Karlsruhe Brainstormers - Design Principles

M. Riedmiller, S. Buck, A. Merke, R. Ehrmann, O. Thate, S. Dilger, A. Sinner,
A. Hofmann, and L. Frommberger

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

Abstract. The following paper describes the design principles of decision making in the Karlruhe Brainstormers team that participated in the RoboCup Simulator League in Stockholm 1999. It is based on two basic ingredigents: the *priority - probability - quality (PPQ)* concept is a hybrid rule-based/ learning approach for tactical decisons, whereas the definition of goal-orientented *moves* allows to apply neural network based reinforcement learning techniques on the lower level.

1 Introduction

The main interest behind the Karlsruhe Brainstormer's effort in the robocup soccer domain is to develop and to apply machine learning techniques in complex domains. Especially, we are interested in Reinforcement Learning methods, where the training signal is only given in terms of success or failure. So our final goal is a learning system, where we only plug in 'Win the match' - and our agents learn to generate the appropriate behaviour. Unfortunately, even from very optimistic complexity estimations it becomes obvious, that in the soccer domain, both conventional solution techniques and also advanced today's reinforcement learning techniques come to their limit - there are more than $(108 \times 50)^{23}$ different states and more than $(1000)^{300}$ different policies per agent per half time. The following describes the modular approach of the Brainstormer's team to tackle this complex decision problem.

2 The Decision Module

The task of the decision module is to compute in each time step a new basic command (i.e. *kick, turn, dash*) that is sent to the server. This command depends on the current situation s_t, which is provided by the world model module (not discussed here). As already discussed in the introduction, it is a very hard problem to do decisions at the level of basic commands.

An obvious approach - which is used in most of the known approaches in various variations (e.g. [3]) - is to introduce two levels of the decision making process. The lower level implements some useful basic skills of an individual player (for example, intercept a rolling ball). In our framework, such basic skills are called *moves* (in analogy to other strategic games as chess or backgammon). The *second level* is realized by the *tactics module.* Its task is to select one of the moves, depending on the situation. An appropriate choice of moves should finally lead to success in terms of scoring a goal. Also, aspects of team play are realized here.

2.1 The 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 outcomes ($\mathcal{S}^+$) 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 kickrange ($\mathcal{S}^+$) or if it encounters a situation, where it is no more possible for the player to reach the ball ($\mathcal{S}^-$).

Since each move has a clearly defined goal, it is now possible to find sequences of basic commands, that finally reach the defined goal. This can be done either by conventional programming, or, as it is the case in our approach, by reinforcement learning methods. In both cases, it is important that the goal of a move is reasonably chosen, that means that the solution policy is not too complex (e.g. a move 'win that game' would be desirable but its implementation will be as complex as the original problem).

Clearly, the quality and the number of different moves eventually determines the power of the individual player and therefore the whole team respectively.

2.2 Reinforcement Learning of Moves

The question now is how to implement a closed-loop policy that, after emitting a sequence of basic commands finally reaches the specified goal of the move? The above move definition directly allows to formulate the problem of 'programming' a move as a (sequential) Reinforcement Learning (RL) problem. The general idea of reinforcement learning is that the agent is only told, what the eventual goal of its acting is. The agent is only provided with a number of actions, that it can apply arbitrarily. In course of learning, it should incrementally learn a (closed-loop) policy, that reaches the final goal increasingly better in terms of a defined optimization criterion. Here we apply Real-Time Dynamic Programming methods [1], that solve the problem by incrementally approximating the optimal value function by repeated control trials. A feedforward neural network is used to approximate the value function [2].

In the current version which was used in Stockholm, the *kick*-move was learned by reinforcement learning. Several other teams have reported tricks how to implement a kick-routine by conventional programming using various heuristics. The problem with this approach is that it can be very time-consuming to find the right heuristics and to tune several parameters by hand. Instead, our reinforcement learning approach is much more convenient to handle - the work of looking for an appropriate policy is done by the agent/ computer itself. The agent is provided with a (finite) number of basic kick commands. The goal is defined in terms of a target ball direction and a target ball velocity. The agent receives costs for every kick command it uses until the ball has left its kickrange. If the ball is lost during the sequence, maximum costs occur; in case of success, the sequence is terminated with 0 costs [2]. This formulation results in a time-optimal policy: the number of kicks until successful termination is minimized. After about 2 hours of learning, the resulting policies were quite sophisticated - similar to the proposed heuristics, the agent learned to pull the ball back and to accelerate it several times in order to produce high speeds. It was able to learn to accelerate the ball to speeds up to 2.5 m/s. Of course, it is too difficult for a single policy (i.e. a single neural net) to manage to kick in all situations to all directions with all imaginable velocities. Instead, the problem was divided into 54 subproblems; therefore the neural kick-move now is based on 54 neural networks (each of them using 4 inputs, 20 hidden and 1 output neuron).

2.3 The Tactics Module and the PPQ approach

The task of the tactics module is to select one out of the set of available moves. The difficulty with this decision is, that in general, a complex sequence of moves has to be selected, until the final goal is achieved, because normally a single move will not lead to scoring a goal. In the soccer framework, this problem becomes even worse, since the success also depends on the behaviour of the whole team - a successful sequence can only be played, if all the agents involved make the correct decisions. Although we already started some promising experiments applying reinforcement learning to this decision level also, we are still some theoretical and practical steps away from a convincing practical solution (other teams also work on this topic [3]).

For our Stockholm competition team, we therefore worked on a different solution for the tactics module, which we call the *priority - probability - quality (PPQ)* approach. The idea origins in the observation, that some parts of the problem can be elegantly solved by simple rules, whereas other aspects are not so easily judged. The PPQ approach tries to combine the worlds of programmed and learned parts.

Priority Classes and Qualities The moves are partitioned into a number of classes, e.g. the class of *goal shots*, the class of *pass plays*, the class of *dribblings*. It is now relatively easy to define a reasonable priority ordering between these classes. For example, in our Stockholm approach, we used the following priorities: *1. shoot to goal, 2. pass forward, 3. dribble, 4. pass backward, 5. hold ball.*

If there are several choices of moves within a priority class, a *quality* function decides which move to chose. Conceptually, this quality function typically follows a very simple decision rule, for example pass to the player that is closer to the goal.

Learning of success probabilities

Each move has a certain probability of success, which depends on the current situation. The idea now is to learn this probability by a simple trial and error training procedure. The agent is set into various situations, executes a certain move and notes the success or failure of the move. The learning task now is to associate each situation with its outcome, for example in terms of a '1' for success and a '0' for failure. A feedforward neural network is used here to learn the training patterns. After training, the neural network, gets a certain situation as its input and outputs the *expected* value for success/ failure.

The decision algorithm

Each priority class has a set of available moves, $\mathcal{M}(i)$. The algorithm works through all the priority classes, until it finds one, where there is a move that has a higher probability of success than a certain threshold. This set is called $\mathcal{M}^+(i) := \{m | P_{NN}(m) \geq \theta_i, m \in \mathcal{M}(i)\}$. The threshold is selected such that a reasonable chance of success is given, for example $\theta_i = 0.8$. If there is more than one such move, one of them is selected by judging its quality. Note that this final judgment can be treated very relaxed, since it already is a nearly maximal useful move (determined by the priority of its class) and its also very likely a successful move (determined by its high success probability).

To guarantee termination of the algorithm, at least one class must exist, where $\mathcal{M}^+(i)$ is not empty. This is done by the definition of a default move that is always possible.

3 Conclusion

The Stockholm version is an intermediate step within our Brainstormers' concept of a learning agent. The final goal is to have an agent, which has learned its fundamental decision behaviour by reinforcement learning. However, until then a lot of work has to be done in the field of multi-agent RL, on Semi-Markov Decision Processes, partially observable domains (POMDPs) and on large-scale RL problems. Some of very recent RL ideas have already been successfully realized. For example, the moves-concept is closely related to Sutton's et.al 'options'-framework [4]. Therefore our work can be regarded as realizing and testing some conceptual ideas in a practical environment. The Brainstormer's Stockholm agent used an ensemble of 67 feedforward neural networks, 54 for the neural kicking (RL) routine, and 13 as probability networks in the tactics module.

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. Concepts and facilities of a neural reinforcement learning control architecture for technical process control. *Neural Computing and Application*, 1999.
3. P. Stone and M. Veloso. Team-partitioned, opaque-transition reinforcement learning. In M. Asada and H. Kitano, *RoboCup-98: Robot Soccer World Cup II.*
4. R. S. Sutton, D. Precup, and S. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 1999. to appear.