned by Reinforcement Learning, i.e. 1. a *kick* move which can accelerate the ball to put it with arbitrary velocity (0 to 2.5 m/s) in a desired direction 2. an *intercept-ball* move that effectively intercepts a rolling ball, taking the stochastic nature of the domain into account 3. an *dribble* move that allows to run without losing control over the ball, 4. a *positioning* move which reaches a particular position while avoiding collisions with other players, 5. a *stop-ball* move specialised in stopping high speed balls, 6. a *hold-ball* move which keeps the ball away from an attacker. Nearly all of the fundamental command decisions are therefore done by neural network based decision making.

On the tactics level, the very promising results for the 2 against 2 attack play are currently not directly applicable in the competition agent. For the tactic level, currently a method is used that we consider to be an intermediate step to Reinforcement Learning: Each possible move is judged by both its usefulness (quality) and probability of success. The quality of a move is given by a simple priority ranking (PPQ-approach).

5.2 Acknowledgements

We would like to thank the CMU-Team for providing parts of the source code of their competition team. In our current agent, we make use of their world model.

References

1. A. G. Barto, S. J. Bradtke, and S. P. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, (72):81–138, 1995.
2. 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.
3. 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.
4. 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.