Proceedings of the AAAI-2004 Workshop on Supervisory Control of Learning and Adaptive Systems,
pp. 30-35, San Jose, CA, July 2004.

Guiding a Reinforcement Learner with Natural Language Advice: Initial Results in RoboCup Soccer

Gregory Kuhlmann, Peter Stone, Raymond Mooney
Department of Computer Sciences
The University of Texas at Austin
{kuhlmann,pstone,mooney}@cs.utexas.edu

Jude Shavlik
Department of Computer Sciences
The University of Wisconsin - Madison
shavlik@cs.wisc.edu

Abstract

We describe our current efforts towards creating a reinforcement learner that learns *both* from reinforcements provided by its environment *and* from human-generated advice. Our research involves two complementary components: (a) mapping advice expressed in English to a formal advice language and (b) using advice expressed in a formal notation in a reinforcement learner. We use a subtask of the challenging RoboCup simulated soccer task (Noda *et al.* 1998) as our testbed.

Introduction

Reinforcement learning (RL) is a common way to create adaptive systems that learn to act in complex, dynamic environments (Sutton & Barto 1998). In RL, the learner repeatedly senses the world, chooses an action to perform, and occasionally receives feedback from its environment, which the learner then uses to improve its performance. Employing RL can be a more effective way to create intelligent robots and software agents than writing programs by hand. In addition, RL also requires much less human intervention than supervised machine learning, which requires large sets of labeled training examples. We describe our current efforts towards creating a reinforcement learner that learns *both* from reinforcements provided by its environment *and* from human-written suggestions; in fact, it is our goal that these suggestions be provided in ordinary English.

Typically, the feedback given to a reinforcement learner is simply a numeric representation of rewards or punishments. However, there usually is much more feedback that a human teacher of such a learner could provide. Several researchers have designed successful methods where this feedback can include high-level "advice" expressed by humans at a natural level of abstraction using statements in a formal language (Noelle & Cottrell 1994; Maclin & Shavlik 1994; Siegelmann 1994; Eliassi-Rad & Shavlik 2003). Since the agent employs machine learning, such advice need not be perfectly accurate, fully precise, nor completely specified. In some approaches, the RL-agent's human partner can provide advice at any time, based on the agent's current behavior. The advice may suggest an action to be taken immediately (Clouse & Utgoff 1992), or it may contain explicit conditions under which an action should be taken henceforth. It is the latter case that we consider in this paper. Good advice can rapidly improve an agent's performance while an agent can recover reasonably quickly from poor advice (Maclin 1995).

Natural language (NL) is the most convenient way for ordinary users to specify such advice. Methods that *learn* to map natural to formal language given training examples of the desired transformation can help automate the construction of an NL-advice interpreter. Learning methods also allow a system to automatically adapt to the particular language of specific users. Mooney's research group has been developing successful methods for learning to translate NL into formal semantic representations (Zelle & Mooney 1996; Thompson & Mooney 2003; Tang & Mooney 2001). By mapping instructions expressed in ordinary English to a formal representation, we aim to alleviate the sizable burden of explicitly supplying formal representations of advice.

In the remainder of this article, we describe our recent research that involves two complementary components: (a) mapping advice expressed in English to a formal advice language and (b) using advice expressed in a formal notation in a reinforcement learner. We use a subtask of the challenging RoboCup simulated soccer task (Noda *et al.* 1998) as our testbed. Before describing our technical approaches, we present the subtask of RoboCup soccer that we are using.

The Keepaway Task in RoboCup Soccer

Keepaway is a subproblem of RoboCup simulated soccer introduced by Stone and Sutton (Stone & Sutton 2001) in which one team, the *keepers*, tries to maintain possession of the ball within a limited region, while the opposing team, the *takers*, attempts to gain possession. Whenever the takers take possession or the ball leaves the region, the *episode* ends and the players are reset for another episode (with the keepers being given possession of the ball again). Parameters of the task include the size of the region, the number of keepers, and the number of takers. Figure 1 shows a screen shot of an episode with 3 keepers and 2 takers (called 3 vs. 2, or 3v2 for short) playing in a $20m \times 20m$ region.[1]

Agents in the RoboCup simulator (Noda *et al.* 1998)

Copyright © 2004, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

[1]Flash files illustrating the task are available at `http://www.cs.utexas.edu/~AustinVilla/sim/keepaway/`

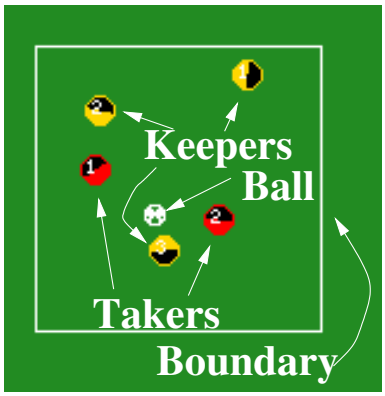

Figure 1: A screen shot from the middle of a 3 vs. 2 keepaway episode in a 20m x 20m region. Flash files illustrating the task are available from `http://www.cs.utexas.edu/~AustinVilla/sim/keepaway/`

receive visual perceptions every 150 *msec* indicating the relative distance and angle to visible objects in the world, such as the ball and other agents. They may execute a primitive, parameterized action such as `turn(`$angle$`)`, `dash(`$power$`)`, or `kick(`$power$`,`$angle$`)` every 100 *msec*. Thus the agents must sense and act asynchronously. Random noise is injected into all sensations and actions. Individual agents must be controlled by separate processes, with no inter-agent communication permitted other than via the simulator itself, which enforces communication bandwidth and range constraints. Full details of the RoboCup simulator are presented in the server manual (Chen *et al.* 2003).

In this work, we focus exclusively on training the keepers. As a way of incorporating domain knowledge, our learners choose not from the simulator's set of primitive actions but from higher-level actions constructed from a set of basic skills that were implemented by the CMUnited-99 team (Stone, Riley, & Veloso 2000).

Keepers have the freedom to decide which action to take only when in possession of the ball. A keeper in possession may either hold the ball or pass to one of its teammates. Keepers not in possession of the ball are required to select the **Receive** option in which the player who can get there the soonest goes to the ball and the remaining players try to get open for a pass.

We further incorporate domain knowledge by providing the keepers with rotationally-invariant state features computed from the world state. The keepers' set of state variables are computed based on the positions of: the keepers K_1–K_n and takers T_1–T_m, ordered by increasing distance from K_1; and C, the center of the playing region. 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 3 keepers and 2 takers, we used the following 13 state variables:

$dist(K_1, C)$, $dist(K_2, C)$, $dist(K_3, C)$,
$dist(T_1, C)$, $dist(T_2, C)$,
$dist(K_1, K_2)$, $dist(K_1, K_3)$,
$dist(K_1, T_1)$, $dist(K_1, T_2)$,
Min$(dist(K_2, T_1), dist(K_2, T_2))$,
Min$(dist(K_3, T_1), dist(K_3, T_2))$,
Min$(ang(K_2, K_1, T_1), ang(K_2, K_1, T_2))$,
Min$(ang(K_3, K_1, T_1), ang(K_3, K_1, T_2))$

For our purposes, the behavior of the takers is "hard-wired" and relatively simple. The two takers that can get there the soonest go to the ball, while the remaining takers try to block open passing lanes.

An obvious performance measure for this task is average episode duration. The keepers attempt to maximize it while the the takers try to minimize it. To this end, the keepers are given a constant positive reward for each time step an episode persists. For full details on the task and the learning scenario, see Stone and Sutton (2001).

Natural Language Interface

Allowing human teachers to provide advice in natural language allows them to instruct a learning agent without having to master a complex formal advice language. In our approach, a parser automatically translates natural-language instructions into an underlying formal language appropriate for the domain. Statements in this formal language are then used to influence the action policy learned by the agent.

In the RoboCup Coach Competition, teams compete to provide effective instructions to a *coachable team* in the simulated soccer domain. Coaching information is provided in a formal language called CLANG (Coach Language) (Chen *et al.* 2003). By constructing English translations for 500 CLANG statements produced by several teams for the 2003 RoboCup Coach Competition, we produced a corpus for training and testing a natural-language interface. Below are some sample annotated statements from this corpus:

- If player 4 has the ball, it should pass the ball to player 2 or 10.
  ```
  ((bowner our {4}) (do our {4} (pass {2 10}))))
  ```
- No one pass to the goalie.
  ```
  ((bowner our {0}) (dont our {0} (pass {1}))))
  ```
- If players 9, 10 or 11 have the ball, they should shoot and should not pass to players 2-8.
  ```
  ((bowner our {9 10 11})
   (do our {9 10 11} (shoot))
   (dont our {9 10 11} (pass {2 3 4 5 6 7 8})))
  ```

For a sufficiently restricted task, such as RoboCup coaching, parsing natural-language sentences into formal representations is a reasonably manageable task using current NLP technology (Jurafsky & Martin 2000). However, developing such a parser is a very labor-intensive software-engineering project. Consequently, methods that *learn* to map natural language to a given formal language from input/output pairs such as those above can significantly automate this difficult development process.

We have previously developed methods for learning to translate natural-language sentences into formal semantic representations. In particular, we have developed two integrated systems, CHILL (Zelle & Mooney 1996), which learns a parser for mapping natural-language sentences directly to logical form, and WOLFIE (Thompson & Mooney 2003), which learns a lexicon of word meanings required by this parser.

We are currently adapting CHILL and WOLFIE to learn to map English to CLANG as well as exploring several new approaches to semantic parsing. Our first new approach uses pattern-based transformation rules to map phrases in natural language directly to CLANG. Our second new approach first uses a statistical parser (Charniak 2000) to produce a syntactic parse tree, then uses pattern-based transformation rules to map subtrees of this parse to CLANG expressions. Our third new approach uses a parser that integrates syntactic and semantic analysis by pairing each production in the syntactic grammar with a compositional semantic function that produces a semantic form for a phrase given semantic forms for its subphrases (Norvig 1992). For lack of space, we elaborate only on the first of these new approaches, which is the simplest method.

CLANG comes with a formal grammar that defines the language using production rules such as:

ACTION → (pass UNUM_SET)
CONDITION → (bowner our UNUM_SET)

where UNUM_SET is a non-terminal symbol for a set of uniform numbers. Our translator uses transformation rules that map natural-language phrase patterns to subtrees of the parse tree for the CLANG representation. These transformation rules are repeatedly applied to construct the parse tree for the CLANG representation bottom-up. The non-terminal symbols in the CLANG grammar provide convenient intermediate representations that can be used to write general, effective transformation rules. For example, consider the sentence:

"If player 2 has the ball, player 2 should pass to player 10."

First, the transformation rule:
"player N has the ball" ⇒
CONDITION → `(bowner our {N})`
rewrites the sentence to
"If CONDITION[`(bowner our {2})`], player 2 should pass to player 10."

Next:
"pass to player N" ⇒ ACTION → `(pass {N})`
rewrites the sentence to
"If CONDITION[`(bowner our {2})`], player 2 should ACTION[`(pass {10})`]."

Next:
"player N should ACTION" ⇒
DIRECTIVE → (do our {N} ACTION)
rewrites the result to
"If CONDITION[`(bowner our {2})`],
DIRECTIVE[`(do our 2 (pass {10}))`]."

Finally:
"If CONDITION, DIRECTIVE." ⇒
RULE → (CONDITION DIRECTIVE)
produces the final CLANG result
"RULE[`((bowner our {2})`
`(do our 2 (pass {10})))`]".

We have developed a learning system called SILT, that automatically induces such transformation rules from manually annotated sentences (Kate *et al.* 2004). Current 10-fold cross-validation experiments on our CLANG corpus, in which 90% of the sentences are used for training and 10% for independent testing, demonstrate that the learned parser produces completely correct translations for about a third of novel sentences, and mostly correct translations for most other sentences.

Giving Advice to the Keeper

Previously, Stone and Sutton (2001) achieved successful learning results in the keepaway domain using a reinforcement learning algorithm called episodic SMDP Sarsa(λ) (Sutton & Barto 1998). They represented the state of the work via linear tile-coding function approximation (CMACs (Albus 1981)). We are using the same reinforcement learning methods in this work.

The function approximator takes in the keeper's set of state variables as input and produces values for each of the available actions: hold or pass to teammate k. The free variables in the function approximator (feature weights) are adjusted so that the value associated with a particular action approximates the expected episode duration should the learner select that action. Normally, these values are then used to determine the keeper's action, with the keeper typically selecting the action with the highest value (occasionally reinforcement learners also need to take *exploratory* actions, i.e., actions that do not have the highest value).

In order to incorporate advice, we add a new component to the learner called the *advice unit*. Like the function approximator, the advice unit generates values for the possible actions in the current world state. However, the output of the advice unit is not learned. Instead, it is determined by advice supplied by the user.

As stated previously, in our system, advice rules are represented using the standardized coach language, CLANG. CLANG rules consist of conditions specifying when the rule is triggered and directives specifying which action a player should or should not take. The conditions are composed of high-level predicates over the world state, such as: "is the ball within $10m$ of player 7?". CLANG includes the ability to refer to basic skills, such as: pass, dribble, shoot, etc.

The input to the advice unit is a set of features about the world state sufficient to evaluate CLANG conditions. Initially, all action values are set to zero. In each time step in which an action is to be chosen, the conditions of the rules supplied to the advice unit are matched against the current world state. If a rule "fires" (i.e., the LHS is satisfied) the corresponding action value is increased or decreased by a constant amount (± 2.0) depending on whether the directive advises for or against the action. The values generated by the advice unit are added to those generated by the function approximator and presented to the learning algorithm.

One of the nice things about this advice incorporation method is that it does not require the function approximator and the advice unit to use the same set of features to describe the world. At the same time, it allows for the advice to be adjusted by the learner. By adjusting the function approximator's value for an advised action, the agent can effectively "unlearn" the given advice.

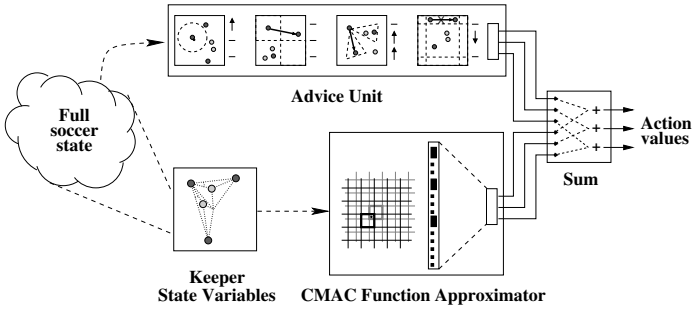

Figure 2: A pictorial summary of the complete advice integration scheme.

Experiments with Advice Giving

We conducted a series of preliminary experiments to measure the impact of advice on the quality of the learned policies and the speed at which those policies converge. In each experiment, we add a single piece of advice at the beginning of the learning trial. The advice applies to every member of the keeper team and remains active for the duration of the experiment.

Four Sample Pieces of Advice

We created four pieces of sample advice to test the system. The advice has not been extensively engineered to maximize performance. It is simply a collection of "first thoughts" made from observing the players during training and recognizing some of their shortcomings.

The first piece of advice, **Hold Advice**, states that the player in possession of the ball should choose to hold onto the ball rather than pass if no opponents are within $8m$. We observed that early in learning the keepers tend to pass more often than they probably should. Therefore, it seems reasonable to advise them not to pass when the takers are too far away to be a threat. Figure 3 illustrates this advice. The keepers are represented as darkly-filled circles and the takers are represented as lightly-filled circles.

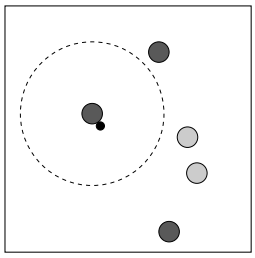

Figure 3: **Hold Advice** - If no opponents are within $8m$ then hold.

To give a sense of how advice is represented in the syntax of CLANG, the following is the CLANG rule corresponding to the **Hold Advice**:

```
(definerule hold-advice direc
  ((ppos opp {0} 0 0
        (arc (pt self) 0 8 0 360))
   (do our {0} (hold))))
```

The rule describes a circular region (shown dashed in Figure 3) centered at one of the keepers with a radius of $8m$, and states that if exactly 0 opponents are in that region, the keeper is advised to perform the hold action.

The next piece of example advice we call **Quadrant Advice**. As seen in Figure 4, the play region is divided into four quadrants. A player is advised to pass to a teammate if that teammate is in a different quadrant and that quadrant contains no opponents. This advice aims to encourage players to pass to teammates that are not being defended.

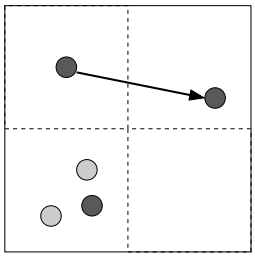

Figure 4: **Quadrant Advice** - Pass to a teammate if he is in a different quadrant that contains no opponents.

Lane Advice (see Figure 5) instructs players to pass to a teammate when the passing lane to that teammate is open. A passing lane is defined as an isosceles triangular region with its apex at the position of the player with the ball and its base midpoint at the position of the intended pass recipient. A lane is open if no opponents are inside the region. The purpose of this advice is to encourage passes that are likely to succeed.

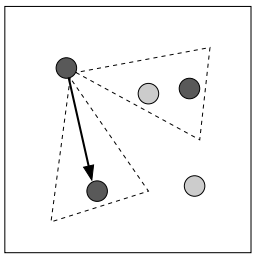

Figure 5: **Lane Advice** - Pass to a teammate if there are no opponents in the corresponding passing lane.

The final piece of advice used in our experiments, **Edge Advice**, differs from the previous advice in that it advises *against* an action rather than for one. As shown in Figure 6, this rule defines edge regions along the sides of the playing field. These regions are each $5m$ wide. A player is advised not to pass to a teammate if both players are in the same edge region. The goal of this advice is to discourage passes along the edges of the play region, which have a high probability of going out of bounds due to noise in the simulator.

Empirical Results when Using Advice

For each piece of advice tested, we ran five learning trials starting from a different random initial state of the learner's function approximator. The results are compared with five learning trials during which no advice is given. For each

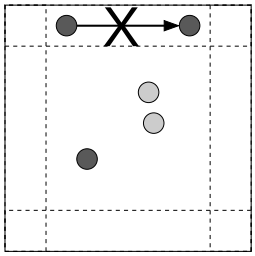

Figure 6: **Edge Advice** - Do not pass along the edges of the play region.

learning trial, we measure the average episode duration over time. Episodes are averaged using a 1000-episode sliding window. We plot the learning curves for every trial on the same graph to give a sense of the variance. The results are shown in Figures 7-10. In all experiments, 3 keepers played against 2 takers on a $20m \times 20m$ field.

By default in the RoboCup simulator, players are only able to see objects that are within a 90^0 view cone. Although we have shown previously that the learning method used in this paper is able to work under this condition (Kuhlmann & Stone 2004), in this work, we have simplified the problem by giving the players 360°. This simplification ensures that the conditions of the advice unit are always accurately evaluated.

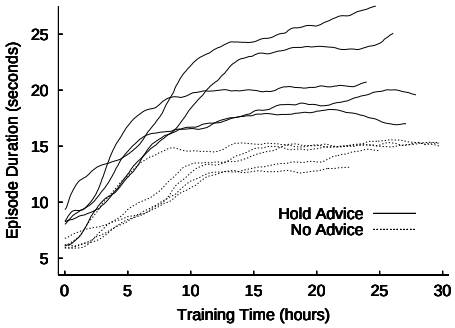

Figure 7: Learning curves comparing **Hold Advice** to no advice.

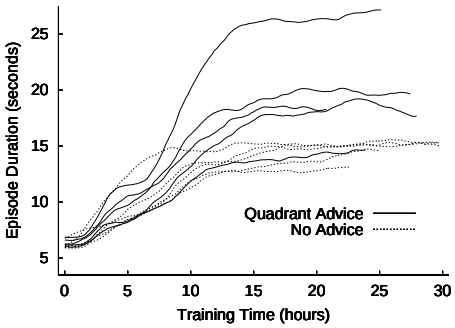

Figure 8: Learning curves comparing **Quadrant Advice** to no advice.

It is clear from Figure 7, that the **Hold Advice** is helpful.

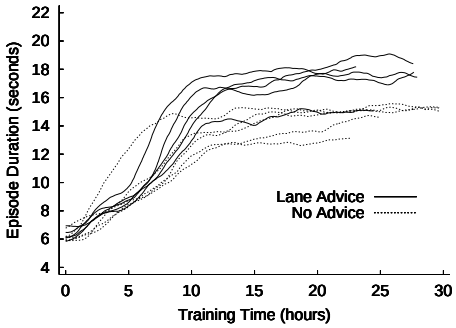

Figure 9: Learning curves comparing **Lane Advice** to no advice.

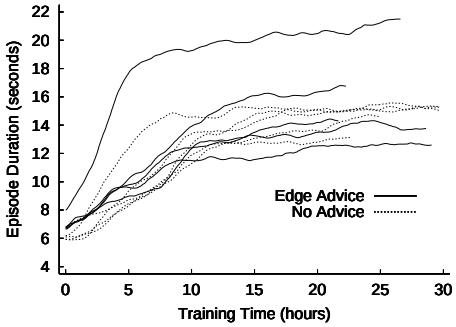

Figure 10: Learning curves comparing **Edge Advice** to no advice.

Learning with this advice consistently outperforms learning without it. However, it is surprising that the players do not learn faster as a result.

Similarly, the results for the **Quadrant Advice** shown in Figure 8 demonstrate that this advice, while not speeding up learning, helps the learners to perform better than without it.

Figure 9 shows that the **Lane Advice** is also helpful. However the performance improvement is not as dramatic as in the previous cases.

Finally, from Figure 10, we see that the learners did not find the **Edge Advice** to be consistently beneficial. However, it appears that in one learning trial, the keepers were able to benefit from the advice.

Additional Experiments

After establishing that several different kinds of advice are beneficial in isolation, we started exploring the possibility of combining the advice. We ran several experiments in which two or more pieces of advice were active at the same time. Typically, while the learners still performed better than with no advice, the results are not as good as those learned with the advice activated individually.

A possible explanation for this result is that in situations in which two pieces of advice that recommend different actions are triggered at the same time, they effectively cancel each other out. Additional work is needed to fully understand and resolve this difficult issue. We plan to continue to explore ways to combine advice to achieve the desired additive effect that has been reported in other work (Maclin 1995).

Conclusion and Future Work

Allowing humans to provide high-level advice to their software assistants is a valuable way to improve the dialog between humans and the software they use. It changes the metaphor from that of giving *commands* to computers to that of giving them *advice*. By their being able to accept, adapt, and even discard advice, advice-taking systems have the potential to radically change how we interact with robots and software agents.

We have empirically investigated the idea of giving advice to an adaptive agent that learns how to use the advice effectively. We show that some simple, intuitive advice can substantially improve a state-of-the-art reinforcement learner on a challenging, dynamic task. Several pieces of advice were shown to improve performance on the RoboCup keepaway task, and we plan to continue extending our work on advisable reinforcement learning to cover the complete simulated RoboCup task.

We are currently investigating additional ways of mapping English statements into formal advice and alternate approaches for using advice in reinforcement learners. We are developing multiple approaches to automatically learning to translate natural-language to semantic representations and we will evaluate them on our assembled English/CLANG corpus. Besides extending how we use advice with a CMAC-based learner (e.g., by modifying the weights in the CMAC directly or changing the learner's exploration function to give higher consideration to advised actions), we are also investigating the use of *knowledge-based support vector machines* (Fung, Mangasarian, & Shavlik 2002) and instructable agents that use *relational* learning methods (Dzeroski, Raedt, & Driessens 2001).

Acknowledgements

We would like to thank Ruifang Ge, Rohit Kate, and Yuk Wah Wong for contributing to the work on natural-language understanding. This research was supported by DARPA Grant HR0011-02-1-0007 and by NSF CAREER award IIS-0237699.

References

Albus, J. S. 1981. *Brains, Behavior, and Robotics*. Peterborough, NH: Byte Books.

Charniak, E. 2000. A maximum-entropy-inspired parser. In *Proceedings of the Meeting of the North American Association for Computational Linguistics*.

Chen, M.; Foroughi, E.; Heintz, F.; Kapetanakis, S.; Kostiadis, K.; Kummeneje, J.; Noda, I.; Obst, O.; Riley, P.; Steffens, T.; Wang, Y.; and Yin, X. 2003. Users manual: RoboCup soccer server manual for soccer server version 7.07 and later. Available at `http://sourceforge.net/projects/sserver/`.

Clouse, J., and Utgoff, P. 1992. A teaching method for reinforcement learning. In *Proceedings of the Ninth International Conference on Machine Learning*, 92–101.

Dzeroski, S.; Raedt, L. D.; and Driessens, K. 2001. Relational reinforcement learning. *Machine Learning* 43:7–52.

Eliassi-Rad, T., and Shavlik, J. 2003. A system for building intelligent agents that learn to retrieve and extract information. *International Journal on User Modeling and User-Adapted Interaction, Special Issue on User Modeling and Intelligent Agents* 13:35–88.

Fung, G. M.; Mangasarian, O. L.; and Shavlik, J. W. 2002. Knowledge-based support vector machine classifiers. In *Advances in Neural Information Processing Systems 14*. MIT Press.

Jurafsky, D., and Martin, J. H. 2000. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Upper Saddle River, NJ: Prentice Hall.

Kate, R. J.; Wong, Y. W.; Ge, R.; and Mooney, R. J. 2004. Learning transformation rules for semantic parsing. under review, available at: `http://www.cs.utexas.edu/users/ml/publication/nl.html`.

Kuhlmann, G., and Stone, P. 2004. Progress in learning 3 vs. 2 keepaway. In Polani, D.; Browning, B.; Bonarini, A.; and Yoshida, K., eds., *RoboCup-2003: Robot Soccer World Cup VII*. Berlin: Springer Verlag.

Maclin, R., and Shavlik, J. 1994. Incorporating advice into agents that learn from reinforcements. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, 694–699.

Maclin, R. 1995. *Learning from instruction and experience: Methods for incorporating procedural domain theories into knowledge-based neural networks*. Ph.D. Dissertation, Computer Sciences Department, University of Wisconsin, Madison, WI.

Noda, I.; Matsubara, H.; Hiraki, K.; and Frank, I. 1998. Soccer server: A tool for research on multiagent systems. *Applied Artificial Intelligence* 12:233–250.

Noelle, D., and Cottrell, G. 1994. Towards instructable connectionist systems. In Sun, R., and Bookman, L., eds., *Computational Architectures Integrating Neural and Symbolic Processes*. Boston: Kluwer Academic.

Norvig, P. 1992. *Paradigms of Artificial Intelligence Programming: Case Studies in Common Lisp*. San Mateo, CA: Morgan Kaufmann.

Siegelmann, H. 1994. Neural programming language. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, 877–882.

Stone, P., and Sutton, R. S. 2001. Scaling reinforcement learning toward RoboCup soccer. In *Proceedings of the Eighteenth International Conference on Machine Learning*, 537–544. Morgan Kaufmann, San Francisco, CA.

Stone, P.; Riley, P.; and Veloso, M. 2000. The CMUnited-99 champion simulator team. In Veloso, M.; Pagello, E.; and Kitano, H., eds., *RoboCup-99: Robot Soccer World Cup III*. Berlin: Springer. 35–48.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press.

Tang, L. R., and Mooney, R. J. 2001. Using multiple clause constructors in inductive logic programming for semantic parsing. In *Proceedings of the 12th European Conference on Machine Learning*, 466–477.

Thompson, C. A., and Mooney, R. J. 2003. Acquiring word-meaning mappings for natural language interfaces. *Journal of Artificial Intelligence Research* 18:1–44.

Zelle, J. M., and Mooney, R. J. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, 1050–1055.