

# Reinforcement Learning In Soccer Simulation

Javad Sha'bani, Ali Honarvar, Mohammad Morovati,

Golnaz Abdollahian, Mona Mahmoodi

{javad, alihon, morovati, abdollahian, mahmoodi}@ee.sharif.edu

Electrical Engineering Department

Sharif University of Technology

**Abstract.** Being of a high complexity, most multi-agent systems are difficult to deal with by a hand-coded approach to decision making. In such complicated environments in which decision making processes should be controlled from both the individuals points' of view and the whole team, the common approach to the subject is the Reinforcement Learning (RL) method which is mainly based on learning the optimal policy through mapping this task to an episodic reinforcement learning framework. Reinforcement learning is the problem of generating optimal behavior in a sequential decision making environment given the opportunity of interacting with it. Since the Robocup domain is a multi-agent dynamic environment, with notable features making it outstanding for multi-agent simulation benchmarks, it has been largely used as a basis for international multi-agent simulation competitions and research challenges. For this purpose, reinforcement learning problems should be made “understandable” for “agents” which are Robocup players in this case. To make the agents “aware” of what they are intended to do, this paper will provide an overview of different methods and alternative approaches as a starting point for robocupers who are not familiar with reinforcement learning problems in the Robocup domain.

## 1. Introduction

During the last few decades, the most important prohibition against handling complicated situations by computers is the software limitations not hardware. In other words, developing the processes involving complex decision making activities such as driving a car, needs computer programs that computer engineers are not able to develop yet. The solution used for this problem is making the computers “learn” somehow. The methods used for solving this learning task may be categorized into two major parts.

First one is *supervised learning*. In this method the learning agent is supplied with a table of

pairs of states and actions which are labeled “true” or “optimal” and “false” or “non-optimal” and the learning agent attempts to generalize the results to obtain the characteristics of this pairs of state-action to enable it to choose the optimal action in those states that it has not encountered before. In this method if the generalization fails, the agent will choose non-optimal action. Usually generalization fails because extracting the features of the pairs of (state,action) is a very difficult task for the machine and needs a huge supply of “true” actions to be compared and generalized. The other method for making the machines learn is *reinforcement learning* which is unsupervised. In this method the agent interacts with its environment and it is given a reward for each action which is chosen in the state space. This means that the machine chooses an action in each state. By doing this it is given a value as a reward. Then agent can consider its situation and understand whether the action was optimal or not. If this exercise be held for largely enough episodes in each of which the machine keeps a record of optimal actions for each situation, the machine can learn to choose the optimal action by itself. In this paper we will describe reinforcement learning as a powerful approach for multi-agent systems and then feature out the RL works in soccer simulation. We put forth an understanding of reinforcement learning and map into Robocup simulation. The remainder of the paper is organized as follows. In section 2 we introduce reinforcement learning as a powerful technique in AI. Section 3 explains coding / generalization. In section 4 we describe mapping reinforcement learning to Robocup simulation subtasks. In section 5 concludes.

## 2. Reinforcement Learning

### 2.1. What is Reinforcement Learning (RL)

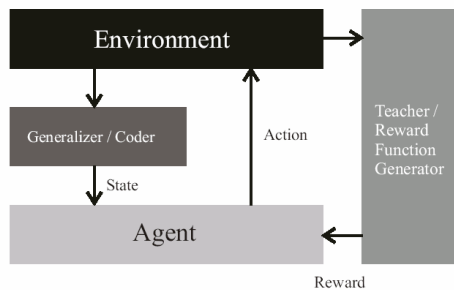
Reinforcement learning is one of the various methods in machine learning and decision theory. It is a kind of learning in which the learner which is called the “Agent” learns directly by interaction with the environment. The role of a

teacher here is very simple while the complexity has been included in the agent.

## 2.2. Description of the problem and the model

The model consists of three parts: **environment**, **agent**, **teacher** or the law of reward. The environment is a system with unknown or at least uncertain model; it depends on many factors, including the agent's action.

The teacher is the source of the learning data called "reward". The teacher can be very complex or very simple itself. But in RL it is often just a simple law from some parameters of the environment, anyway we call it the teacher like other learning dilemmas. In fact it is the only way for teaching the agent what we totally expect it to do.



The teacher can be evaluative or instructive: if the reward function depends on the action taken by agent, it is evaluative. Otherwise we consider this as instructive.

The agent interacts with environment and then receives rewards from the teacher. Its goal is to collect more reward. It receives some parameters from the environment and chooses an action. These are some features in contrast with other types of learning:

- 1) The agent has to learn anything it needs to interact with the environment. There is no one to directly teach it. This is the major difference with supervised learning, where the teacher provides the agent with samples and the whole problem will be a kind of function approximation.
- 2) The reward rule depends on environment and so depends on many parameters e.g. the actions the agent has performed until now, some random parameters, and the states until now.

## 2.3. Sources of Complexity

The problem, in its full sense, is very difficult, because it has an extremely large input space.

The obvious non-optimal solution for it is the blind full exploration in which the agent tests each case many times to obtain a stochastic model of the reward function. This approach is obviously non-practical in most problems. So some approximations and assumptions seem necessary to decrease the input space.

First we can assume that the environment model is a stationary stochastic one. But this is not enough to make the problem practically solvable. Also it doesn't apply to some problems as other additional assumptions are needed. A major common approach to this kind is Markov Decision Process (MDP). A more general model is semi-Markov Decision Process (SMDP).

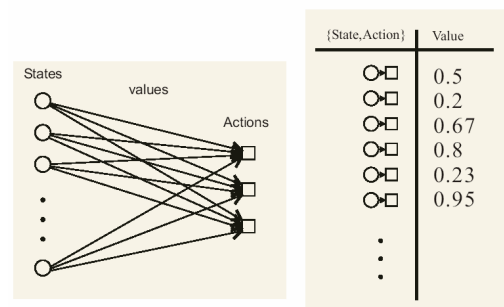
## 2.4. A common model for the agent

Assume a value or "sense" function  $Q$  from pairs  $(S_i, A_j)$  of {states, actions}.

$Q: \{\text{states, actions}\} \rightarrow \text{values (senses)}$

This function is roughly representing the probability of reward if at the state  $S_i$  the Action  $A_j$  is chosen. When the learning has been accomplished, on the run, the agent chooses the action with maximum  $Q$  (hope of reward) at each state. In addition if the environment is not stationary, the learning can continue online. So we can have agents who improve themselves even online which is an important advantage over supervised learning. The learning algorithm depends on obtaining a reasonable  $Q$ . There are various versions of learning algorithms, most differ in few features. These algorithms constitute of two major parts:

- How to modify  $Q$  due to rewards
- What action to choose

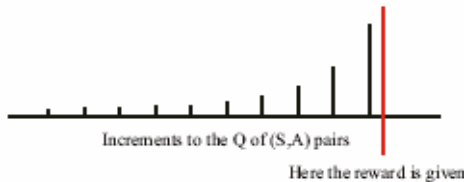


A good choice for the first part is Q-Learning algorithm, in which  $Q$  is initialized with some value for example all 0.5 s. Then after each reward the  $Q$  of (states, action) pairs that caused the reward are increased with some policy. The pairs before the reward are considered effective

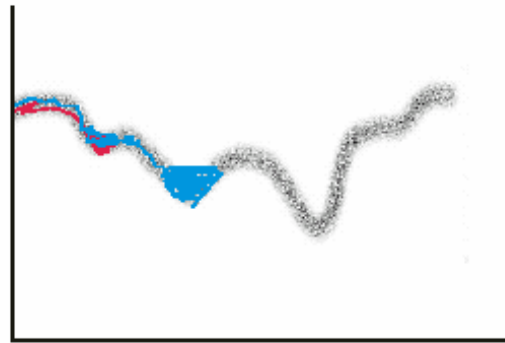
in the reward depending on how long after them the reward is given. For each pair (S,A) before the reward we have:

$$Q(S,A)=(1-\alpha)Q(S,A)+(\alpha)(\gamma^n)R$$

;where R is the reward,  $\alpha$  is a factor that makes the changes more stable,  $\gamma$  is the factor that shows the effect of previous pairs in the reward and n shows how long after this pair the reward happened.



Another useful policy is the common gradient method due to [5]. As a matter of the fact, gradient-descent methods are among widely used function approximation methods and are particularly well-suited to reinforcement learning.



this figure simply illustrates how  $\epsilon$  affects convergence time and quality of the learning process the red lines are due to small  $\epsilon$ . note that they converge to the first local minima. The blue lines are from larger  $\epsilon$  they also converge to a local minima but a better one .the black points are due to full exploration (from  $\epsilon = 1$ ) ,they in fact show the reward for various input.they can find the absolute maxima but they consume a lot of time.

Also, there's a whole body of theoretical work describing convergence problems using a variety of value based learning techniques with a variety of function approximation techniques [8]. Gradient techniques have been shown to be successful for simultaneous learning in matrix games [3, 10]. Next, we consider the policy that shows how to choose the action:

- Blind Search or full exploration:

In this algorithm the agent tests each pair of (state, action) many times to obtain a probability distribution model for the reward function. The major problem with this algorithm is that it's very time consuming.

- Greedy or full Exploitation:

Choose the Action  $A = \text{Max}\{ Q(S,A_j) \text{ over all } A_j \}$  ;where S is the current state.

Roughly speaking this method conceives a rough sketch of the reward function and acts as it says. We call it greedy because when it finds out a "rewardable action", it doesn't want to try anything else that doesn't know whether it is better or not. (It hasn't got any bravery!) The major problem with this method is that it may find just a local maximum of the function that may not be an optimum one.

- $\epsilon$ -Greedy Algorithm:

This is a modification of the previous algorithm. In this algorithm, at the decision time, the Action A is randomly one of Actions whose  $Q(S, A)$  is not further than a certain threshold  $\epsilon$  from  $\text{Max} Q(S_i, A_j)$ :

$$A \in \{ \text{Act} \mid ((\text{Max} Q(S_i, A_j)) - Q(S_i, \text{Act})) / \text{Max} Q(S_i, A_j) < \epsilon \}$$

(A is chosen randomly from this set).

This enables the agent to sometimes explore the actions, not just be greedy.

As you see there is a trade off between exploration and exploitation. The blind search uses a full exploration policy, the greedy algorithm uses a full exploitation policy and the  $\epsilon$ -greedy algorithm is a hybrid of both. If  $\epsilon = 1$  it is equal to the blind search and if  $\epsilon = 0$  it is equal to the greedy one. Choosing the threshold (or the trade off factor) is important in these algorithms. It affects directly the convergence time and "quality". Another useful approach is to simplify the input space: Limitate the input space. For this method a block, often called generalizer, is added to the system (illustrated in the first diagram). This block decreases the input from

environment into a much smaller space, so even full exploration policies can be applied to some problems. Robocop soccer simulation international competitions and research challenges has been regarded as a basis for researchers [21]. It is a fully distributed, multi-agent domain with both teammates and adversaries. There is hidden state, meaning that each agent has only a partial world view at any given moment. The agents also have noisy sensors and actuators. Meaning that they do not perceive the world exactly as it is, nor can they affect the world exactly as intended. Communication opportunities are limited, and the agents must make their decisions in real-time explained thoroughly in [1]. These italicized domain characteristics combine to make simulated robotic soccer a realistic and challenging domain. Robotic soccer is a sequential decision problem: agents attempt to achieve a goal by executing a sequence of actions over time. Every time an action can be executed, there are many possible actions from which to choose.[19] In principle, modern machine learning methods are reasonably well suited to meet the challenges of Robocop simulated soccer. As a common approach, reinforcement learning is all about sequential decision making, achieving delayed goals, and handling noise and stochasticity. However, Robocop soccer is a large and difficult instance of many of the issues which have been addressed in small, isolated cases in previous reinforcement learning research, the extent to which modern reinforcement learning methods can meet these challenges remains an open question. In simple learning tasks, estimates of value functions can be represented using a table. Although this approach is very clear, it is practical only to tasks with small numbers of states and actions. Most problems normally have very large state spaces; some have large action spaces. Using tables to store estimates of value functions in such cases makes inefficient use of experience and normally means impractical memory requirements. The problem of learning in large spaces is handled through generalization techniques, which allows compact representation of learned information and transfer of knowledge between similar states and actions. In large smooth state spaces, it is expected that similar states will have similar values and similar optimal actions. Therefore it should be possible to use experience gathered through interaction with a limited subset of the state space and produce a good approximation over a much larger subset. In most reinforcement learning (RL) architectures, the storage of a

variety of mappings is required, including State , Action (policy), State Reward (value functions), and (State  $\times$  Action  $\times$  State) . Some of these mappings can be learned using straightforward supervised learning. Function approximation techniques for supervised learning include neural network methods [17], fuzzy logic [15], and CMACs [11].training sets of input-output pairs are not available so other mappings can not directly be involved.

Numerous generalization techniques, both over input and over actions, are presented in [9]. In linear methods, the approximate function is a linear function of the parameter vector. Linear methods can be very efficient depends critically on the states represented in terms of features. Some of the most popular feature representation techniques include coarse coding [6], tile coding [14], and radial basis functions (RBF) [20].

### 3. Coding

Choosing features appropriate to the task is an important way of adding prior domain knowledge to reinforcement learning systems. Intuitively, the features should match to the features of the task, those for which generalization is most appropriate. In general, we also need features for combinations of these qualities. This is because the linear form prohibits the representation of interactions between features, such as the presence of feature [4, 24]. A good feature only may depend on the absence of another feature. In cases with such interactions one needs coding to introduce features when using a linear function approximator. We next consider two ways of tile coding [13, 22, 25] and Kanerva coding [7, 19].

#### 3.1. Tile coding

Tile coding [19], CMAC, is a popular technique for creating a set of boolean features from a set of continuous features. In reinforcement learning, tile coding has been used extensively to create linear approximators of state action values (e.g., [6]). Each tile has an associated boolean variable, so the continuous feature vector gets mapped into a very high dimensional boolean vector. In addition, nearby points will fall into the same tile for many of the offset grids, and so share many of the same boolean variables in their resulting vector. This provides the important feature of generalization.

Hashing is a common trick with tile coding to keep the number of parameters manageable. Each tile is hashed into a table of fixed size.

Collisions are simply ignored, meaning that two unrelated tiles may share the same parameter. Hashing reduces the memory requirements with little loss in performance. This is because only a small fraction of the continuous space is actually needed or visited while learning and so independent parameters for every tile are often not necessary. Hashing provides a means for using only the number of parameters which the problem requires while not knowing, in advance, which state-action pairs need parameters we lose a little in performance.

### 3.2. Kanerva coding

This method can be practical for high-dimensional tasks. As the key here is to keep the number of features from scaling explosively. *Kanerva coding* meets these criteria and chooses binary features that correspond to particular *prototype states*. The prototypes are randomly selected from the entire state space. The receptive field of such a feature is all states sufficiently close to the prototype. Kanerva coding uses a different kind of distance metrics than in tile coding and RBFs. The strength of Kanerva coding is that the complexity of the functions that can be learned depends entirely on the number of features, which bears no necessary relationship to the dimensionality of the task. To handle more complex tasks, a Kanerva coding approach simply needs more features.

## 4. Robocup Simulation subtasks and RL

Different subtask may have different results because of the complexity of the problem. so for comparing that which method can be more useful in soccer simulation we should apply different methods to a specific subtask .one of the important subtasks of soccer simulation is keepaway introduced by[4] in which one team merely seeks to keep control of the ball as long as possible. In one team, the keepers, are trying to maintain possession of the ball within a limited region, while in another team, the takers, are trying to gain possession. Reinforcement learning has been previously applied to robotic soccer. Using real robots, [23] used reinforcement learning methods to learn how to shoot a ball into a goal while avoiding an opponent. This task has a well-defined goal state. Also using goal-scoring as the goal state, TPOT-RL [18] was successfully been used to learn a full team of agents passing and shooting (using a

Monte Carlo learning approach). [11] Used observational reinforcement learning to update players' positions on the field based on where the ball has previously been located. [16] Uses reinforcement learning to learn low-level skills , such as kicking, ball-interception, and dribbling, as well as a cooperative behavior in which 2 attackers try to score against one or two takers. This work has used the full sensor space as the input representation, with a neural network used as a function approximator. Distributed reinforcement learning has been explored in discrete environments, such as the pursuit domain [21] and elevator control [12]. [4] has developed an approach of episodic Sarsa with linear tile-coding function approximation and variable  $\lambda$  to learning higher-level decisions in a keepaway subtask of Robocup soccer. In [2] Kanerva coding and reinforcement learning are combined to produce the KaBaGeRL decision making module. Kanerva coding is used as a generalisation method to produce a feature vector from the raw sensory input and the reinforcement learning uses this feature vector in order to learn an optimal policy. The efficiency of KaBaGeRL has been tested using the "3 versus 2 possession football" challenge, a sub problem of the Robocup domain.

## 5. Conclusions

In this paper, we explained reinforcement learning as a powerful means for applying to large state environments such as robocup simulation. Many researchers have been engaged in and the results suggest that reinforcement learning and robotic soccer may have much to offer each other. Our future work will involve further experimentation. We are going to find the most suitable method in soccer simulation subtasks such as keepaway to achieve more goals.

### Acknowledgements

We would like to thank Prof. Bijan Vosoughi Vahdat who has made contribution to the project. We also thank Resana Scientific Group for supporting our project.

### References

- [1] Peter Stone. Layered Learning in Multiagent Systems: A Winning Approach to Robotic Soccer. MIT Press, 2000.
- [2] Kostas Kostiadis and Huosheng Hu. KaBaGeRL: Kanervabased Generalisation

and Reinforcement Learning for Possession Football

[3] Singh, S.; Kearns, M.; and Mansour, Y. 2000. Nash convergence of gradient dynamics in general-sum games. In Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence, 541-548. Morgan Kaufman.

[4] Stone P., Sutton R. S., and Singh S., Reinforcement Learning for 3 vs. 2 Keepaway, RoboCup2000: Robot Soccer World Cup IV, Stone P., Balch T., and Kretzschmar G., editors., Springer Verlag, Berlin, 2001.

[5] Sutton, R. S.; McAllester, D.; Singh, S.; and Mansour, Y. 2000. Policy gradient methods for reinforcement learning with function approximation. In Advances in Neural Information Processing Systems 12. MIT Press.

[6] Stone, P., and Sutton, R. 2001. Scaling reinforcement learning toward Robocup soccer. In Proceedings of the Eighteenth International Conference on Machine Learning, 537-534.

[7] Kanerva P., Sparse Distributed Memory, The MIT Press, Cambridge, MA, 1988.

[8] Gordon, G. 2000. Reinforcement learning with function approximation converges to a region. In Advances in Neural Information Processing Systems 12. MIT Press.

[9] Kaelbling P. Leslie, Littman L. Michael, Moore W. Andrew, Reinforcement Learning: a survey, Journal of Artificial Intelligence 4, pp. 237-285, 1996.

[10] Bowling, M., and Veloso, M. 2002a. Multiagent learning using a variable learning rate. Artificial Intelligence.

[11] Albus J. S., Brains, Behaviour, and Robotics, BYTE Books, Subsidiary of McGrawHill, Peterborough, New Hampshire, 1981.

[12] R. H. Crites and A. G. Barto. Improving elevator performance using reinforcement learning. In D. S.

Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, Advances in Neural Processing Systems 8, Cambridge, MA, 1996. MIT Press.

[13] T. Dean, K. Basye, and J. Shewchuk. Reinforcement learning for planning and control. In S. Minton, editor, Machine Learning Methods for Planning and Scheduling. Morgan Kaufmann, 1992.

[14] C.-S. Lin and H. Kim. CMAC-based adaptive critic self-learning control. In IEEE Trans. Neural Networks, volume 2, pages 530-533, 1991.

[15] Lee C. C., A selflearning rulebased controller employing approximate reasoning and neural net

concepts, International Journal of Intelligent Systems, 6(1):71-93, 1991.

[16] Riedmiller M., Merke A., Meier D., Hoffman A., Sinner A., Thate O., and Ehrmann R., Karlsruhe brainstormers -- a reinforcement learning approach to robotic soccer, In Stone P., Balch T., and Kraetschmar G., editors, RoboCup 2000: Robot Soccer World Cup IV, Berlin: Springer Verlag, 2001

[17] Rumelhart D. E., and McClelland J. L., editors, Parallel Distributed Processing: Explorations in the microstructures of cognition, Vol. 1: Foundations, The MIT Press, Cambridge, MA, 1986.

[18] 13. P. Stone and M. Veloso. Team-partitioned, opaque-transition reinforcement learning. In M. Asada and H. Kitano, editors, RoboCup-98: Robot Soccer World Cup II. Springer Verlag, Berlin, 1999. Also in Proceedings of the Third International Conference on Autonomous Agents, 1999.

[19] 14. R. S. Sutton and A. G. Barto. Reinforcement Learning: An Introduction. MIT Press, Cambridge, Massachusetts, 1998.

[20] 15. R. S. Sutton and S. D. Whitehead. Online learning with random representations. In Proceedings of the Tenth International Conference on Machine Learning, pages 314-321, 1993.

[21] H. Kitano, M. Tambe, P. Stone, M. Veloso, S. Coradeschi, E. Osawa, H. Matsubara, I. Noda, and M. Asada. The RoboCup synthetic agent challenge 97. In Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, pages 24-29, San Francisco, CA, 1997. Morgan Kaufmann.

[22] 17. C. K. Tham. Modular On-Line Function Approximation for Scaling up Reinforcement Learning. PhD thesis, Cambridge Univ., Cambridge, England, 1994.

[23] 18. E. Uchibe. Cooperative Behavior Acquisition by Learning and Evolution in a Multi-Agent Environment for Mobile Robots. PhD thesis, Osaka University, January 1999

[24] Sutton R. S., Generalisation in reinforcement learning: Successful examples using sparse coarse coding, In Touretzky D. S., Mozer M. C., and Hasselmo M. E., Advances in Neural Information Processing Systems: Proceedings of the 1995 Conference, pp. 1038-1044, The MIT Press, Cambridge, MA, 1996

[25] 20. C. J. C. H. Watkins. Learning from Delayed Rewards. PhD thesis, King's College, Cambridge, UK, 1989.