Chapter 4

Layered Learning

As described in Section 3.2, our agents build up a world model and then use it as the basis for behaviors that respond effectively to the environment. Internal behaviors update the internal state while external behaviors produce executable actuator commands. Spanning both internal and external behaviors, *layered learning* [Stone and Veloso 98c] is our bottom-up hierarchical approach to agent behaviors that allows for machine learning at the various levels.

The introduction and implementation of layered learning is one of the main contributions of this thesis. Layered learning is a machine learning paradigm defined as a set of principles for the construction of a hierarchical, learned solution to a complex task. This chapter lays out the principles of layered learning (Section 4.1) and gives an overview of the implementation which is detailed in Chapters 5–7 (Section 4.2). In Section 4.3, I discuss some of the general multi-agent learning issues that arise within layered learning and within our implementation in particular. Section 4.4 presents related work.

4.1 Principles

Layered learning is defined by four principles. In this section, I identify, motivate, and specify these four principles.

Principle 1: Motivated by robotic soccer, layered learning is designed for domains that are too complex for learning a mapping directly from an agent's sensory inputs to its actuator outputs. We assume that any domain that fits the description in Section 1.2.1—limited communication, real-time, noisy environments with both teammates and adversaries—is too complex for agents to learn direct mappings from their sensors to actuators.

Instead, the layered learning approach consists of breaking a problem down into several behavioral layers and using machine learning (ML) techniques at each level. Layered learning uses a bottom-up incremental approach to hierarchical task decomposition. Starting with low-level behaviors, the process of creating new ML subtasks continues until reaching high-level strategic behaviors that deal with the full domain complexity.

Principle 2: The appropriate behavior granularity and the aspects of the behaviors 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.

Layered learning can, however, be combined with any algorithm for learning abstraction levels. In particular, let A be an algorithm for learning 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.

Figure 4.1 illustrates an abstract layered learning task decomposition within a collaborative and adversarial multi-agent domain. Learning can begin with individual behaviors, which facilitate multi-agent collaborative behaviors, and eventually lead to full-team collaborative and adversarial behaviors.

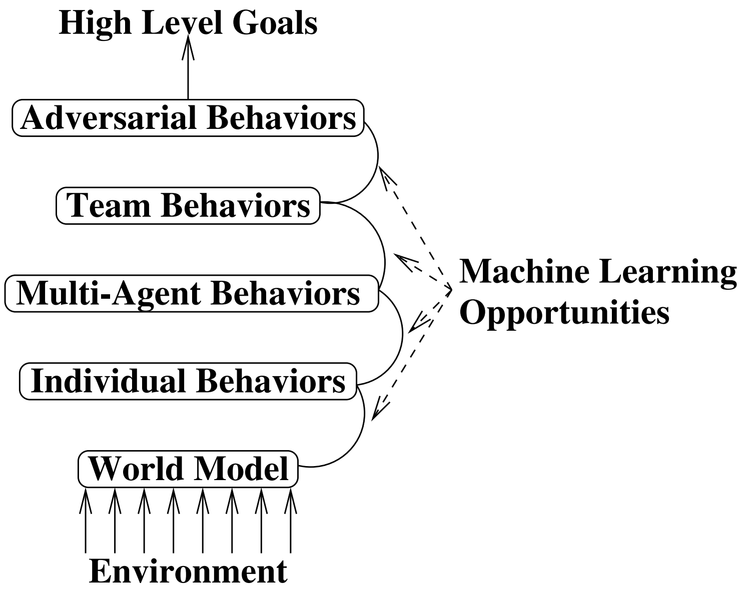

Figure 4.1: A sample task decomposition within the layered learning framework in a collaborative and adversarial multi-agent domain. Layered learning is designed for use in domains that are too complex to learn a mapping straight from sensors to actuators. We use a hierarchical, bottom-up approach

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 behaviors 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 during actual task execution. In the latter, on-line learning is necessary: since the agent 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.

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 either (i) by providing a portion of the behavior used during training or (ii) by creating the input representation of the learning algorithm. In general, machine learning algorithms—e.g. neural networks, Q-learning [Watkins 89], and decision trees [Quinlan 93]—require an input and output representation, a target mapping from inputs to outputs, and training examples. The goal of learning is to generalize the target mapping from the training examples which provide the correct outputs for only a portion of the input space.

When using ML for behavior learning (as opposed to classification), training examples are generated by placing an agent in a situation corresponding to a specific instance of the input representation; allowing it to act; and then giving some reward, or indication of the merit of the action in the context of the target mapping. Thus previous learned layers can (i) provide a portion of the behavior used during training by either determining the actions available or affecting the reinforcement received. Previous learned layers can also (ii) create the inputs to the learning algorithm by affecting or determining the agent's input representation. Both possibilities are illustrated in our simulated robotic soccer implementation described in Section 4.2.

If each learned layer in a task decomposition directly affects the learning at the next layer, then the system is a layered learning system, even if the domain does not have identical properties to those considered in this thesis. Without this characteristic, the approach does not fall within the realm of layered learning.

In summary, layered learning is a machine learning paradigm designed to allow agents to learn to accomplish a goal in a complex environment. Layered learning allows for a bottom-up definition of agent capabilities at different levels in a complex domain. Machine learning opportunities are identified when data is available or the task is unpredictable and hand-coded solutions are too complex to generate. Individual learned behaviors are organized, learned, and combined in a layered fashion, each facilitating the creation of the next. The principles of the layered learning technique are summarized in Table 4.1.

1. A mapping directly from sensors to actuators 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 4.1: The key principles of layered learning.

4.2 Instantiation in Simulated Robotic Soccer

One tempting way to approach any new agent-based domain is to try to learn a direct mapping from the agent's sensors to its actuators. However, a quick consideration of the robotic soccer domain is enough to convince oneself that it is too complex for such an approach: the space of possible sensory inputs is huge, there are many possible actions, and there is a large amount of hidden state. Such complexity is an important characteristic of the domain for the purposes of this thesis, since robotic soccer is meant to represent other domains which are too complex for the straightforward approach.

Table 4.2 illustrates a possible set of learned behavior levels within the simulated robotic soccer domain that correspond to the abstract task decomposition represented in Figure 4.1. 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 multi-agent and team behaviors. Using our own experience and insights to help the agents learn, we acted as human coaches do when they teach young children how to play real soccer. Section 4.2 gives an overview of our layered learning implementation in the simulated robotic soccer domain and discusses how the implementation could be extended further.

Layer	Strategic level	Behavior type	Example
1	robot–ball	individual	ball interception
2	one-to-one player	multi-agent	pass evaluation
3	one-to-many player	team	pass selection
4	team formation	team	strategic positioning
5	team-to-opponent	adversarial	strategic adaptation

Table 4.2: Examples of different behavior levels in robotic soccer.

4.2.1 Implemented Learned Layers

Our implementation consists of three learned subtasks, each of which is described more fully along with extensive empirical tests later in the thesis:

1. Ball Interception — an individual skill (Chapter 5).

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]. Since the skill does not need to change during the course of play, we are able to use off-line learning, specifically a neural network with four inputs.

[1]The learning was done in an earlier implementation of the soccer server in which agents did not receive any velocity information when seeing the ball. Thus the ball interception skill described in Appendix B was not applicable.

2. **Pass Evaluation** — a multi-agent behavior (Chapter 6).

   Second, the agents use their learned ball-interception skill as part of the behavior for training a multi-agent 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. Again, we choose to have our agents learn the pass-evaluation capability because it is easier to collect training data than to construct it by hand. Working under the assumption that the concept would remain relatively constant from game to game, we again use an off-line learning algorithm, namely the C4.5 decision tree training algorithm [Quinlan 93]. Decision trees are chosen over neural networks because of their ability to ignore irrelevant attributes: in this case, our input representation has 174 continuous attributes.

3. **Pass Selection** — a collaborative and adversarial team behavior (Chapter 7).

   Third, the agents use their learned pass-evaluation capability to create the input space for learning a team behavior. When an agent has the ball, it must decide to which teammate it should pass the ball[2]. 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. In this sense, Q-learning, a reinforcement learning method for maximizing long-term discounted reward, seemed applicable. However, Q-learning is known for working poorly with large input representations. Therefore, we drastically reduce the input space with the help of the previously learned decision tree: rather than considering the positions of all of the players on the field, only the pass evaluation for the possible passes to each teammate are considered. Nonetheless, due to the multi-agent and opaque transition characteristics of the task, Q-learning still does not apply directly. Instead, we create TPOT-RL, a new multi-agent reinforcement learning method motivated by Q-learning. In this case, since the learned behavior is meant to adapt to a dynamically changing environment (changing in part because teammates are learning their decision-making policies simultaneously), we need an on-line learning method, which TPOT-RL is.

The learning methods used for each of the above behaviors are summarized in Table 4.3. Possible implementations of the last two layers in Table 4.2 are future work and are described in Section 4.2.2.

The three learned layers described above illustrate the principles of the layered learning paradigm as laid out in Section 4.1:

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

Layer	Learned behavior	Learning method	Training type
1	ball interception	neural network	off-line
2	pass evaluation	decision tree	off-line
3	pass selection	TPOT-RL	on-line

Table 4.3: The learning methods used for the implemented layers in the simulated robotic soccer layered learning implementation.

- The decomposition of the task into smaller subtasks enables the learning of a more complex behavior than would be possible if learning straight from the agents' sensory inputs (see Section 4.4 for supporting evidence).

- The hierarchical task decomposition is constructed in a bottom-up, domain-dependent fashion.

- Machine learning methods are chosen or created to suit the subtask in question. They exploit available data to train difficult behaviors (ball interception and pass evaluation) or to adapt to changing/unforeseen circumstances (pass selection).

- 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 (pass evaluation – pass selection).

4.2.2 Future Learned Layers

This section carries the robotic soccer example of layered learning beyond its implementation in this thesis, describing how the principles of layered learning could be used to create additional, higher-level learned layers. In particular, I discuss how to implement the last two layers in Table 4.2—strategic positioning and strategic adaptation—on top of the currently-implemented learned layers.

The strategic positioning and strategic adaptation behaviors build upon two characteristics of the implementation of the pass-selection learning that are not described above:

- When using TPOT-RL, the simulated robotic soccer agents stay in a fixed formation and do not switch positions.

- TPOT-RL training is done against a fixed opponent.

In contrast, the following new behaviors would facilitate the learning of the agent positioning which best suits the pass-selection algorithm and the learning of which learned policy to use against a new opponent.

4. Strategic Positioning — a team behavior.

The team of agents could use their pass-selection behavior as part of the training behavior for creating another team behavior. As described in Chapter 3, the team can play in any of a number of different formations. Each player also has some flexibility within its position. Our training of the pass-selection behavior uses players in an arbitrarily chosen fixed formation. However, there is no reason to believe that this formation is the best for learning against a particular opponent. If allowed to use the TPOT-RL learning algorithm while simultaneously adjusting their formation, the team may be able to perform even better. The position adjustments could be done using observational reinforcement learning (RL), another on-line algorithm [Andou 98].

5. **Strategic Adaptation** — an adversarial behavior.

   Finally, the agents comprising the team could use their combined strategic-positioning/pass-selection behaviors as the input representation for strategic adaptation. The previous behavior is trained against a fixed opponent. However, there are many different possible opponent strategies. By training the strategic-positioning/pass-selection behavior against a variety of opponents, several different policies could be learned. Then, the opportunity arises to try to match the current opponent with the most similar past opponent as quickly as possible so that the previously learned policy can be adopted. A possible off-line learning approach to this problem would be to characterize teams by a set of performance statistics (e.g. average player location) over the course of a fixed time period. Based on this representation, a memory-based algorithm could be used to match teams with the previously seen opponents that behave most similarly. Then during games, after the fixed evaluation period has identified a similar past opponent, the pass-selection policy that was learned against this similar opponent team could be adopted by the agents.

The learning methods proposed for both of the above behaviors are summarized in Table 4.4. Again, just as is the case for the implemented behaviors, these new behaviors follow the principles of layered learning as laid out in Section 4.1.

Layer	Learned behavior	Learning method	Training type
4	strategic positioning	observational RL	on-line
5	strategic adaptation	memory-based	off-line

Table 4.4: The proposed learning methods for the future layers in the simulated robotic soccer layered learning implementation.

4.3 Discussion

In this section I discuss some issues pertaining to layered learning and our implementation thereof as they relate to multi-agent learning in general. I begin by bringing up several issues related to the relative merits of, and appropriate times for, off-line and on-line learning. I also frame the above implementation within our team member agent architecture and discuss the possible propagation of errors among the learned behaviors in a layered learning system.

4.3.1 On-line and Off-line Learning

As illustrated in Section 4.2, the type of learning used at each layer in layered learning depends upon the subtask characteristics. We use neural networks and decision trees to learn ball interception and pass evaluation. These off-line approaches are appropriate for fixed tasks that can be trained outside of game situations. In fact, off-line learning is necessary for such basic skills due to the fact that no individual gets enough training examples in game situations to learn them on-line.

This off-line learning of basic skills is similar to chunking [Newell 90], something done by human experts in many domains. Performing tasks closely related to simulated robotic soccer, human athletes spend a good deal of time acquiring basic skills so that they can execute them "automatically" during competition. For example, human soccer players, like our robotic soccer agents, learn to control and pass a ball through hours and hours (over many years) of practice outside of game situations.

Similarly, it has been documented that chess experts, rather than reasoning from first principles all the time, learn to recognize common patterns of pieces on the board [Newell and Simon 72]. Their chunking allows them to quickly eliminate or focus on certain lines of play so that they can efficiently evaluate their options on a given move.

Unlike human learners, when AI agents use off-line learning, there is the opportunity for them to share their learned knowledge with other team members. Rather than each individual agent needing to learn a particular skill, a single agent can do the learning and transfer knowledge to teammates. In fact, we use this method of knowledge sharing for both the ball-interception and pass-evaluation layers.

In PTS domains (see Chapter 3) such as robotic soccer, even on-line learning can be shared to some extent. While the agents may learn different things due to their different experiences during execution, they can still share their experiences completely at the periodic team synchronization opportunities. This approach is the one we take for the learned pass-selection behavior: each agent learns a policy while staying in a fixed position within a fixed formation. However, when given the opportunity, the agents share their learned policies so that they can subsequently switch positions freely.

This sharing of all learned information leads to completely homogeneous agents. It is in contrast with attempts to promote behavioral diversity in learning teams of agents [Balch 98]. While it is important that not all agents in a team are doing exactly the same thing at the same time, there is no need to enforce difference in behavior *capabilities* or knowledge. Rather than using heterogeneous agents, we have our agents decompose and share their task via their flexible team formations. Although at any given time each agent fills a different role, each agent is capable of switching to any other role.

Two other common issues related to on-line multi-agent learning are exploration versus exploitation and the escalating arms race problem. On-line behavior learning methods, whether or not they involve multiple agents, must contend with the tradeoff between exploration and exploitation. Exploration is acting sub-optimally given the available information in order to gather additional information. On the other hand, exploitation is always taking the best available action given the current state of knowledge at the cost of possibly missing new opportunities. Particularly in simulated robotic soccer, where games last only 10 minutes, agents can only afford to explore on-line briefly. In fact, even our on-line pass-selection training is executed over the course of several games against a fixed opponent using Boltzmann exploration: a probabilistic exploration method that gradually increases the rate of exploitation. And the proposed strategic adaptation learning method is executed off-line with only a brief (perhaps one minute) on-line period to gather performance statistics for the current opponent. In order to adapt on-line within the course of a single game, the learning method must be capable of almost immediate exploitation.

The escalating arms race problem is a phenomenon occurring in on-line multi-agent learning systems with concurrently adapting opponents. As one team changes its strategy to counteract the current opponent strategy, the opponent simultaneously alters its strategy to improve its own performance. In this way, the target concept to be learned constantly shifts and neither team is able to converge to a stable strategy. Our layered learning implementation sidesteps this issues by training against fixed opponents. We have not attempted to learn against another adaptive team.

4.3.2 Framing within the Team Member Agent Architecture

In the context of the team member agent architecture presented in Chapter 3, the implemented ball-interception and pass-selection layers are external behaviors, directly affecting the agent's actions. On the other hand, the pass-evaluation layer affects the agent's internal state: it is an internal behavior. Strategic positioning and strategic adaptation are also internal behaviors as they affect aspects of the formation and overall strategy which are components of the internal state via the locker-room agreement.

4.3.3 Error Propagation

An issues that is often raised in regards to layered learning is the problem of error propagation. That is, since the learning at each layer affects the learning at subsequent layers, if something goes wrong, how can we determine which layer is to blame? Or conversely, if the team performs well, which layer deserves the credit? While this question is worth careful examination, such an examination is not a part of this thesis. When testing the effectiveness of each new learned layer, we isolate it by comparing agent performance with and without the new layer. Thus we validate that the learning does indeed improve performance. However any particular error that does occur with the new layer could be due to that layer or to any previous layer.

One often overlooked fact with regard to error propagation is that each additional layer does not necessarily add to the overall error rate. Instead, it is possible for higher-level behaviors to learn to overcome errors in the underlying layers. For example, in our robotic soccer implementation, the pass-evaluation function may be able learn which passes will be likely failures for the ball-interception function and therefore degrade the predicted likelihood of completing the pass. Similarly, the pass-selection routine may be able to compensate for faulty evaluations by the pass-evaluation layer by, for example, requiring higher success confidences for passes in critical situations.

A detailed study of error propagation and compensation within layered learning implementations is a promising area for future research.

4.4 Related Work

In this section, I briefly discuss some of the most related work to layered learning along three dimensions. (Related work for the thesis as a whole is discussed in detail in Chapter 9.) First, I substantiate my claim that robotic soccer is too complex a domain for learning straight from sensors to actuators. Second, I present other layered architectures. Third, I bring some other hierarchical learning approaches to attention.

4.4.1 Robotic Soccer

One of the principles of layered learning as laid out in Section 4.1 is that it is to be used in domains that are too complex for learning a mapping directly from sensors to actuators. In general, we assume that any domain that fits the description in Section 1.2.1—limited communication, real-time, noisy environments with both teammates and adversaries—is such a domain. In particular, empirical evidence indicates that robotic soccer is such a domain: layered learning enables the creation of more successful behaviors than could be achieved if using the agents' sensors as the input representation for a single, all-encompassing learned behavior.

This evidence is based on two attempts to do just that—learn straight from the agents' sensors—in the same robotic soccer domain. First, Luke et al. [98] set out to create a completely learned team of agents using genetic programming [Koza 92]. 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 RoboCup-97 competition, losing in the second round. In that same competition, our CMUnited-97 team, using layered learning, made it to the semi-finals (fourth round) and, although never directly matched against the genetic programming team, was qualitatively clearly a better team.

The following year, at RoboCup-98, another genetic programming attempt at learning the entire team behavior was made [Andre and Teller 99]. 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.

Given the complex nature of the robotic soccer domain, our initial inclination was that it would be impossible to create successful behaviors by learning straight from sensors to actuators: the space of possible sensory inputs is huge, there are many possible actions, and there is a large amount of hidden state. Combined with the anecdotal evidence provided by the two generally unsuccessful attempts at such learning, we feel confident in our claim that layered learning's decomposition of the robotic soccer task into smaller subtasks enables the creation of more complex and successful learned behaviors than are possible if learning straight from the agents' sensory inputs.

For a further discussion of other learning approaches within the robotic soccer domain, see Chapter 9 (Section 9.2).

4.4.2 Layered Architectures

The layered learning approach is somewhat reminiscent of Brooks' subsumption architecture [Brooks 86] which layers reactive control modules, allowing high-level controllers to override lower-lever ones. Each control level is capable of controlling the robot on its own up to a specified level of functionality. Brooks implements his approach on real robots, building controllers for simple tasks such as avoiding collisions and wandering around.

Mataric brings the subsumption architecture to a multi-agent learning domain, building controllers on top of a set of learned *basis behaviors* [Mataric 95]. Mataric's basis behaviors are chosen to be necessary and sufficient for the learning task, while remaining as simple and robust as possible. Since Mataric's robots were to learn social behaviors such as flocking and foraging, they were equipped with basis behaviors such as the ability to follow each other and the ability to wander without running into obstacles.

While layered learning also makes use of multiple behavior layers, the robotic soccer task is much more complex: the agents must be able to generalize across situations, handle adversaries, and achieve complex goals. In order to move quickly to high-level behaviors, the commitment to have every layer be completely able to control the robot is abandoned. Instead, many situation-specific (but as general as possible) behaviors are produced which are then managed by higher-level behaviors. In addition, the robotic soccer behaviors are not entirely reactive, building upon the agents' internal state which changes over time. Nevertheless, the idea of building higher levels of functionality on top of lower levels is retained. It is in producing the situation-specific behaviors that ML techniques are used.

Also building on top of the subsumption architecture, the ALLIANCE systems was built to deal with "loosely-coupled, largely independent tasks" for heterogeneous robots [Parker 94]. Example tasks are box pushing, janitorial service (dust, empty trash, clean floors), and hazardous waste cleanup. ALLIANCE assumes that the robots have the abilities to do each of the low-level tasks; it provides the coordination mechanism. The L-ALLIANCE variant learns some of the system parameters indicating, for instance, how good each robot is at performing each task. Like in Mataric's work, the learning in L-ALLIANCE is at the level of choosing among hard-wired behaviors: the learning does not cross multiple layers in the architecture. In addition, the tasks considered are neither real-time nor adversarial.

As opposed to the purely reactive subsumption architecture, three-layer architectures include behaviors that refer to an agent's internal state. The three layers are the *controller*, which executes purely reactive behaviors mapping sensors to actuators; the *sequencer*, which references past world states in order to select which reactive behavior to use at a given time; and the *deliberator*, which predicts future states in order to do time-consuming computations such as "planning and other exponential search-based algorithms" [Gat 98].

INTERRAP [Mueller 96] is a recent example of a three-layer architecture which is designed for multi-agent systems. In INTERRAP, the highest layer includes collaborative reasoning using a BDI [Rao and Georgeff 95] model to consider other agents' goals and intentions and to resolve conflicts. INTERRAP does not include any machine learning. Indeed, learning is not a consideration in three-layer architectures in general [Gat 98].

The layers in layered learning are orthogonal to the layers in a three-layer architecture. Layered learning could easily be used within a three-layer approach with learned layers intersecting the controller, the sequencer, and the deliberator in any way. While our team member agent architecture is not a three-layer approach since there are no purely reactive behaviors (even the lowest-level skills rely on predictions of future world states based on current and past states), robotic soccer could be implemented within the three-layer paradigm while retaining the same learned layers: ball interception would be a part of the controller; pass evaluation and pass selection would be part of the sequencer; and strategic positioning and strategic adaptation would be part of the deliberator.

Layered learning can be used within existing layered architectures or within most other robotic architectures. As long as there are behaviors which build upon each other in some ways, then learning in one behavior can be used to facilitate or influence learning in subsequent behaviors.

4.4.3 Hierarchical Learning

The original hierarchical learning constructs were devised to improve the generalization of a single learning task by running multiple learning processes. Both boosting [Shapire 90] and stacked generalization [Wolpert 92] 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 thesis are hierarchical reinforcement learning algorithms. Because of the well-known "curse of dimensionality" in reinforcement learning—RL algorithms require time that is polynomial in the number of states [Dietterich 98]—RL researchers have been very interested in hierarchical learning approaches. As surveyed in [Kaelbling *et al.* 96], 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. [Kaelbling *et al.* 96]

In some cases the behaviors are learned [Mahadevan and Connell 91], in some cases the gating function is learned [Maes and Brooks 90], and in some cases both are learned [Lin 93]. In this last example, the behaviors are learned and fixed prior to learning the gating function. On the other hand, feudal Q-learning [Dayan and Hinton 93] and the MAXQ algorithm [Dietterich 98] 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 [Digney 96] 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).

The hierarchical RL research mentioned in this section is all done with single learning agents. For a full discussion of multi-agent learning approaches, see Chapter 9 (Section 9.1).