Reinforcement Learning for 3 vs. 2 Keepaway

Peter Stone, Richard S. Sutton, and Satinder Singh

AT&T Labs — Research
180 Park Ave.
Florham Park, NJ 07932
`{pstone,sutton,baveja}@research.att.com`
`http://www.research.att.com/{~pstone,~sutton,~baveja}`

Abstract. As a sequential decision problem, robotic soccer can benefit from research in reinforcement learning. We introduce the 3 vs. 2 keepaway domain, a subproblem of robotic soccer implemented in the RoboCup soccer server. We then explore reinforcement learning methods for policy evaluation and action selection in this distributed, real-time, partially observable, noisy domain. We present empirical results demonstrating that a learned policy can dramatically outperform hand-coded policies.

1 Introduction

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.

Reinforcement learning (RL) is a useful framework for dealing with sequential decision problems [11]. RL has several strengths, including: it deals with stochasticity naturally; it deals with delayed reward; architectures for dealing with large state spaces and many actions have been explored; there is an established theoretical understanding (in limited cases); and it can deal with random delays between actions.

These are all characteristics that are necessary for learning in the robotic soccer domain. However, this domain also presents several challenges for RL, including: distribution of the team's actions among several teammates; real-time action; lots of hidden state; and noisy state.

In this paper, we apply RL to a subproblem of robotic soccer, namely 3 vs. 2 keepaway. The remainder of the paper is organized as follows. Sections 2 and 3 introduce the 3 vs. 2 keepaway problem and map it to the RL framework. Section 4 introduces tile coding or CMACs, the function approximator we use with RL in all of our experiments. Sections 5 and 6 describe the experimental scenarios we have explored. Section 7 describes related work and Section 8 concludes.

2 3 vs. 2 Keepaway

All of the experiments reported in this paper were conducted in the RoboCup soccer server [6], which has been used as the basis for successful international competitions and research challenges [4]. As presented in detail in [8], it is a fully *distributed*, *multiagent* 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. In addition, the perception and action cycles are *asynchronous*, prohibiting the traditional AI paradigm of using perceptual input to trigger actions. *Communication* opportunities are limited; and the agents must make their decisions in *real-time*. These italicized domain characteristics combine to make simulated robotic soccer a realistic and challenging domain.

The players in our experiments are built using CMUnited-99 agent skills [9] as a basis. In particular, their skills include the following:

HoldBall(): Remain stationary while keeping possession of the ball in a position that is as far away from the opponents as possible.

PassBall(f): Kick the ball directly towards forward f.

GoToBall(): Intercept a moving ball or move directly towards a stationary ball.

GetOpen(): Move to a position that is free from opponents and open for a pass from the ball's current position (using SPAR [15]).

We consider a "keepaway" task in which one team (forwards) is trying to maintain possession of the ball within a region, while another team (defenders) is trying to gain possession. Whenever the defenders gain possession or the ball leaves the region, the *episode* ends and the players are reset for another episode (with the forwards being given possession of the ball again).

For the purposes of this paper, the region is a 20m x 20m square, and there are always 3 forwards and 2 defenders (see Figure 1). Both defenders use the identical fixed strategy of always calling GoToBall(). That is, they keep trying to intercept the ball (as opposed to marking a player). Upon reaching the ball, they try to maintain possession using HoldBall().

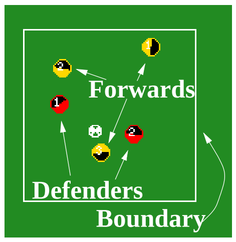

Fig. 1. A screen shot from a 3 vs. 2 keepaway episode.

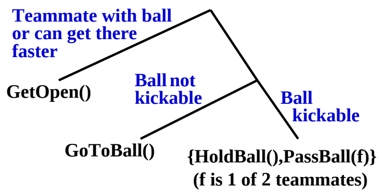

Fig. 2. The forwards' policy space.

An omniscient coach agent manages the play, ending episodes when a defender gains possession of the ball for a set period of time or when the ball goes outside of the region. At the beginning of each episode, it resets the location of the ball and the players semi-randomly within the region of play. The defenders both start in one corner and each forward is placed randomly in one of the other three corners – one forward per corner.

3 vs. 2 keepaway is a subproblem of robotic soccer. The principal simplifications are that there are fewer players involved; they are playing in a smaller area; and the players are always focussed on the same high-level goal: they don't need to balance offensive and defensive considerations. Nevertheless, the skills needed to play 3 vs. 2 keepaway well would clearly also be very useful in the full problem of robotic soccer.

3 Mapping Keepaway onto Reinforcement Learning

Our keepaway problem maps fairly directly onto the discrete-time, episodic RL framework. The RoboCup Soccer server operates in discrete time steps, $t = 0, 1, 2, \ldots$, each representing 100 ms of simulated time. When one episode ends (e.g., the ball is lost to the defenders), another begins, giving rise to a series of episodes. RL views each episode as a sequence of states, actions, and rewards:

$$s_0, a_0, r_1, s_1, \ldots, s_t, a_t, r_{t+1}, s_{t+1}, \ldots, r_T, s_T$$

where s_t denotes the state of the simulator at step t, a_t denotes the action taken based on some, presumably incomplete, perception of s_t, and $r_{t+1} \in \Re$ and s_{t+1} represent the resultant reward and the next state. If we wish to refer to an event within a particular episode, we add an additional index for the episode number. Thus the state at time t in the mth episode is denoted $s_{t,m}$, the corresponding action is $a_{t,m}$, and the length of the mth episode is T_m. The final state of any episode, s_{T_m} is the state where the defenders have possession or the ball has gone out of bounds. In this case we consider, all rewards are zero except for the last one, which is minus one, i.e., $r_{T_m} = -1, \forall m$. We use a standard temporal discounting technique which causes the RL agorithms to try to postpone this final negative reward as long as possible, or ideally avoid it all together.[1]

A key consideration is what one takes as the action space from which the RL algorithm chooses. In principle, one could take the full joint decision space of all players on a team as the space of possible actions. To simplify, we consider only top-level decisions by the forward in possession of the ball, that is, $a_t \in$ {HoldBall(), PassBall(f), GoToBall(), GetOpen()}.

That is, we considered only policies within the space shown in Figure 2. When a teammate has possession of the ball,[2] or a teammate could get to the

[1] We also achieved successful results in the undiscounted case in which $r_t = 1, \forall t$. Again, the RL algorithm aims to maximize total reward by extending the episode as long as possible.

[2] "Possession" in the soccer server is not well-defined since the ball never occupies the same location as a player. One of our agents considers that it has possession of the ball if the ball is close enough to kick it.

ball more quickly than the forward could get there itself, the forward executes GetOpen(). Otherwise, if the forward is not yet in possession of the ball, it executes GoToBall(); if in possession, it executes either HoldBall() or PassBall(f). In the last case, it also selects to which of its two teammates to bind f. Within this space, policies can vary only in the behavior of the forward in possession of the ball. For example, we used the following three policies as benchmarks:

Random Execute HoldBall() or PassBall(f) randomly
Hold Always execute HoldBall()
Hand-coded

```
    **If** no defender is within 10m: execute HoldBall()
    **Else If** receiver $f$ is in a better location than the forward with the ball and
    the other teammate; and the pass is likely to succeed (using the CMUnited-
    99 pass-evaluation function, which is trained off-line using the C4.5 decision
    tree training algorithm [7]): execute PassBall($f$)
    **Else** execute HoldBall()
```

4 Function Approximation

In large state spaces, RL relies on function approximation to generalize across the state space. To apply RL to the keepaway task, we used a sparse, coarse-coded kind of function approximation known as tile coding or CMAC [1, 16]. This approach uses multiple overlapping tilings of the state space to produce a feature representation for a final linear mapping where all the learning takes place (see Figure 3). Each tile corresponds to a binary feature that is either 1 (the state is within this tile) or zero (the state is not within the tile). The overall effect is much like a network with fixed radial basis functions, except that it is particularly efficient computationally (in other respects one would expect RBF networks and similar methods (see [12]) to work just as well).

It is important to note that the tilings need not be simple grids. For example, to avoid the "curse of dimensionality," a common trick is to ignore some dimensions in some tilings, i.e., to use hyperplanar slices instead of boxes. A second major trick is "hashing"—a consistent random collapsing of a large set of tiles into a much smaller set. Through hashing, memory requirements are often reduced by large factors with little loss of performance. This is possible because high resolution is needed in only a small fraction of the state space. Hashing frees us from the curse of dimensionality in the sense that memory requirements need not be exponential in the number of dimensions, but need merely match the real demands of the task. We used both of these tricks in applying tile coding to 3 vs. 2 keepaway.

5 Policy Evaluation

The first experimental scenario we consider is policy evaluation from an omniscient perspective. That is, the players were given fixed, pre-determined policies

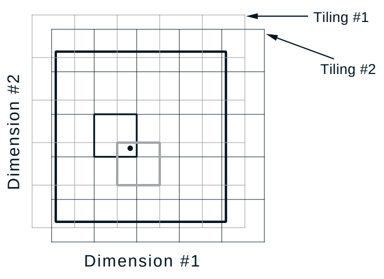

Fig. 3. Our binary feature representation is formed from multiple overlapping tilings of the state variables. Here we show two 5×5 regular tilings offset and overlaid over a continuous, two-dimensional state space. Any state, such as that shown by the dot, is in exactly one tile of each tiling. A state's tiles are used to represent it in the RL algorithm. The tilings need not be regular grids such as shown here. In particular, they are often hyperplanar slices, the number of which grows sub-exponentially with dimensionality of the space. Tile coding has been widely used in conjunction with reinforcement learning systems (e.g., [16]).

(not learned on-line) and an omniscient agent attempts to estimate the value of any given state from the forwards' perspective. Let us denote the overall policy of all the agents as π. The value of a state s given policy π is defined as

$$V^{\pi}(s) = E\left\{\sum_{t=0}^{\infty} \gamma^t r_{t+1} \mid s_0 = s, \pi\right\} = E\{-\gamma^{T-1} \mid s_0 = s, \pi\}$$

where $0 \leq \gamma < 1$ is a fixed *discount-rate* parameter (in our case $\gamma = 0.9$), and T is the number of time steps in the episode starting from s.[3] Thus, a state from which the forwards could keep control of the ball indefinitely would have the highest value, zero; all other states would have negative value, approaching -1 for states where the ball is lost immediately. In this section we consider an RL method for learning to approximate the value function V^{π}.

As is the case throughout this paper, both defenders always call GoToBall() until they reach the ball. The forwards use the hand-coded policy defined in Sections 2.

Every simulator cycle in which the ball is in kicking range of one of the forwards we call a *decision step*. Only on these steps does a forward actually have to make a decision about whether to HoldBall or PassBall(f). On each decision step the omniscient coach records a set of 13 state variables, computed based on F1, the forward with the ball; F2, the other forward that is closest to the ball; F3, the remaining forward; D1, the defender that is closest to the ball; D2, the other defender; and C, the center of the playing region (see Figure 4). Let dist(a,b) be the distance between a and b and ang(a,b,c) be the angle between a and c with vertex at b. For example, ang(F3,F1,D1) is shown in Figure 4. We used the following 13

[3] In the undiscounted case, the γ parameter can be eliminated and $V^{\pi}(s) = E\{T \mid s_0 = s\}$.

state variables: dist(F1,C), dist(F1,F2), dist(F1,F3), dist(F1,D1), dist(F1,D2), dist(F2,C), dist(F3,C), dist(D1,C), dist(D2,C), Minimum(dist(F2,D1),dist(F2,D2)), Minimum(dist(F3,D1),dist(F3,D2)), Minimum(ang(F2,F1,D1),ang(F2,F1,D2)), and Minimum(ang(F3,F1,D1),ang(F3,F1,D2)).

By running the simulator with the standard policy π through a series of episodes, we collected a sequence of 1.1 million decision steps. The first 1 million of these we used as a training set and the last 100,000 we reserved as a test set. As a post-processing step, we associated with each decision step t, in episode m, the empirical return, $-\gamma^{T_m-t-1}$. This is a sample of the value, $V^\pi(s_{t,m})$, which we are trying to learn. We used these samples as targets for a supervised learning, or *Monte Carlo*, approach to learning an approximation to V^π. We also applied temporal-difference methods, in particular, the TD(0) algorithm (see Sutton and Barto (1998) for background on both kinds of method).

It is not entirely clear how to assess the performance of our methods. One performance measure is the standard deviation of the learned value function from the empirical returns on the test set. For example, the best constant prediction (ignoring the state variables) on the test set produced a standard deviation of 0.292. Of course it is not possible to reduce the standard deviation to zero even if the value function is learned exactly. Because of the inherent noise in the domain, the standard deviation can be reduced only to some unknown base level. Our best result reduced it to 0.259, that is, by 11.4%. Our tests were limited, but we did not find a statistically significant difference between Monte Carlo and TD methods.

Another way to assess performance is to plot empirical returns vs estimated values from the test set, as in Figure 5. Due to the discrete nature of the simulation, the empirical returns can only take discrete values, corresponding to $-\gamma^k$ for various integers k. The fact that the graph skews to the right indicates a correlation between the estimated values and the sample returns in the test set.

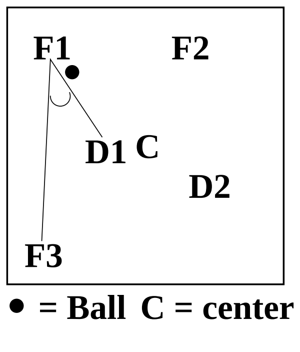

Fig. 4. The state variables input to the value function approximator.

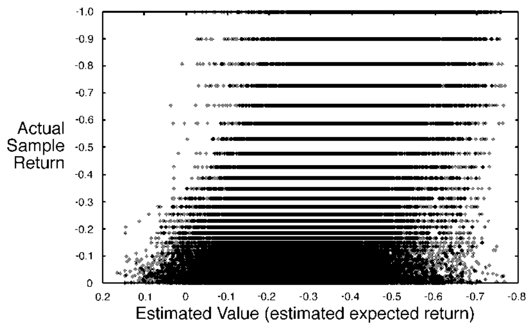

Fig. 5. Correlation of our best approximation of V^π to the sample returns in the training set.

The biggest influence on performance was the kind of function approximation used—the number and kind of tilings. We experimented with a few possibilities and obtained our best results using 14 tilings, 13 that sliced along a single state variable (one for each of the 13 state variables), and 1 that jointly considered dist(F1,D1) and dist(F1,D2), the distances from the forward with the ball to the two defenders.

6 Policy Learning

Next we considered reinforcement learning to adapt the forwards' policies to maximize their time of possession. To address this challenge, we used Q-learning [16], with each forward learning independently of the others. Thus, all forwards might learn to behave similarly, or they might learn different, complementary policies. Each forward maintained three function approximators, one for each of its possible actions when in possession of the ball (otherwise it was restricted to the policy space given by Figure 2). Each function approximator used the same set of 14 tilings that we found most effective in our policy evaluation experiments.

Below we outline the learning algorithm used by each forward. Here ActionValue(a, x) denotes the current output of the function approximator associated with action a given that the state has features x. The function UpdateRL(r) is defined below.

```
– Set counter = $-1$
– If episode over (ball out of bounds or defenders have possession)
    • If counter $> 0$, UpdateRL($-1$)
– Else, If the ball is not kickable for anyone,
    • If counter $\geq 0$, increment counter
    • If in possession of the ball, GoToBall()
    • Else, GetOpen()
– Else, If ball is within kickable range
    • If counter $> 0$, UpdateRL(0)
    • Set LastAction = Max(ActionValue($a$,current state variables)) where $a$
      ranges over possible actions, LastVariables = current state variables
    • Execute LastAction
    • Set counter = 0
– Else (the ball is kickable for another forward)
    • If counter $> 0$, UpdateRL(0) (with action possibilities and state variables
      from the other forward's perspective)
    • Set counter = $-1$
```

UpdateRL(r):

```
– TdError = $r * \gamma^{counter-1} + \gamma^{counter}$ Max(ActionValue($a$,current state vari-
  ables)) - ActionValue(LastAction,LastVariables)
– Update the function approximator for LastAction and LastFeatures, with
  TdError
```

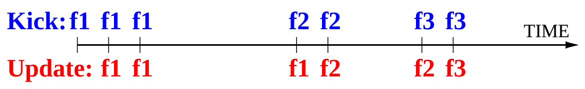

Fig. 6. A timeline indicating sample decision steps (kicks) and updates for the three forwards f1, f2, and f3.

Note that this algorithm maps onto the semi-markov decision process framework: value updates are only done when the ball is kickable for *someone* as indicated in Figure 6.

To encourage exploration of the space of policies, we used *optimistic initialization*: the untrained function approximators initially output 0, which is optimistic in that all true values are negative. This tends to cause all actions to be tried, because any that are untried tend to look better (because they were optimistically initialized) than those that have been tried many times and are accurately valued. In some of our runs we further encouraged exploration by forcing a random action selection on 1% of the decision steps.

With $\gamma = 0.9$, as in Section 5, the forwards learned to always hold the ball (presumably because the benefits of passing were too far in the future). The results we describe below used $\gamma = 0.99$. To simplify the problem, we gave the agents a full 360 degrees of noise-free vision for this experiment (object movement and actions are still both noisy). We have not yet determined whether this simplification is necessary or helpful.

Our main results to date are summarized in the learning curves shown in Figure 7. The y-axis is the average time that the forwards are able to keep the ball from the defenders (average episode length); the x-axis is training time (simulated time $\approx$ real time). Results are shown for five individual runs over a couple of days of training, varying the learning-rate and exploration parameters. The constant performance levels of the benchmark policies (separately estimated) are also shown as horizontal lines. All learning runs quickly found a much better policy than any of the benchmark policies, including the hand-coded policy. A better policy was even found in the first data point, representing the first 500 episodes of learning. Note that the variation in speed of learning with the value of the learning-rate parameter followed the classical inverted-U pattern, with fastest learning at an intermediate value. With the learning-rate parameter set to 1/8, a very effective policy was found within about 15 hours of training. Forcing 1% random action to excourage further exploration did not improve performance in our runs. Many more runs would be needed to conclude that this was a reliable effect. However, Figure 8, showing multiple runs under identical conditions, indicates that the successful result is repeatable.

7 Related Work

Distributed reinforcement learning has been explored previously in discrete environments, such as the pursuit domain [13] and elevator control [3]. This task

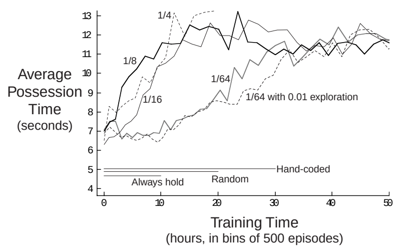

Fig. 7. Learning curves for five individual runs varying the learning-rate parameter as marked. All runs used optimistic initialization to encourage exploration. One run used 1% random actions to further encourage exploration.

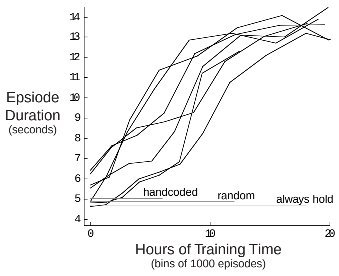

Fig. 8. Multiple successful runs under identical characteristics. In this case, undiscounted reward and 1% random exploration were used. Q-values were initialized to 0.

differs in that the domain is continuous, that it is real-time, and that there is noise both in agent actions and in state-transitions.

Reinforcement learning has also been previously applied to the robotic soccer domain. Using real robots, Uchibe used RL to learn to shoot a ball into a goal while avoiding an opponent [14]. This task differs from keepaway in that there is a well-defined goal state. Also using goal-scoring as the goal state, TPOT-RL was successfully used to allow a full team of agents to learn collaborative passing and shooting policies using a Monte Carlo learning approach, as opposed to the TD learning explored in this paper [10]. Andou's "observational reinforcement learning" was used for learning to update players' positions on the field based on where the ball has previously been located [2].

In conjunction with the research reported in this paper, we have been exploring techniques for keepaway in a full 11 vs. 11 scenario played on a full-size field [5].

8 Conclusion

In this paper, we introduced the 3 vs. 2 keepaway task as a challenging subproblem of the entire robotic soccer task. We demonstrated (in Section 5) that RL prediction methods are able to learn a value function for the 3 vs. 2 keepaway task. We then applied RL control methods to learn a significantly better policy than could easily be hand-crafted. Although still preliminary, our results suggest that reinforcement learning and robotic soccer may have much to offer each other.

Our future work will involve much further experimentation. We would like to eliminate omniscience and centralized elements of all kinds, while expanding the task to include more players in a larger area. Should the players prove capable of learning to keep the ball away from an opponent on a full field with 11 players on each team, the next big challenge will be to allow play to continue when the defenders get possession, so that the defenders become forwards and vice versa.

Then all players will be forced to balance both offensive and defensive goals as is the case in the full robotic soccer task.

References

1. J. S. Albus. *Brains, Behavior, and Robotics.* Byte Books, Peterborough, NH, 1981.
2. T. Andou. Refinement of soccer agents' positions using reinforcement learning. In H. Kitano, editor, *RoboCup-97: Robot Soccer World Cup I*, pages 373–388. Springer Verlag, Berlin, 1998.
3. 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.
4. 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.
5. D. McAllester and P. Stone. Keeping the ball from cmunited-99. In P. Stone, T. Balch, and G. Kraetszchmar, editors, *RoboCup-2000: Robot Soccer World Cup IV*, Berlin, 2001. Springer Verlag. To appear.
6. I. Noda, H. Matsubara, K. Hiraki, and I. Frank. Soccer server: A tool for research on multiagent systems. *Applied Artificial Intelligence*, 12:233–250, 1998.
7. J. R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, San Mateo, CA, 1993.
8. P. Stone. *Layered Learning in Multiagent Systems: A Winning Approach to Robotic Soccer.* MIT Press, 2000.
9. P. Stone, P. Riley, and M. Veloso. The CMUnited-99 champion simulator team. In M. Veloso, E. Pagello, and H. Kitano, editors, *RoboCup-99: Robot Soccer World Cup III*, pages 35–48. Springer Verlag, Berlin, 2000.
10. 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.
11. R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge,Massachusetts, 1998.
12. 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.
13. M. Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 330–337, 1993.
14. E. Uchibe. *Cooperative Behavior Acquisition by Learning and Evolution in a Multi-Agent Environment for Mobile Robots.* PhD thesis, Osaka University, January 1999.
15. M. Veloso, P. Stone, and M. Bowling. Anticipation as a key for collaboration in a team of agents: A case study in robotic soccer. In *Proceedings of SPIE Sensor Fusion and Decentralized Control in Robotic Systems II*, volume 3839, Boston, September 1999.
16. C. J. C. H. Watkins. *Learning from Delayed Rewards.* PhD thesis, King's College, Cambridge, UK, 1989.