

Karlsruhe Brainstormers - Design Principles

Karlsruhe-Brainstormers

*Martin Riedmiller, Sebastian Buck, Sergio Dilger,
Ralf Ehrmann, Artur Merke*

MR, SB, SD, RE, AM: University of Karlsruhe, ILKD

Abstract. *The following paper describes the design principles of the Karlsruhe Brainstormers team for the RoboCup Simulator League. The basic motivation behind our approach is to broadly apply Machine Learning techniques. In particular, our longterm goal is to apply Reinforcement Learning techniques to autonomously learn team playing capabilities. This longterm goal determined the structure of the decision module, which has to choose between several available 'high-level' moves based on evaluation functions. We plan to reach the final autonomously learning agent in several stages. The current version uses a hybrid decision module with both rule-based and learning components.*

1 Introduction

From a high-level point of view, the idea behind our approach is that we try to treat the task of playing soccer as a (distributed) optimization problem. The optimization objective is to control our eleven players such that in the given noisy environment, our team scores one more goal than the opponent. To do so, in each time-step we can choose among a couple of basic actions (*turn, dash, kick, nop*), which are parametrized by one or two real variables. What we are heading for is to find an appropriate (finite) sequence of these commands (one for each time-step), that when executed leads to the fulfillment of the optimization goal.

A simple calculation shows, that directly applying conventional optimization techniques fails due to the large amount of possible policies. Even if we discretize the parametrized basic commands rather coarsely, in each time step about 1000 different choices are possible. For example, if we discretize the turn angle in steps of 5° , then we have 72 different turn commands. For the two parameters of the kick command, if we discretize the power in steps of 10 and the angle in steps of 5° we have $14 \times 72 = 1008$ different combinations. Similarly we can assume about 14 *dash* commands, leading to more than 1000 different actions per time-step. Even in a very short time-period of, say 100 time-steps, this makes an overall of more than $(10^3)^{100} = 10^{300}$ different policies *per player*. This becomes even worse, if we consider that each of the 11 players chooses an action per time-step, which makes

$(1000^{11})^{100}$ different team policies (during the considered 100 time-steps period!).

Clearly, no one would earnestly formulate the soccer-problem in the above described fashion. Instead, other solution mechanisms are applied here, i.e. rule-based policies, heuristics, domain specific knowledge, planning or learning. Our academic interest is to consider the soccer game as a benchmark example where we may figure out more or less generic methods that can be generally applied for the solution of large optimization problems. What we are especially interested in is the application of Reinforcement Learning methods for optimization, that means the autonomous finding of nearly optimal policies through experience.

2 Basic Architecture

We treat our agent as a reactive closed-loop controller, that receives as an input the current state of the field¹ and outputs a basic command (*turn*, *dash*, ...) to the server. In principle, an agent sees a stochastic control task, where the stochasticity results from a) partial state information, b) from noise introduced by the server when the action is executed or when sensor information is given and c) from the various policies that are executed by the other players (both teammates and opponents).

Like in most soccer-agent architectures (e.g. [3]), the state information is provided by a world-model. This state information is then processed to the main controller, which makes the decision. As already discussed in the introduction, the number of decisions performed during the optimization period exponentially influences the number of possible policies. Thus as a *general* design principle it is very important to reduce the number of decisions. One way to do this, is to compound basic commands to useful *command sequences*. The term 'useful' already indicates, that here domain knowledge plays an important role. But there is a tradeoff: By allowing only a limited number of command sequences instead of the full combinatorical range of basic commands, we possibly sacrifice some of the optimality of the solution. It is therefore important, to still provide an appropriate range of decisions for the controller. In the soccer game, the *command sequences* are what we call '*moves*' (in analogy to board games; in other architectures these moves are sometimes called '(extended) skills'). These moves reduce the complexity of the optimization problem in two dimensions: first the number of choices at a time-step is reduced (from more than 1000 different basic commands to about 20 different moves) and second, the number of time-steps at which decisions have to be made is reduced, since every move implies a complete *sequence* of basic commands.

2.1 The Moves

The moves provided to the controller belong to two classes: moves with ball and moves without ball. Examples for moves with ball are *pass_to_player*, *shoot_to_goal*, *dribble_forward*, ..., examples for moves without ball are

¹currently position/ velocity information of ball and players; incorporating more knowledge like stamina information etc. is possible

go_to_ball, *go_to_position*, ... Each move typically generates a sequence of basic commands. Further, the moves are reactive, which means that the command sequence may vary with the currently observed situation (to allow for example to react to noise or unexpected events as for example a sudden obstacle). Some of the moves require additional control information. For example, *shoot_to_goal* would require a parameter indicating the target direction, i.e. left corner or right corner of the goal. One possibility is to give this decision to the move itself, meaning that when *shoot_to_goal* is called, it autonomously determines the best target region, depending on the situation. This corresponds to the concept of an 'intelligent' move. Another possibility is to directly call the move more specifically, e.g. *shoot_to_goal_Left*. Since our controller should have as much control as possible, we implemented the latter variant in our approach. The provided moves are designed such that the controller should have a reasonable choice in order to solve its task appropriately.

2.2 The controller

Up to now, nothing was said about how the controller chooses between the various moves. Since we aim at applying Reinforcement Learning techniques, one natural formulation is that the controller chooses the move that gets the best evaluation in the current situation. Let U be the set of all available moves and $U(s)$ be the set of moves that is available in the current situation (for example, if the ball is not kickable in s , then $U(s)$ is the set of moves without ball). Note that a proper determination of $U(s)$ can again be used to reduce the complexity of the original optimization problem. For example, additional if-then-rules might be used to restrict the set of available moves. Again, there is the danger of losing optimality by a too harsh restriction. The basic decision making process is then to evaluate all remaining moves by a value function $Q(s, u)$ and to select the best move, i.e.

$$u^*(s) = \arg \min_{u \in U(s)} Q(s, u)$$

The question that remains is: how do we get $Q(s, u)$ that evaluates the quality of applying action u in situation s ?

2.3 Determining $Q(s, u)$

In our current agent, we are experimenting with three different ways to get an appropriate state/ action-evaluation function $Q(s, u)$:

- definition of heuristic evaluation functions by a human
- supervised learning from relative evaluation information ('this move is better than this')
- reinforcement learning through experience

Whereas our longterm goal is to get to the third approach (reinforcement learning in a multi agent scenario), in the current stage approach one and

two are realized. This is not only to get some experience about the principle capabilities and problems of this 'move-selection' approach, but also since it might be useful to combine the first two approaches with the third one. Several things are possible here: we might use the heuristic knowledge to get a good starting point for the reinforcement learning approach, or we might even use this a priori knowledge in some situation as a fixed policy again in order to reduce the complexity of the decision problem.

In case of the second and the third approach, the Q -function is approximated by feedforward neural networks. In the second approach, a human expert provides examples of good play. The way the examples are presented is not in an absolute manner (in situation s do action u') but instead is given as relative information: in this situation action u' is better than u'' . This information can directly be used to train a Q -evaluation function.

In a reinforcement learning scenario, the Q -function is determined through experience gathered during playing. Several formulations of the basic optimization goal (shoot one more goal than the opponent) are possible. One can optimize the time to shoot a goal, or the probability of shooting a goal (in the remaining time) or minimize the use of stamina until the next goal is shot. The constraint is that the opponent should not shoot a goal himself in the meantime. We are currently experimenting with several different formulations of the optimization goal in a reasonably formulated subproblem of the overall task. In contrast to the approach proposed in [3] we will not discretize the state space but live in a continuous world. Therefore, we will use neural networks as function approximators for the Q -function and will gather our information on sample trajectories [1], [2].

2.4 Open Questions

The following are the open questions on which we are working:

- general multi agent scenario: theoretical foundation of multi-agent reinforcement learning algorithms, use of communication
- general reinforcement learning: tackling of big optimization problems: modularization, integration of a priori knowledge, using function approximators, efficient use of training information
- soccer related reinforcement learning: reinforcement learning of team play policies, learning basic skills (RL for the kick-command)
- soccer related other questions: clever positioning

3 Acknowledgement

Thanks to the supporting 'human brainstormers'-team, that tries to make the concepts work in practice: Sebastian Buck, Artur Merke, Ralf Ehrmann, Sergio Dilger, Andreas Hofmann, Alex Sinner, Ortwin Thate and Lutz Frommberger.

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. Riedmiller. *Autonomously learning neural controllers*. VDI-Verlag, 1996. Dissertation University of Karlsruhe (in german).
- [3] 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.