Layered Learning

Peter Stone[1] and Manuela Veloso[2]

[1]AT&T Labs — Research
180 Park Ave.
Florham Park, NJ 07932, USA

[2]Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213, USA

Abstract. This paper presents *layered learning*, a hierarchical machine learning paradigm. Layered learning applies to tasks for which learning a direct mapping from inputs to outputs is intractable with existing learning algorithms. Given a hierarchical task decomposition into *subtasks*, layered learning seamlessly integrates separate learning at each subtask layer. The learning of each subtask directly facilitates the learning of the next higher subtask layer by determining at least one of three of its components: (i) the set of training examples; (ii) the input representation; and/or (iii) the output representation. We introduce layered learning in its domain-independent general form. We then present a full implementation in a complex domain, namely simulated robotic soccer.

1 Introduction

Machine learning (ML) algorithms select a hypothesis from a hypothesis space based on a set of training examples such that the chosen hypothesis is predicted to characterize unseen examples as accurately as possible. Each hypothesis maps a set of input features to a set of output features. Inputs are constructed from information in the domain and outputs are possible classifications or actions.

Our research focuses on learning tasks for which learning a direct mapping from inputs to outputs is intractable given existing learning algorithms. The approach we take is to break the problem down into several hierarchical learning layers such that each layer facilitates the learning of the next. By determining the set of training examples, the input representation, or the output representation, previously learned functions can enable the creation of increasingly complex learned functions. We call this approach to machine learning "layered learning."

Layered learning assumes that the appropriate aspects of the task to be learned are determined as a function of the specific domain. It does not include an automated hierarchical decomposition of the task. Each layer is learned by applying an ML algorithm that is appropriate for the specific subtask characteristics. In this paper, we apply layered learning to a complex multiagent learning task, namely simulated robotic soccer.

We have previously presented the individual learned tasks in this domain [18, 20] as well as a preliminary version of the concept of layered learning [18]. This paper contributes the concrete domain-independent specification of layered learning as presented in Sections 2 and 3. Section 4 reviews our machine learning research in the simulated robotic soccer domain, couching it in the terms of our layered learning specification. This layered learning example is fully implemented and tested as described in Section 5. In Sections 6 and 7, we relate layered learning to previous research and discuss directions for future work.

2 Layered Learning

Table 1 summarizes the principles of our layered learning paradigm which are described in detail in this section.

1. A mapping directly from inputs to outputs is not tractably learnable.
2. A bottom-up, hierarchical task decomposition is given.
3. Machine learning exploits data to train and/or adapt. Learning occurs separately at each level.
4. The output of learning in one layer feeds into the next layer.

Table 1. The key principles of layered learning.

2.1 Principle 1

Layered learning is designed for domains that are too complex for learning a mapping directly from the input to the output representation. Instead, the layered learning approach consists of breaking a problem down into several task layers. At each layer, a concept needs to be acquired. A machine learning algorithm abstracts and solves the local concept-learning task.

2.2 Principle 2

Layered learning uses a bottom-up incremental approach to hierarchical task decomposition. Starting with low-level subtasks, the process of creating new ML subtasks continues until reaching the high-level task that deal with the full domain complexity. The appropriate learning granularity and subtasks to be learned are determined as a function of the specific domain. The task decomposition in layered learning is not automated. Instead, the layers are defined by the ML opportunities in the domain.

2.3 Principle 3

Machine learning is used as a central part of layered learning to exploit data in order to *train* and/or *adapt* the overall system. ML is useful for training functions that are difficult to fine-tune manually. It is useful for adaptation when the task details are not completely known in advance or when they may change dynamically. In the former case, learning can be done off-line and frozen for future use. In the latter, on-line learning is necessary: since the learner needs to adapt to unexpected situations, it must be able to alter its behavior even while executing its task. Like the task decomposition itself, the choice of machine learning method depends on the subtask.

2.4 Principle 4

The key defining characteristic of layered learning is that each learned layer directly affects the learning at the next layer. A learned subtask can affect the subsequent layer by:

- constructing the set of training examples;
- providing the features used for learning; and/or
- pruning the output set.

All three cases are illustrated in our implementation described in Section 4.

3 Formalism

Consider the learning task of identifying a hypothesis h from among a class of hypotheses H which map a set of state feature variables S to a set of outputs O such that, based on a set of training examples, h is most likely (of the hypotheses in H) to represent unseen examples.

When using the layered learning paradigm, the complete learning task is decomposed into hierarchical subtask layers $\{L_1, L_2, \ldots, L_n\}$ with each layer defined as

$$L_i = (\boldsymbol{F_i}, O_i, T_i, M_i, h_i)$$

where:

$\boldsymbol{F_i}$ is the input vector of state features relevant for learning subtask L_i. $\boldsymbol{F_i} = \, <F_i^1, F_i^2, \ldots>$. $\forall j,\ F_1^j \in S$.

$\boldsymbol{O_i}$ is the set of outputs from among which to choose for subtask L_i. $O_n = O$.

$\boldsymbol{T_i}$ is the set of training examples used for learning subtask L_i. Each element of T_i consists of a correspondence between an input feature vector $\boldsymbol{f} \in \boldsymbol{F_i}$ and $o \in O_i$.

$\boldsymbol{M_i}$ is the ML algorithm used at layer L_i to select a hypothesis mapping $\boldsymbol{F_i} \mapsto O_i$ based on T_i.

$\boldsymbol{h_i}$ is the result of running M_i on T_i. h_i is a function from $\boldsymbol{F_i}$ to O_i.

As set out in Principle 2 of layered learning, the definitions of the layers L_i are given a priori. Principle 4 is addressed via the following stipulation. $\forall i < n$, h_i directly affects L_{i+1} in at least one of three ways:

- h_i is used to construct one or more features F_{i+1}^k.
- h_i is used to construct elements of T_{i+1}; and/or
- h_i is used to prune the output set O_{i+1}.

It is noted above in the definition of $\boldsymbol{F_i}$ that $\forall j,\ F_1^j \in S$. Since F_{i+1} can consist of new features constructed using h_i, the more general version of the above special case is that $\forall i, j,\ F_i^j \in S \cup_{k=1}^{i-1} O_k$.

4 Implementation

In this section, we illustrate layered learning via a full-fledged implementation in the RoboCup Soccer Server [14]. Here, the high-level goal is for a team of independently controlled agents to achieve complex collaborative and adversarial behavior. The subtasks are increasingly complex individual and multiagent behaviors.

The purpose of this section is to illustrated layered learning via a fully-implemented system. full details of the domain and each individual learned subtask have been previously reported. However they have not been represented in terms of the formalism presented in Section 3.

4.1 Simulated Robotic Soccer

The RoboCup soccer server [14] has been used as the basis for successful international competitions and research challenges [8]. As presented in detail in [17], 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; the agents must make their decisions in *real-time*; and the actions taken by other agents, both teammates and adversaries, and their resulting state transitions are unknown. We refer to this last quality of unknown state transitions as *opaque transitions*. These italicized domain characteristics combine to make simulated robotic soccer a realistic and challenging domain.

4.2 Layered Learning in Robotic Soccer

Consider the task of a robotic soccer agent retrieving a moving ball and deciding what to do with it. It could dribble the ball, pass to a teammate, or shoot towards the goal. While this task does not encompass the entire robotic soccer task (agents must also decide what to do when they don't have the ball), it comprises an important part of the complete task.

We decompose this task into three learning components or subtasks: ball interception, pass evaluation, and pass selection. Given this hierarchical decomposition, layered learning allows us to create effective team-oriented agent behaviors.

Table 2 illustrates our set of learned behavior levels within the simulated robotic soccer domain. We identify a useful low-level skill that must be learned before moving on to higher-level strategies. Then we build upon it to create higher-level multiagent and team behaviors. Full details regarding the training and testing of each learned behavior are reported in [17].

Layer	**Behavior type**	**Example**
L_1	individual	ball interception
L_2	multiagent	pass evaluation
L_3	team	pass selection

Table 2. Examples of different behavior levels in robotic soccer.

L_1: Ball Interception — an individual skill. First, the agents learn a low-level individual skill that allows them to control the ball effectively. While executed individually, the ability to intercept a moving ball is required due to the presence of other agents: it is needed to block or intercept opponent shots or passes as well as to receive passes from teammates. As such, it is a prerequisite for most ball-manipulation behaviors. We chose to have our agents learn this behavior because it was easier to collect training data than to fine-tune the behavior by hand.[1]

[1] The learning was done in an early implementation of the soccer server (Version 2) in which agents did not receive any velocity information when seeing the ball.

L_1 is defined as follows.

$\boldsymbol{F_1} = \{\boldsymbol{BallDist_t}, \boldsymbol{BallAng_t}, \boldsymbol{BallDist_{t-1}}\}$**:** The agent learns what action to take based on the ball's current distance and angle from the defender, and the ball's distance a fixed time (250 msec.) in the past.

$\boldsymbol{O_1} = \{\boldsymbol{TurnAng}\}$**:** The agent chooses an angle to turn such that it will be likely to intercept the ball.

$\boldsymbol{T_1}$**:** The training procedure for ball interception involves a stationary forward repeatedly shooting the ball towards a defender in front of a goal. The defender collects training examples by acting randomly and noticing when it successfully stops the ball. Test examples are classified as saves (successful interceptions), goals (unsuccessful attempts), and misses (shots that went wide of the goal).

$\boldsymbol{M_1}$ **= a neural network:** Ball interception is trained with a fully-connected neural network with 4 sigmoid hidden units and a learning rate of 10^{-6}. The weights connecting the input and hidden layers use a linearly decreasing weight decay starting at .1%. We use a linear output unit with no weight decay. The neural network was trained for 3000 epochs.

$\boldsymbol{h_1}$ **= a trained interception behavior:** Table 3 shows the effect of the number of training examples on learned save percentage. With about 750 training examples, the defender is able to stop 91% of shots on goal (saves + goals: misses are omitted), a comparable save rate to that achieved when using an analytic ball interception behavior [18].

Training Examples	Saves(%)	Goals(%)	$\frac{\text{Saves}}{\text{Goals+Saves}}$(%)
100	57	33	63
200	73	18	80
300	81	13	86
400	81	13	86
500	84	10	89
750	**86**	9	**91**
1000	83	10	89
4773	84	9	90

Table 3. The defender's performance when using neural networks trained with different numbers of training examples.

$\boldsymbol{L_2}$**: Pass Evaluation — a multiagent behavior.** Second, the agents use their learned ball-interception skill as part of the behavior for training a multiagent behavior. When an agent has the ball and has the option to pass to a particular teammate, it is useful to have an idea of whether or not the pass will actually succeed if executed: will the teammate successfully receive the ball? Such an evaluation depends on not only the teammate's and opponents' positions, but also their abilities to receive or intercept the pass. Consequently, when creating training examples for the pass-evaluation function, we equip the intended pass recipient as well as all opponents with the previously learned ball-interception behavior, h_1. Again, we chose to have our agents learn the pass-evaluation capability because it is easier to collect training data than to construct it by hand.

L_2 is defined as follows.

F_2 = a set of 174 continuous and ordinal features: There are many features that could possibly affect pass evaluation. We encode a large set of attributes representing the relative positions of teammates and opponents on the field as well as statistical counts reflecting their relative positioning [18]. These features are not carefully chosen. On the contrary, many possibly irrelevant features are included, leaving the ML algorithm to select the proper ones. A full list of the attributes can be found in [18].

$O_2 = [-1, 1]$: A potential pass to a particular receiver is classified as a success with a confidence factor $\in (0, 1]$, a failure with a confidence factor $\in [-1, 0)$, or a miss ($= 0$).

T_2: The training procedure for pass evaluation involves a passer kicking the ball towards randomly-placed teammates interspersed with randomly-placed opponents. The training scenario is illustrated within a screen shot of the soccer server in Figure 1. The dashed line indicates the region in which the teammates and opponents are randomly placed. *The intended pass recipient and the opponents all use the learned ball-interception behavior, h_1.* Trials are classified as successes (a teammate intercepts the ball), failures (an opponent intercepts the ball), and misses (no player intercepts the ball). When passing to a random teammate, 51% of passes are successful.

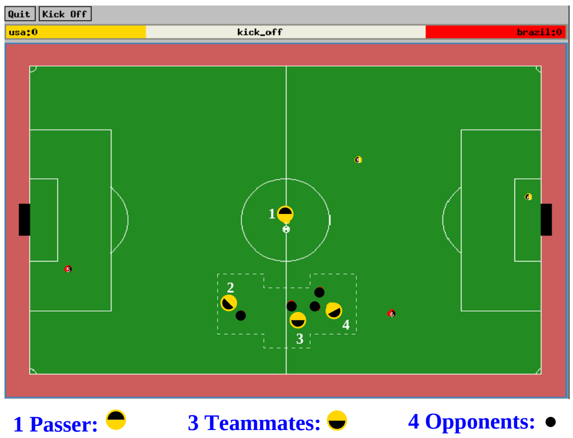

Fig. 1. The training scenario for pass evaluation. The dashed line indicates the region in which the teammates and opponents are randomly placed prior to each trial.

M_2 = C4.5: To learn pass evaluation, we use the C4.5 decision tree training algorithm [15] with all of the default parameters. Decision trees are chosen over neural networks because of their ability to ignore irrelevant input features.

h_2 = a trained pass-evaluating decision tree: During testing, the trained decision tree returns a predicted classification as well as a confidence factor, resulting in a value between -1 and 1. Table 4 tabulates our results indicating that the trained decision tree enables the passer to choose successfully from among its potential receivers. Overall results are given as well as a breakdown by the passer's confidence prior to the pass. In this experiment, the passer is forced to pass even if it predicts failures for all 3 teammates. In that case, it passes to the teammate with the lowest likelihood of failure. 65% of all passes and 79% of passes predicted to succeed with high confidence are successful.

Result	Overall	Success Confidence: .8–.9	.7–.8	.6–.7
(Number)	(5000)	(1050)	(3485)	(185)
SUCCESS (%)	65	**79**	63	58
FAILURE (%)	26	15	29	31
MISS (%)	8	5	8	10

Table 4. The results of 5000 trials during which the passer uses the DT to choose the receiver. Results are given in percentages of the number of cases falling within each confidence interval (shown in parentheses).

L_3: Pass Selection — a collaborative and adversarial team behavior. Third, the agents use their learned pass-evaluation capability h_2 to create the input space and output set for learning pass selection.[2] When an agent has the ball, it must decide to which teammate it should pass the ball.[3] Such a decision depends on a huge amount of information including the agent's current location on the field, the current locations of all the teammates and opponents, the teammates' abilities to receive a pass, the opponents' abilities to intercept passes, teammates' subsequent decision-making capabilities, and the opponents' strategies. The merit of a particular decision can only be measured by the long-term performance of the team as a whole. Therefore, we drastically reduce the input space with the help of the previously learned decision tree, h_2: rather than considering the positions of all of the players on the field, only the pass evaluations for the possible passes to each teammate are considered.

L_3 is defined as follows.

$F_3 = \{PlayerPosition, O_2, O_2, O_2, \ldots\}$: The input representation consists of one coarse geographical component and one action-dependent feature [20] for each possible pass. *The action-dependent features are precisely the result of h_2 executed for each possible receiver.*

$O_3 = \{shoot\} \cup \{Teammates\}$: The result of a pass selection decision is either a shot on goal or a pass to a particular teammate.

T_3: Training examples are gathered on-line by individual team members during real games. Each individual agent learns in a separate partition of F_3

[2] The pass most likely to succeed is not in general the best pass strategically.

[3] It could also choose to shoot. For the purposes of this behavior, the agents are not given the option to dribble.

according to its position on the field. Agents learn based on the observed long-term effects of their actions [17]. *For each particular action decision, the eligible members of O_3 are pruned based on h_2: only passes predicted to succeed are considered.*

$\boldsymbol{M_3}$ **= TPOT-RL:** For training pass selection, we use TPOT-RL [20], an on-line, multiagent, reinforcement learning method motivated by Q-learning that is applicable in team-partitioned, opaque-transition domains such as simulated robotic soccer. We use the default parameters as reported in [20].

$\boldsymbol{h_3}$ **= a distributed pass-selection policy:** We test the pass-selection learning by directly comparing two teams with identical behaviors other than their pass-selection policies. Agents on both teams begin by passing randomly, but agents on one team adjust their behavior based on experience using TPOT-RL. The other agents continue passing randomly. Figure 2 demonstrates the effectiveness of the learned passing policies. Additional tests against goal-directed opponents are reported in [20].

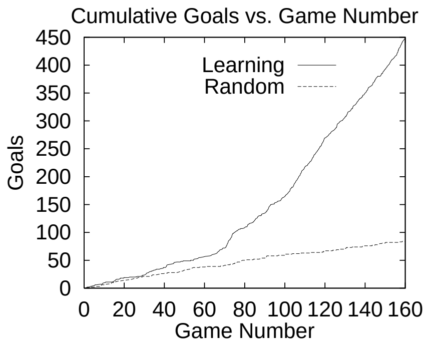

Fig. 2. Total goals scored by a learning team playing against a randomly passing team. The independent variable is the number of 10-minute games that have elapsed.

5 Discussion

In this section, we analyze the key benefits and limitations of layered learning. We also present empirical results of our overall layered learning implementation.

5.1 Analysis

The three learned layers described in Section 4 illustrate the four principles of the layered learning paradigm from Section 2:

1. The decomposition of the task into smaller subtasks enables the learning of a more complex behavior than is possible when learning straight from the agents' sensory inputs.
   Indeed, there have been two attempts at monolithic learning of agent behaviors in the soccer server. First, Luke et al. [11] set out to create a completely learned team of agents using genetic programming [9]. However, the ambition was eventually scaled back and low-level player skills were created by hand as

the basis for learning. The resulting learned team won two of its four games at the international RoboCup-97 robotic soccer simulator competition, losing in the second round. The following year, at RoboCup-98, another genetic programming attempt at learning the entire team behavior was made [1]. This time, the agents were indeed allowed to learn directly from their sensory input representation. While making some impressive progress given the challenging nature of the approach, this entry was unable to advance past the first round in the tournament.

2. The hierarchical task decomposition is constructed in a bottom-up, domain-dependent fashion. The fact that the the task decomposition needs to be provided to layered learning a priori our paradigm's main limitations, and it is this characteristic that leads us to describe layered learning as a "paradigm" or a "method" as opposed to an "algorithm." Automatically selecting abstractions for learning is still a challenging open problem.
   However, layered learning could be combined with any algorithm for *generating* abstraction levels to create an abstraction selection routine. In particular, let A be an algorithm for generating task decompositions within a domain. Suppose that A does not have an objective metric for comparing different decompositions. Applying layered learning on the task decomposition and quantifying the resulting performance can be used as a measure of the utility of A's output.
3. Learning methods are chosen or created to suit the subtask. They exploit available data to train difficult behaviors (ball interception and pass evaluation) or to adapt to changing/unforeseen circumstances (pass selection).
   Again, this need to select the ML algorithm by hand is a limitation of layered learning. Automatically mapping from tasks to ML algorithms is another challenging open problem in the field. However, the flexibility to use any algorithm to match the needs of the subtask is an important characteristic of layered learning. For example, we exploited the ability of neural networks to learn continuous output values in L_1 of our robotic soccer implementation, used C4.5 to ignore irrelevant input features in L_2, and created a multiagent learning algorithm capable of learning from limited training data in L_3.
4. Learning in one layer feeds into the next layer either by providing a portion of the behavior used for training (ball interception – pass evaluation) or by creating the input representation and pruning the action space (pass evaluation – pass selection).
   This last characteristic is a key principle of layered learning. It specifies how each successive subtask can leverage off of the learning of previous subtasks.

5.2 Results

The layered learning approach has contributed to our success at the first three international RoboCup robotic soccer competitions.[4] Although competitions are

[4] Robust low-level skills and a sophisticated team member agent architecture [19] also contributed significantly. We thank Patrick Riley for his implementation of the low-level skills [21].

not controlled testing scenarios and they do not provide means for isolating the positive and negative aspects of an approach, they do allow for evaluation of an overall implementation. We present our results at these competitions as supporting evidence, rather than proof, that layered learning is effective. Note that all of the individual learned layers described in Section 4 were validated in controlled experiments.

At the first robotic soccer world cup competition, RoboCup-97 [7], our team made it to the semi-finals in a field of 29 teams. At RoboCup-98 [2], our team won in a field of 34 teams. And at RoboCup-99 [22], our team repeated as champion in a field of 37 teams. Full details of the competitions are available at www.robocup.org.

6 Related Work

The original hierarchical learning constructs were devised to improve the generalization of a single learning task by running multiple learning processes. Both boosting [16] and stacked generalization [23] improve function generalization by combining the results of several generalizers or several runs of the same generalizer. These approaches contrast with layered learning in that the layers in layered learning each deal with *different* tasks. Boosting or stacked generalization could potentially be used within any given layer, but not across different layers.

More in line with the type of hierarchical learning discussed in this paper are hierarchical reinforcement learning algorithms. Because of the well-known "curse of dimensionality" in reinforcement learning RL researchers have been very interested in hierarchical learning approaches. As surveyed in [6], most hierarchical RL approaches use *gated* behaviors:

> There is a collection of behaviors that map environment states into low-level actions and a gating function that decides, based on the state of the environment, which behavior's actions should be switched through and actually executed. [6]

In some cases the behaviors are learned [13], in some cases the gating function is learned [12], and in some cases both are learned [10]. In this last example, the behaviors are learned and fixed prior to learning the gating function. On the other hand, feudal Q-learning [3] and the MAXQ algorithm [4] learn at all levels of the hierarchy simultaneously. A constant among these approaches is that the behaviors and the gating function are all control tasks with similar inputs and actions (sometimes abstracted). In the RL layer of our layered learning implementation, the input representation itself is learned. In addition, none of the above methods has been implemented in a large-scale, complex domain.

In all of the above RL approaches, like in layered learning, the task decomposition is constructed manually. However, there has been at least one attempt at the challenging task of learning the task decomposition. Nested Q-learning [5] generates its own hierarchical control structure and then learns low-level skills at the same time as it learns to select among them. Thus far, like other hierarchical RL approaches, it has only been tested on very small problems (on the order of 100 states in this case).

7 Conclusion and Future Work

This paper has presented the layered learning paradigm and illustrated it with a fully-implemented example in the robotic soccer domain. Our layered learning implementation, along with robust low-level skills and a sophisticated team member agent architectures which incorporates a flexible teamwork structure [19], has contributed to the success of our complete team of simulated robotic soccer competitions.

An important direction for future work is to apply layered learning in a new domain. As an example apparently orthogonal to robotic soccer, consider natural language understanding as another application of layered learning. Natural language understanding can have a clear hierarchical task decomposition. For example, learned word sense disambiguation could facilitate learned sentence parsing, which in turn could facilitate semantic encoding of sentences or paragraphs (see Table 5). While it is currently not possible in general to learn sentence semantics straight from a string of words, a hierarchical decomposition of the task coupled with the layered learning paradigm may render the learning task tractable. Indeed, layered learning is potentially applicable to any complex learning problem for which a hierarchical decomposition exists.

Layer	**Learning Task**
L_1	Word sense disambiguation
L_2	Sentence syntax
L_3	Sentence semantics

Table 5. Natural language understanding: a proposed layered learning application.

Layered learning is potentially applicable to this and other tasks that are too complex for monolithic learning. Its power is derived from the concept of directly combining different ML algorithms within a hierarchically decomposed task representation.

References

1. David Andre and Astro Teller. Evolving team Darwin United. In Minoru Asada and Hiroaki Kitano, editors, *RoboCup-98: Robot Soccer World Cup II*. Springer Verlag, Berlin, 1999.
2. Minoru Asada and Hiroaki Kitano, editors. *RoboCup-98: Robot Soccer World Cup II*. Lecture Notes in Artificial Intelligence 1604. Springer Verlag, Berlin, 1999.
3. Peter Dayan and Geoffrey E. Hinton. Feudal reinforcement learning. In S. J. Hanson, J. D. Cowan, and C. L. Giles, editors, *Advances in Neural Information Processing Systems 5*. Morgan Kaufmann, San Mateo, CA, 1993.
4. Thomas G. Dietterich. The MAXQ method for hierarchical reinforcement learning. In *Proceedings of the Fifteenth International Conference on Machine Learning*. Morgan Kaufmann, 1998.
5. Bruce L. Digney. Emergent hierarchical control structures: Learning reactive/hierarchical relationships in reinforcement environments. In *Proceedings of*

the 4th International Conference of Simulation of Adaptive Behavior, pages 363–372. MIT Press, 1996.
6. Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, May 1996.
7. Hiroaki Kitano, editor. *RoboCup-97: Robot Soccer World Cup I.* Springer Verlag, Berlin, 1998.
8. Hiroaki Kitano, Milind Tambe, Peter Stone, Manuela Veloso, Silvia Coradeschi, Eiichi Osawa, Hitoshi Matsubara, Itsuki Noda, and Minoru 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.
9. John R. Koza. *Genetic Programming.* MIT Press, 1992.
10. Long-Ji Lin. *Reinforcement Learning for Robots Using Neural Networks.* PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1993.
11. Sean Luke, Charles Hohn, Jonathan Farris, Gary Jackson, and James Hendler. Co-evolving soccer softbot team coordination with genetic programming. In Hiroaki Kitano, editor, *RoboCup-97: Robot Soccer World Cup I*, pages 398–411, Berlin, 1998. Springer Verlag.
12. Pattie Maes and Rodney A. Brooks. Learning to coordinate behaviors. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 796–802. Morgan Kaufmann, 1990.
13. Sridhar Mahadevan and Jonathan Connell. Scaling reinforcement learning to robotics by exploiting the subsumption architecture. In *Proceedings of the Eighth International Workshop on Machine Learning*, pages 328–332, 1991.
14. Itsuki Noda, Hitoshi Matsubara, Kazuo Hiraki, and Ian Frank. Soccer server: A tool for research on multiagent systems. *Applied Artificial Intelligence*, 12:233–250, 1998.
15. J. Ross Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, San Mateo, CA, 1993.
16. Robert E. Shapire. The strength of weak learnability. *Machine Learning*, 5:197–227, 1990.
17. Peter Stone. *Layered Learning in Multi-Agent Systems.* PhD thesis, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, December 1998. Available as technical report CMU-CS-98-187.
18. Peter Stone and Manuela Veloso. A layered approach to learning client behaviors in the RoboCup soccer server. *Applied Artificial Intelligence*, 12:165–188, 1998.
19. Peter Stone and Manuela Veloso. Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork. *Artificial Intelligence*, 110(2):241–273, June 1999.
20. Peter Stone and Manuela Veloso. Team partitioned, opaque transition reinforcement learning. In *Proceedings of the Third Annual Conference on Autonomous Agents*, pages 206–212. ACM Press, May 1999.
21. Peter Stone, Manuela Veloso, and Patrick Riley. The CMUnited-98 champion simulator team. In Minoru Asada and Hiroaki Kitano, editors, *RoboCup-98: Robot Soccer World Cup II.* Springer Verlag, Berlin, 1999.
22. Manuela Veloso, Enrico Pagello, and Hiroaki Kitano, editors. *RoboCup-99: Robot Soccer World Cup III.* Springer Verlag, Berlin, 2000. To appear.
23. David H. Wolpert. Stacked generalization. *Neural Networks*, 5:241–259, 1992.