1 Supervised Actor-Critic Reinforcement Learning

MICHAEL T. ROSENSTEIN and ANDREW G. BARTO
Department of Computer Science
University of Massachusetts, Amherst

Editor's Summary: Chapter **??** introduced policy gradients as a way to improve on stochastic search of the policy space when learning. This chapter presents supervised actor-critic reinforcement learning as another method for improving the effectiveness of learning. With this approach, a supervisor adds structure to a learning problem and supervised learning makes that structure part of an actor-critic framework for reinforcement learning. Theoretical background and a detailed algorithm description are provided, along with several examples that contain enough detail to make them easy to understand and possible to duplicate. These examples also illustrate the use of two kinds of supervisors: a feedback controller that is easily designed yet sub-optimal, and a human operator providing intermittent control of a simulated robotic arm.

1.1 INTRODUCTION

Reinforcement learning (RL) and supervised learning are usually portrayed as distinct methods of learning from experience. RL methods are often applied to problems involving sequential dynamics and optimization of a scalar performance objective, with online exploration of the effects of actions. Supervised learning methods, on the other hand, are frequently used for problems involving static input-output mappings and minimization of a vector error signal, with no explicit dependence on how training examples are gathered. As discussed by Barto and Dietterich (this volume), the key feature distinguishing RL and supervised learning is whether training information from the environment serves as an evaluation signal or as an error signal, and in this chapter, we are interested in problems where both kinds of feedback are available to a learning system *at the same time*.

As an example, consider a young child learning to throw a ball. For this motor task, as well as many others, there is no substitute for ongoing practice. The child repeatedly throws a ball under varying conditions and with variation in the executed motor commands. Bernstein [5] called this kind of trial-and-error learning "repetition without repetition." The outcome of each movement, such as the visual confirmation of whether the ball reached a nearby parent, acts as an evaluation signal that provides the child with feedback about the quality of performance—but with no specific information about what corrections should be made. In addition, the parent may interject error information in the form of verbal instruction or explicit demonstration of what went wrong with each preceding throw. In reality, the feedback may be much more subtle than this. For instance, the final position of the ball reveals some directional information, such as too far to the left or the right; a learned forward model [13] can then be used to make this corrective information more specific to the child's sensorimotor apparatus. Similarly, the tone of the parent's voice may provide evaluative praise simultaneously with the verbal error information. In any case, the two kinds of feedback play interrelated, though complementary roles [2]: The evaluation signal drives skill optimization, whereas the error signal provides a standard of correctness that helps ensure a certain level of performance, either on a trial-by-trial basis or for the learning process as a whole.

The richness of the training information in this example, which we believe is the rule rather than the exception in realistic learning problems, is not effectively used by systems that rely on RL alone. This contributes to some of the difficulties that have been observed when attempting to apply RL to practical problems. Learning can require a very large number trials, and system behavior during learning can lead to unacceptable risks. For this reason, in most large-scale applications RL is applied to simulated rather than real experience. To overcome some of these difficulties, a number of researchers have proposed the use of supervisory information that effectively transforms a learning problem into one which is easier to solve. Common examples involve shaping [10, 18, 20], learning from demonstration [14, 16, 25, 29], or the use of carefully designed controllers [12, 15, 21]. Approaches that explicitly model the role of a supervisor include ASK FOR HELP [7], RATLE [17], and the mentor framework [23]. In each case, the goal of learning is an optimal policy, i.e., a mapping from states to actions that optimizes some performance criterion. Despite the many successful implementations, none of these approaches combines both kinds of feedback as described shortly. Either supervised learning precedes RL during a separate training phase, or else the supervisory information is used to modify a *value function* rather than a policy. Those methods based on Q-learning [32], for instance, build a value function that ranks the actions available in a given state. The corresponding policy is then represented implicitly, usually as the action with the best ranking for each state.

The approach taken in this chapter involves an actor-critic architecture for RL [3]. Actor-critic architectures differ from other value-based methods in that separate data structures are used for the control policy (the "actor") and the value function (the "critic"). One advantage of the actor-critic framework is that action selection requires minimal computation [31]. Methods that lack a separate data structure for the policy typically require a repeated search for the action with the best value, and this search

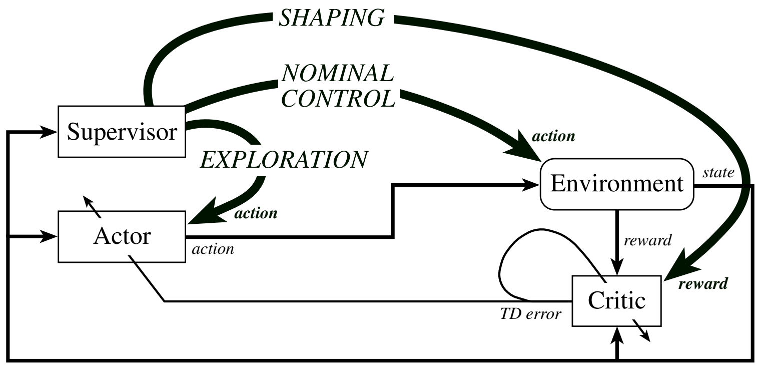

Fig. 1.1 Actor-critic architecture and several pathways for supervisor information.

can become computationally prohibitive, especially for real-valued actions as in the examples of Section 1.3.

Another important advantage of the actor-critic framework is that the policy can be modified directly by standard supervised learning methods. In other words, the actor can change its behavior based on state-action training pairs provided by a supervisor, without the need to calculate the values of those training data. The critic (or some other comparable mechanism) is still required for optimization, whereas the supervisor helps the actor achieve a level of proficiency whenever the critic has a poor estimate of the value function. In the next section we describe a *supervised* actor-critic architecture where the supervisor supplies not only error information for the actor, but also actions for the environment.

1.2 SUPERVISED ACTOR-CRITIC ARCHITECTURE

Figure 1.1 shows a schematic of the usual actor-critic architecture [31] augmented by three major pathways for incorporating supervisor information. Along the “shaping” pathway, the supervisor supplies an additional source of evaluative feedback, or *reward*, that essentially simplifies the task faced by the learning system. For instance, the critic may receive favorable evaluations for behavior which is only approximately correct given the original task. As the actor gains proficiency, the supervisor then gradually withdraws the additional feedback to shape the learned policy toward optimality for the true task. With “nominal control” the supervisor sends control signals (*actions*) directly to the controlled system (the *environment*). For example, the supervisor may override bad commands from the actor as a way to ensure safety and to guarantee a minimum standard of performance. And along the “exploration” pathway, the supervisor provides the actor with hints about which actions may or may not be promising for the current situation, thereby altering the exploratory nature of the actor’s trial-and-error learning. In this section, we focus on the latter

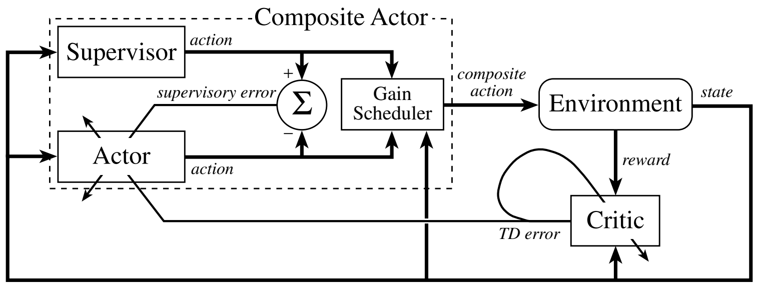

Fig. 1.2 The supervised actor-critic architecture.

two pathways, and we examine the use of supervised learning which offers a powerful counterpart to RL methods.

The combination of supervised learning with actor-critic RL was first suggested by Clouse and Utgoff [8] and independently by Benbrahim and Franklin [4]. Their approach has received almost no attention, yet the more general problem of how to combine a teacher with RL methods has become quite popular. In their work, Benbrahim and Franklin [4] used pre-trained controllers, called "guardians," to provide safety and performance constraints for a biped robot. Joint position commands were sent to the robot by a central pattern generator, but those commands were modified by the guardians whenever a constraint was violated. Superimposed with the joint position commands were exploratory actions generated according to Gullapalli's SRV algorithm [11]. In effect, the central pattern generator learned not only from exploration, but from the guardians as well.

Figure 1.2 shows our version of the supervised actor-critic architecture, which differs from previous work in a key way described in Section 1.2.2. Taken together, the actor, the supervisor and the gain scheduler[1] form a "composite" actor that sends a composite action to the environment. The environment responds to this action with a transition from the current state, s, to the next state, s'. (Appendix 1 contains a list of symbols used throughout the remainder of this chapter.) The environment also provides an evaluation called the immediate reward, r. The job of the critic is to observe states and rewards and to build a value function, $V^{\pi}(s)$, that accounts for both immediate and future rewards received under the composite policy, π. This value function is defined recursively as

$$V^{\pi}(s) = \sum_{s' \in \mathcal{S}} \Pr(s' \mid s,a)\{R(s') + \gamma V^{\pi}(s')\},$$

[1]"Gain scheduling" refers to the construction of a global nonlinear controller by interpolation, or scheduling, of local linear controllers [27]. We use the term in a broader sense to mean the blending of two or more sources of control actions.

where $R(s')$ is the expected value of r, $\gamma \in [0,1]$ is a factor that discounts the value of the next state, and $\Pr(s' \mid s,a)$ is the probability of transitioning to state s' after executing action $a = \pi(\mathrm{s})$. Here we focus on deterministic policies, although this work also generalizes to the stochastic case where π represents a distribution for choosing actions probabilistically. For RL problems, the expected rewards and the state-transition probabilities are typically unknown. Learning, therefore, must proceed from samples, i.e., from observed rewards and state transitions. For RL algorithms, temporal-difference (TD) methods [30] are commonly used to update the state-value *estimates*, $V(s)$, by an amount proportional to the TD error, defined as

$$\delta = r + \gamma V(s') - V(s).$$

1.2.1 The Gain Scheduler

For deterministic policies and real-valued actions, the gain scheduler computes the composite action, a, as simply a weighted sum of the actions given by the component policies. In particular,

$$a \leftarrow ka^E + (1-k)a^S,$$

where a^E is the actor's exploratory action and a^S is the supervisor's action, as given by policies π^E and π^S, respectively. (The supervisor's actions are observable but its policy is unknown.) We also denote by a^A the actor's greedy action determined by the corresponding policy, π^A. Typically, π^E is a copy of π^A modified to include an additive random variable with zero mean. Thus, each exploratory action is simply a noisy copy of the corresponding greedy action, although we allow for the possibility of more sophisticated exploration strategies.

The parameter $k \in [0,1]$ interpolates between π^E and π^S, and therefore k determines the level of control, or autonomy, on the part of the actor.[2] In general, the value of k varies with state, although we drop the explicit dependence on s to simplify notation. The parameter k also plays an important role in modifying the actor's policy, as described in more detail below. We assume that π^A is given by a function approximator with the parameter vector w, and after each state transition, those parameters are updated according to a rule of the form

$$w \leftarrow w + k\Delta w^{\mathrm{RL}} + (1-k)\Delta w^{\mathrm{SL}}, \tag{1.1}$$

where Δw^{RL} and Δw^{SL} are the individual updates based on RL and supervised learning, respectively. Thus, k also interpolates between two styles of learning.

The use of k—a single state-dependent parameter that trades off between two sources of action and learning—allows for a wide range of interactions between actor and supervisor. If the actor has control of the gain scheduler, for instance, then the

[2]For the stochastic case, k gives the probability that the gain scheduler chooses the actor's exploratory action rather than the supervisor's action.

actor can set the value of k near 0 whenever it needs help from its supervisor, cf. ASK FOR HELP [7]. Similarly, if the supervisor has control of the gain scheduler, then the supervisor can set $k = 0$ whenever it loses confidence in the autonomous behavior of the actor, cf. RATLE [17]. The gain scheduler may even be under control of a third party. For example, a linear feedback controller can play the role of supervisor, and then a human operator can adjust the value of k as a way to switch between actor and supervisor, perhaps at a longer time scale than that of the primitive actions.

1.2.2 The Actor Update Equation

To make the reinforcement-based adjustment to the parameters of π^A we compute

$$\Delta w^{\mathrm{RL}} \leftarrow \alpha\delta(a^E - a^A)\nabla_w \pi^A(s), \tag{1.2}$$

where α is a step-size parameter. Eq. (1.2) is similar to the update used by the *REINFORCE* class of algorithms [33], although we utilize the gradient of the deterministic policy π^A rather than that of the stochastic exploration policy π^E. When the TD error is positive, this update will push the greedy policy evaluated at s closer to a^E, i.e., closer to the exploratory action which led to a state with estimated value better than expected. Similarly, when $\delta < 0$, the update will push $\pi^A(s)$ away from a^E and in subsequent visits to state s the corresponding exploratory policy will select this unfavorable action with reduced probability.

To compute the supervised learning update, Δw^{SL}, we seek to minimize in each observed state the supervisory error

$$E = \frac{1}{2}[\pi^S(s) - \pi^A(s)]^2.$$

Locally, this is accomplished by following a steepest descent heuristic, i.e., by making an adjustment proportional to the negative gradient of the error with respect to w:

$$\Delta w^{\mathrm{SL}} \leftarrow -\alpha\nabla_w E(s).$$

Expanding the previous equation with the chain rule and substituting the observed actions gives the usual kind of gradient descent learning rule:

$$\Delta w^{\mathrm{SL}} \leftarrow \alpha(a^S - a^A)\nabla_w \pi^A(s). \tag{1.3}$$

Finally, by substituting Eqs. (1.2) and (1.3) into Eq. (1.1) we obtain the desired actor update equation:

$$w \leftarrow w + \alpha[k\delta(a^E - a^A) + (1-k)(a^S - a^A)]\nabla_w \pi^A(s). \tag{1.4}$$

```
input
    critic value function, $V(s)$, parameterized by $\theta$
    actor policy, $\pi^A(s)$, parameterized by $w$
    exploration size, $\sigma$
    actor step size, $\alpha$, and critic step size, $\beta$
    discount factor, $\gamma \in [0,1]$
    eligibility trace decay factor, $\lambda$
initialize $\theta, w$ arbitrarily
repeat for each trial
    $e \leftarrow 0$ (clear the eligibility traces)
    $s \leftarrow$ initial state of trial
    repeat for each step of trial
        $a^A \leftarrow$ action given by $\pi^A(s)$
        $a^E \leftarrow a^A + N(0,\sigma)$
        $a^S \leftarrow$ action given by supervisor's unknown policy, $\pi^S(s)$
        $k \leftarrow$ interpolation parameter from gain scheduler
        $a \leftarrow ka^E + (1-k)a^S$
        $e \leftarrow \gamma\lambda e + \nabla_\theta V(s)$
        take action $a$, observe reward, $r$, and next state, $s'$
        $\delta \leftarrow r + \gamma V(s') - V(s)$
        $\theta \leftarrow \theta + \beta\delta e$
        $w \leftarrow w + \alpha\left[k\delta(a^E - a^A) + (1-k)(a^S - a^A)\right]\nabla_w \pi^A(s)$
        $s \leftarrow s'$
    until $s$ is terminal
```

Fig. 1.3 The supervised actor-critic learning algorithm for deterministic policies and real-valued actions.

Eq. (1.4) summarizes a steepest descent algorithm where k trades off between two sources of gradient information:[3] one from a performance surface based on the evaluation signal and one from a quadratic error surface based on the supervisory error. Figure 1.3 gives a complete algorithm.

As mentioned above, the architecture shown in Figure 1.2 is similar to one suggested previously by Benbrahim and Franklin [4] and by Clouse and Utgoff [8]. However, our approach is novel in the following way: In the figure, we show a direct connection from the supervisor to the actor, whereas the supervisor in both [4] and [8] influences the actor indirectly through its effects on the environment as well as

[3]In practice an additional parameter may be needed to scale the TD error. This is equivalent to using two step-size parameters, one for each source of gradient information.

the TD error. Using our notation the corresponding update equation for these other approaches, e.g., [4, Eq. (1)], essentially becomes

$$w \leftarrow w + \alpha[k\delta(a^E - a^A) + (1-k)\delta(a^S - a^A)]\nabla_w \pi^A(s) \tag{1.5}$$

$$= w + \alpha\delta[ka^E + (1-k)a^S - a^A]\nabla_w \pi^A(s). \tag{1.6}$$

The key attribute of Eq. (1.5) is that the TD error modulates the supervisory error, $a^S - a^A$. This may be a desirable feature if one "trusts" the critic more than the supervisor, in which case one should view the supervisor as an additional source of exploration. However, Eq. (1.5) may cause the steepest descent algorithm to ascend the associated error surface, especially early in the learning process when the critic has a poor estimate of the true value function. Moreover, when δ is small, the actor loses the ability to learn from its supervisor, whereas in Eq. (1.4) this ability depends primarily on the interpolation parameter, k.

1.3 EXAMPLES

In this section we present several examples that illustrate a gradual shift from full control of the environment by the supervisor to autonomous control by the actor. In each case, the supervisor enables the composite actor in Figure 1.2 to solve the task *on the very first trial*, and on every trial while it improves, whereas the task is virtually impossible to solve with RL alone. The first three examples are targeting tasks—each with a stable controller that brings the system to target, although in a sub-optimal fashion. The final example involves a human supervisor that controls a simulated robot during a peg insertion task.

For each example we used the learning algorithm in Figure 1.3 with step-size parameters of $\alpha = 0.1$ for the actor and $\beta = 0.3$ for the critic. To update the critic's value function, we used the TD(λ) algorithm [30] with $\lambda = 0.7$. We implemented both actor and critic by a tile coding scheme, i.e., CMAC [1], with a total of 25 tilings, or layers, per CMAC. (Appendix 2 provides a brief description of CMAC function approximators.)

1.3.1 Ship Steering Task

For our first experiment we adapted a standard problem from the optimal control literature where the task is to steer a ship to a goal in minimum time [6]. The ship moves at a constant speed of $C = 0.01\,\text{km}\cdot\text{s}^{-1}$, and the real-valued state and action are given, respectively, by the ship's two-dimensional position and scalar heading. For this problem, the supervisor is a hand-crafted controller that always steers directly toward the center of the goal region, i.e., toward the origin $(0, 0)$. Under full supervision, this strategy ensures that the ship will reach the goal eventually—but not in minimum time due to a water current that complicates the task. More

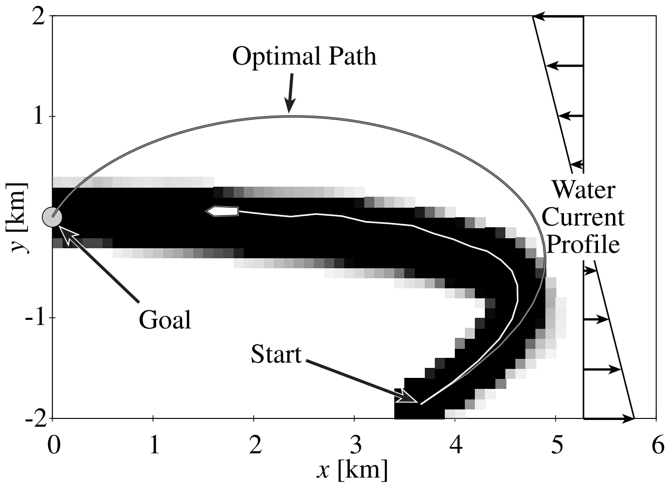

Fig. 1.4 Ship steering task simulator after 1000 learning trials. The grayscale region indicates the level of autonomy, from $k = 0$ (white) to $k = 1$ (black).

specifically, the equations of motion are

$$\dot{x} = C\,(\cos\phi - y),\ \dot{y} = C\sin\phi, \tag{1.7}$$

where ϕ is the ship's heading. Notice that the water current acts along the horizontal direction, x, yet depends solely on the vertical position, y. The start location is $x_0 = 3.66$, $y_0 = -1.86$, and the goal region has a of radius 0.1 km. A convenient feature of this test problem is that one can solve for the optimal policy analytically [6], and the darker curve in Figure 1.4 shows the corresponding optimal path. Under the optimal policy the minimum time to goal is 536.6 s while the supervisor's time to goal is 1111 s.

We integrated Eq. (1.7) numerically using Euler's method with a step size of 1 s. Control decisions by the gain scheduler were made every 25 s, at which time the ship changed heading instantaneously. Exploratory actions, a^E, were Gaussian distributed with a standard deviation of 10 degrees and a mean equal to the greedy action, a^A. The CMAC tiles were uniform with a width of 0.5 km along each input dimension. The actor CMAC was initialized to steer the ship leftward while the critic CMAC was initialized to $V(s) = -1000$, for all s. Rewards were -1 per time step, and the discount factor was $\gamma = 1$, i.e., no discounting.

To make the interaction between supervisor and actor dependent on state, the interpolation parameter, k, was set according to a state-visitation histogram, also implemented as a CMAC with 25 uniform tilings. At the end of each trial, the weights from the "visited" histogram tiles were incremented by a value of 0.0008, for a total increment of 0.02 over the 25 layers. During each step of the simulation, the value of k was set to the CMAC output for the current state, with values cut off at a maximum of $k = 1$, i.e., at full autonomy. Thus, the gain scheduler made a gradual shift from full supervision to full autonomy as the actor and critic acquired enough control knowledge to reach the goal reliably. A decay factor of 0.999 was also used to downgrade the weight of each CMAC tile; in effect, autonomy "leaked away" from infrequently visited regions of state space.

Figure 1.5 shows the effects of learning for each of two cases. One case corresponds to the supervised actor-critic algorithm in Figure 1.3, with the parameter values described above. The other case is from a two-phase learning process where 1000 trials were used to seed the actor's policy as well as the critic's value function, followed by RL alone, cf. [14, 16, 25, 29]. That is, $k = 0$ for the first 1000 trials, and $k = 1$ thereafter. Both cases show rapid improvement during the first 100 trials of RL, followed by slower convergence toward optimality. In panel (a) the two-phase process appears to give much improved performance—if one is willing to pay the price associated with an initial learning phase that gives no immediate improvement. Indeed, if we examine cumulative reward instead, as in panel (b), the roles become reversed with the two-phase process performing worse. Performance improves somewhat if we reduce the number of seed trials, although with fewer than 500 seed trials of supervised learning, the actor is able to reach the goal either unreliably (300 and 400 trials) or else not at all (100 and 200 trials).

1.3.2 Manipulator Control

Our second example demonstrates that the style of control and learning used for the ship steering task is also suitable for learning to exploit the dynamics of a simulated robotic arm. The arm was modeled as a two-link pendulum with each link having length 0.5 m and mass 2.5 kg, and the equations of motion [9] were integrated numerically using Euler's method with a step size of 0.001 s. Actions from both actor and supervisor were generated every 0.75 s and were represented as two-dimensional velocity vectors with joint speed limits of ± 0.5 rad/sec. The task was to move with minimum effort from the initial configuration with joint angles of -90 and 0 degrees to the goal configuration with joint angles of 135 and 90 degrees. For this demonstration, effort was quantified as the total integrated torque magnitude.

Similar to the ship steering task, the supervisor in this example is a hand-crafted controller that moves the arm at maximum speed directly toward the goal in configuration space. Therefore, actions from the supervisor always lie on a unit square centered at the origin, whereas the actor is free to choose from the entire set of admissible actions. In effect, the supervisor's policy is to follow a straight-line path to the goal—which is time-optimal given the velocity constraints. Due to the dynamics of the robot, however, straight-line paths are not necessarily optimal with respect to other performance objectives, such as minimum energy.

A lower-level control system was responsible for transforming commanded velocities into motor torques for each joint. This occurred with a control interval of 0.001 s and in several stages: First, the commanded velocity was adjusted to account for acceleration constraints that eliminate abrupt changes in velocity, especially at the beginning and end of movement. The adjusted velocity, along with the current

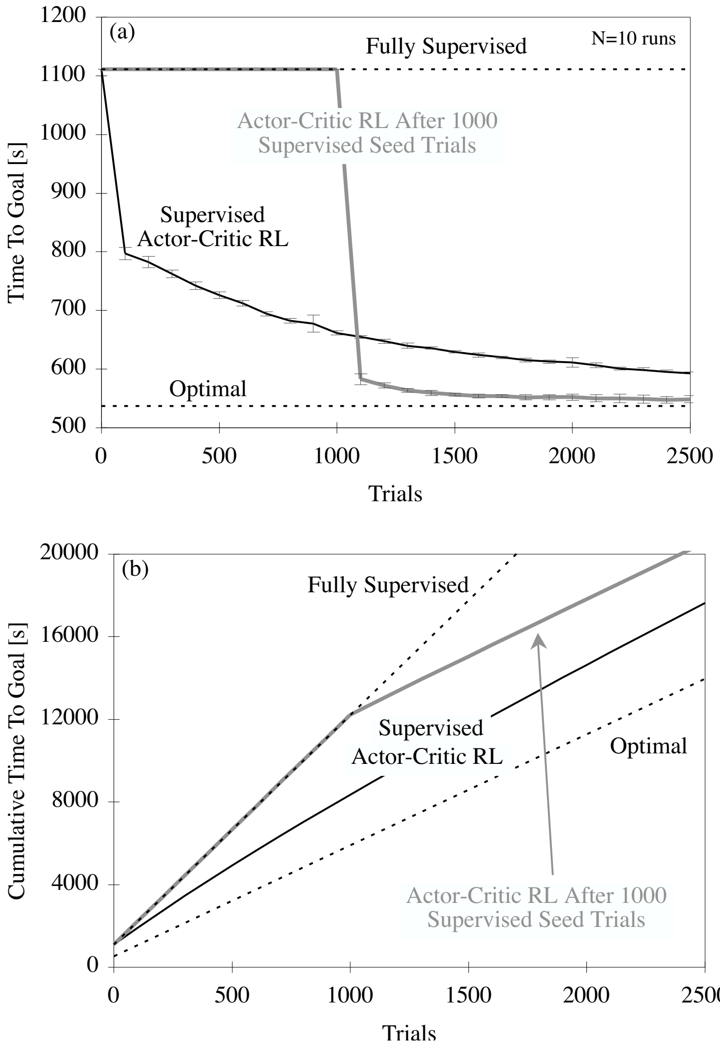

Fig. 1.5 Ship steering task effects of learning averaged over 10 runs of 2500 trials each: (a) time to goal, and (b) cumulative time to goal. For the supervised learning seed trials the initial position of the ship was chosen randomly from the region $0 \leq x \leq 6$, $-2 \leq y \leq 2$.

position, was then used to compute the desired position at the end of the next control interval. Third, a proportional-derivative controller converted this target position into joint torques, but with a target velocity of zero rather than the commanded velocity. And finally, a simplified model of the arm was used to adjust the feedback-based torque to include a feedforward term that compensates for gravity. This scheme is intended to match the way some industrial manipulators are controlled once given a higher-level movement command, e.g., velocity as used here. Gravity compensation guarantees stability of the lower-level controller [9], and the target velocity of zero helps ensure that the arm will stop safely given a communications failure with the higher level.

The above control scheme also holds an advantage for learning. Essentially, the manipulator behaves in accordance with a tracking controller—only the desired trajectory is revealed gradually with each control decision from the higher level. At this level, the manipulator behaves like an overdamped, approximately first-order system, and so policies need not account for the full state of the robot. That is, for both actor and supervisor it suffices to use reduced policies that map from positions to velocity commands, rather than policies that map from positions *and* velocities to acceleration commands. As is common with tracking controllers, this abstraction appears to cancel the dynamics we intend to exploit. However, by designing an optimal control problem, we allow the dynamics to influence the learning system by way of the performance objective, i.e., through the reward function.

For the RL version of this optimal control problem, rewards were the negative effort accumulated over each 0.75 s decision interval. As with the ship steering task, the discount parameter was set to $\gamma = 1$ and the exploratory actions, a^E, were Gaussian distributed with a mean equal to the greedy action (although a^E was clipped at the joint speed limits). The standard deviation of the exploratory actions was initially 1.0 rad/sec, but this value decayed exponentially toward zero by a factor of 0.999 after each trial. CMAC tiles were uniform with a width of 25 degrees along each input dimension; the actor CMAC was initialized to zero whereas the critic CMAC was initialized to -300. Like the previous example, a third CMAC was used to implement a state-visitation histogram that stored the value of the interpolation parameter, k. As above, the histogram increment was 0.02 over the 25 layers and the decay factor was 0.999.

Figure 1.6(a) shows the configuration of the robot every 0.75 s along a straight-line path to the goal. The proximal joint has more distance to cover and therefore moves at maximum speed, while the distal joint moves at a proportionately slower speed. The total effort for this fully supervised policy is 258 Nm·s. Figures 1.6(b) and 1.6(c) show examples of improved performance after 5000 trials of learning, with a final cost of 229 and 228 Nm·s, respectively. In each of the left-hand diagrams, the corresponding "spokes" from the proximal link fall in roughly the same position, and so the observed improvements are due to the way the distal joint modulates its movement around the straight-line path, as shown in the right-hand diagrams.

Figure 1.7 shows the effects of learning averaged over 25 runs. The value of the optimal policy for this task is unknown, although the best observed solution has a cost of 216 Nm·s. Most improvement happens within 400 trials and the remainder of learning shows a drop in variability as the exploration policy "decays" toward the greedy policy. One difficulty with this example is the existence of many locally optimal solutions to the task. This causes the learning system to wander among solutions, with convergence to one of them only when forced to do so by the reduced exploration.

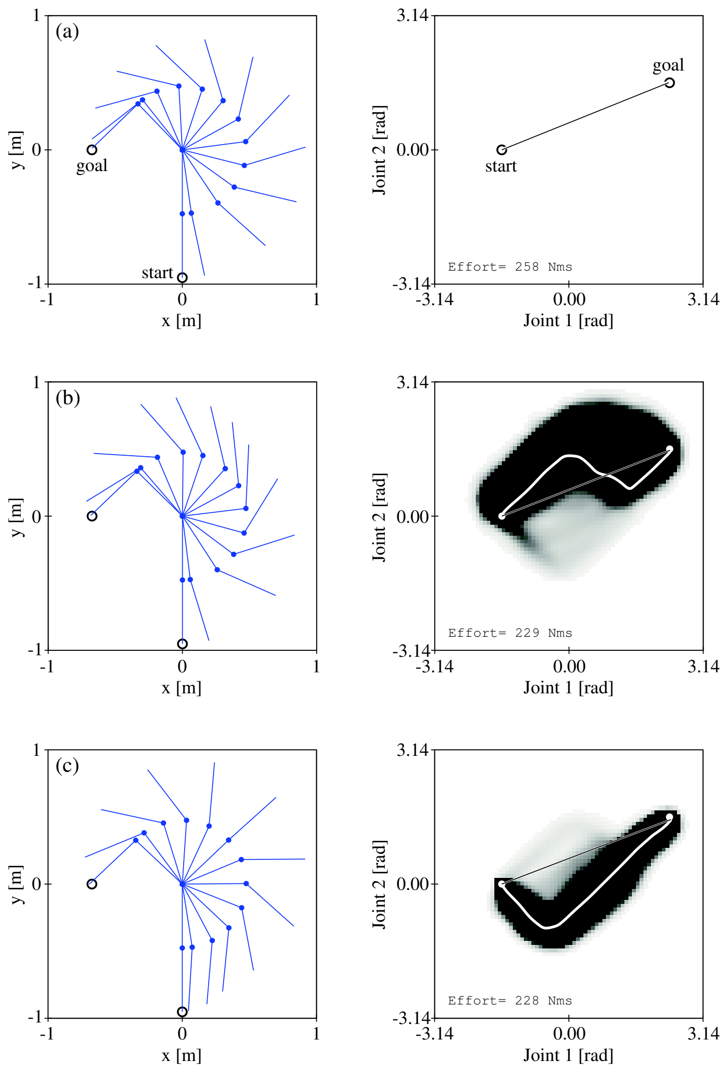

Fig. 1.6 Simulated two-link arm after (a) no learning and (b,c) 5000 learning trials. Configuration-space paths after learning are shown in white, and the grayscale region indicates the level of autonomy, from $k = 0$ (white) to $k = 1$ (black).

1.3.3 Case Study With a Real Robot

To demonstrate that the methods in this chapter are suitable for real robots, we replicated the previous example with a seven degree-of-freedom whole arm manipulator (WAM; Barrett Technology Inc., Cambridge, MA). Figure 1.8 shows a sequence of several postures as the WAM moves from the start configuration (far left frame) to

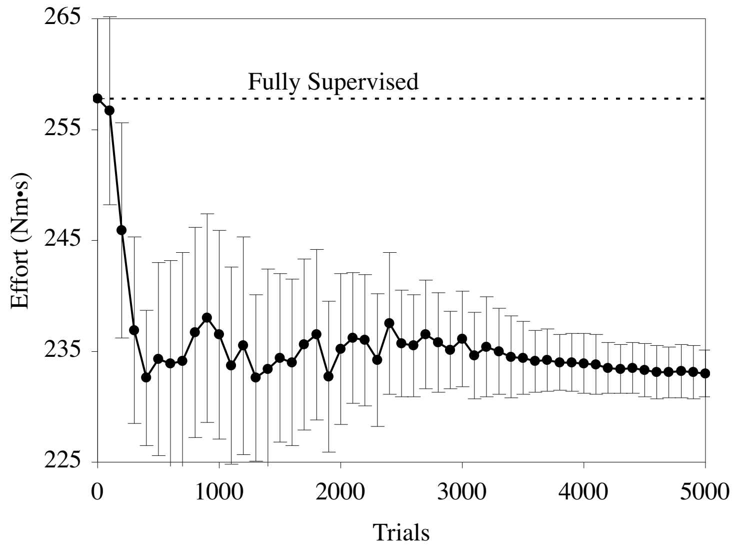

Fig. 1.7 Effects of learning for the simulated two-link arm averaged over 25 runs of 5000 trials each.

the goal configuration (far right frame). As with the previous example the task was formulated as a minimum-effort optimal control problem—utilizing a stable tracking controller and a supervisor that generates straight-line trajectories to the goal in configuration space. The joint speed limits for this example were increased to ± 0.75 rad/sec rather than ± 0.5 rad/sec as used above. The learning algorithm was virtually identical to the one in the previous example, although several parameter values were modified to encourage reasonable improvement with very few learning trials. For instance, the histogram increment was increased from 0.02 to 0.10, thereby facilitating a faster transition to autonomous behavior. Also, the level of exploration did not decay, but rather remained constant, and a^E was Gaussian distributed with a standard deviation of 0.25 rad/sec.

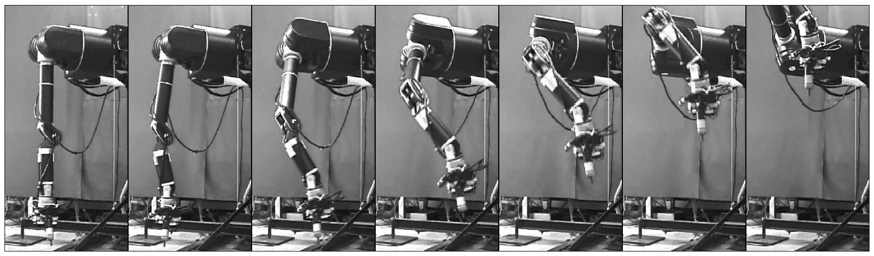

Fig. 1.8 Representative configurations of the WAM after learning.

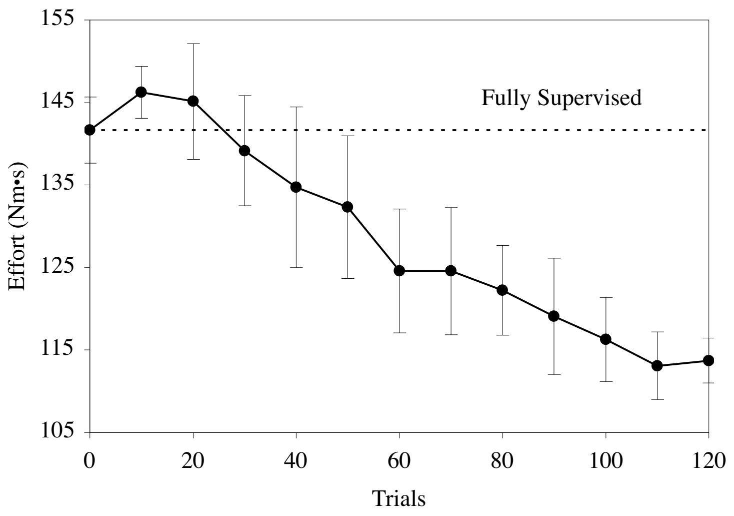

Fig. 1.9 Effects of learning for the WAM averaged over 5 runs of 120 trials each.

Figure 1.9 shows the effects of learning averaged over 5 runs. Performance worsens during the first 10 to 20 trials due to the initialization of the actor's policy. More specifically, at the start of learning the actor's policy maps all inputs to the zero velocity vector, and so the actor cannot move the robot until it has learned how to do so from its supervisor. The drawback of this initialization scheme—along with a fast transition to autonomous behavior—is that early in the learning process the supervisor's commands become diminished when blended with the actor's near-zero commands. The effect is slower movement of the manipulator and prolonged effort while raising the arm against gravity. However, after 60 trials of learning the supervised actor-critic architecture shows statistically significant improvement ($p < 0.01$) over the supervisor alone. After 120 trials, the overall effect of learning is approximately 20% reduced effort despite an increased average movement time from 4.16 s to 4.34 s (statistically significant with $p < 0.05$).

1.3.4 Peg Insertion Task

One goal of our ongoing work is to make the techniques described in this chapter applicable for telerobotics, i.e., for remote operation of a robot by a human supervisor, possibly over great distances. Figure 1.10 shows the setup for our current tool-use experiment. For such applications, the human operator is often responsible for setting immediate goals, for managing sub-tasks, for making coarse-grained motor decisions (e.g., grasp a tool with the palm up rather than down) and also for executing fine-grained control of the robot. Moreover, these many responsibilities on the part of the human are just one source of the operator fatigue which hampers the effectiveness of virtually all telerobotic systems. The operator deals not only with a hierarchy of

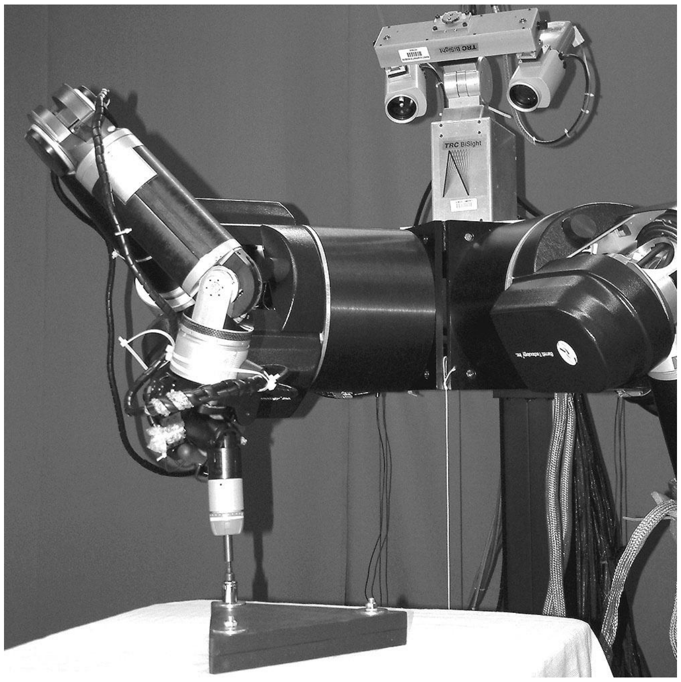

Fig. 1.10 Humanoid robot using a tool to perform a variation of a canonical parts insertion task.

control objectives but also with limited, and sometimes confusing feedback across the interface between human and machine. The potential contribution from machine learning is a way to push the human's involvement further up the hierarchy as the machine gains competence at each level of control. In particular, a supervised actor-critic architecture allows the human supervisor to remain "in the loop" as the actor learns about the details of the robot's task, especially those details which are difficult to convey across the user interface, e.g., tactile feedback.

As a preliminary example, Figure 1.11 shows several snapshots during a simulated peg insertion task. With no initial control knowledge about the task, the actor is completely dependent upon input from its human supervisor (via a mouse). After just 10-20 trials, the actor has gathered sufficient information with which to propose actions. Short bars in the figure depict the effects of such actions, as projected forward in time by prediction through a kinematic model. In Figure 1.11(a), for instance, the bars indicate to the operator that the actor will move the gripper toward the (incorrect) middle slot. In this scenario, the target slot is hidden state for the actor, and so brief input from the operator (panel b) is needed to push the actor into the basin of attraction for the upper target. Full autonomy by the actor is undesirable for this task. The human supervisor remains in control of the robot, while short intervals of autonomous behavior alleviate much of the operator's fatigue.

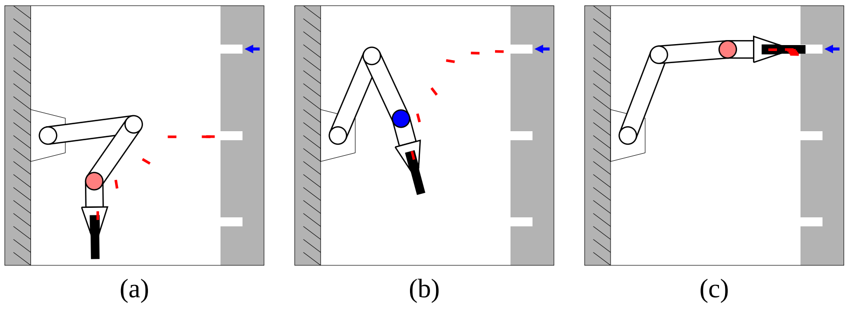

Fig. 1.11 Human-robot interface for a simulated peg insertion task after 20 learning trials. The arrow marks the target slot, and small bars indicate predicted gripper positions under autonomous control by the actor. (a) After successful re-grasp of the peg, the actor begins movement toward the middle slot. (b) A momentary correction by the human supervisor places the robot on track for the upper target, after which (c) the actor completes the sub-task autonomously.

1.4 CONCLUSIONS

The examples in Section 1.3 demonstrate a gradual shift from full supervision to full autonomy—blending two sources of actions and learning feedback. Much like the examples by Clouse [7] and by Maclin and Shavlik [17], this shift happens in a state-dependent way with the actor seeking help from the supervisor in unfamiliar territory. Unlike these other approaches, the actor also clones the supervisor's policy very quickly over the visited states. This style of learning is similar to methods that seed an RL system with training data, e.g., [25, 29], although with the supervised actor-critic architecture, the interpolation parameter allows the seeding to happen in an incremental fashion at the same time as trial-and-error learning. Informally, the effect is that the actor knows what the supervisor knows, but only on a need-to-know basis.

One drawback of these methods for control of real robots is the time needed for training. By most standards in the RL literature, the supervised actor-critic architecture requires relatively few trials, at least for the examples presented above. However, some robot control problems may permit extremely few learning trials, say 10 or 20. Clearly, in such cases we should not expect optimality; instead we should strive for methods that provide gains commensurate with the training time. In any case, we might tolerate slow optimization if we can deploy a learning robot with provable guarantees on the worst-case performance. Recent work by Kretchmar *et al.* [15] and by Perkins and Barto [22] demonstrates initial progress in this regard.

With regard to telerobotic applications, our results thus far are promising, although several key challenges remain. First, our simulated peg insertion task is somewhat simplified—in terms of the noise-free sensors and actuators, the user interface, the surface contact model, etc.—and so our present efforts are focused on a more convincing demonstration with the humanoid robot shown in Figure 1.10. Another difficulty is that input from the supervisor can quickly undo any progress made by the RL component. Consequently, we are also exploring principled ways to weaken the effects of the supervised learning aspect without necessarily weakening the human operator's control over the robot. A third challenge is related to the way a human-robot interface introduces constraints during the learning process. For example, the interface may restrict the supervisor's control of the robot to various subsets of its degrees of freedom. In turn, this biases the way training data are gathered, such that the actor has difficulty learning to coordinate all degrees of freedom simultaneously.

Despite the challenges when we combine supervised learning with an actor-critic architecture, we still reap benefits from both paradigms. From actor-critic architectures we gain the ability to discover behavior that optimizes performance. From supervised learning we gain a flexible way to incorporate domain knowledge. In particular, the internal representations used by the actor can be very different from those used by the supervisor. The actor, for example, can be an artificial neural network, while the supervisor can be a conventional feedback controller, expert knowledge encoded as logical propositions, or a human supplying actions that depend on an entirely different perception of the environment's state. Moreover, the supervisor can convey intentions and solution strategies to the actor, and so this work is similar in spirit to work on imitation learning, e.g., [19, 26]. And presumably the supervisor has a certain proficiency at a given task, which the actor exploits for improved performance throughout learning.

Acknowledgments

We thank Andrew Fagg, Mohammad Ghavamzadeh, and Balaraman Ravindran for helpful discussions. This work was supported by NASA under award No. NAG-9-1379. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Aeronautics and Space Administration.

Bibliography

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

2. A. G. Barto. Reinforcement learning in motor control. In M. A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks, Second Edition*, pages 968–972. The MIT Press, Cambridge, MA, 2003.

3. A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 13:835–846, 1983.

4. H. Benbrahim and J. A. Franklin. Biped dynamic walking using reinforcement learning. *Robotics and Autonomous Systems*, 22:283–302, 1997.

5. N. A. Bernstein. *The Co-ordination and Regulation of Movements*. Pergamon Press, Oxford, 1967.

6. A. E. Bryson and Y.-C. Ho. *Applied Optimal Control*. Hemisphere Publishing Corp., New York, 1975.

7. J. A. Clouse. *On Integrating Apprentice Learning and Reinforcement Learning*. PhD thesis, University of Massachusetts, Amherst, 1996.

8. J. A. Clouse and P. E. Utgoff. A teaching method for reinforcement learning. In *Proceedings of the Nineth International Conference on Machine Learning*, pages 92–101, 1992. Morgan Kaufmann, San Francisco, CA.

9. J. J. Craig. *Introduction To Robotics : Mechanics and Control*. Addison-Wesley Publishing Company, Reading, MA, 1989.

10. M. Dorigo and M. Colombetti. Robot shaping: developing autonomous agents through learning. *Artificial Intelligence*, 71(2):321–370, 1994.

11. V. Gullapalli. A stochastic reinforcement learning algorithm for learning real-valued functions. *Neural Networks*, 3(6):671–692, 1990.

12. M. Huber and R. A. Grupen. A feedback control structure for on-line learning tasks. *Robotics and Autonomous Systems*, 22(3–4):303–315, 1997.

13. M. I. Jordan and D. E. Rumelhart. Forward models: Supervised learning with a distal teacher. *Cognitive Science*, 16(3):307–354, 1992.

14. M. Kaiser and R. Dillmann. Building elementary robot skills from human demonstration. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2700–2705, 1996. IEEE, Piscataway, NJ.

15. R. M. Kretchmar, P. M. Young, C. W. Anderson, D. C. Hittle, M. L. Anderson, C. C. Delnero, and J. Tu. Robust reinforcement learning control with static and dynamic stability. *International Journal of Robust and Nonlinear Control*, 11: 1469–1500, 2001.

16. L.-J. Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8(3–4):293–321, 1992.

17. R. Maclin and J. W. Shavlik. Creating advice-taking reinforcement learners. *Machine Learning*, 22((1-3)):251–281, 1996.

18. M. J. Mataric. Reward functions for accelerated learning. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 181–189, 1994. Morgan Kaufmann, San Francisco, CA.

19. M. J. Mataric. Sensory-motor primitives as a basis for imitation: linking perception to action and biology to robotics. In C. Nehaniv and K. Dautenhahn, editors, *Imitation in Animals and Artifacts*. The MIT Press, Cambridge, MA, 2000.

20. A. Y. Ng, D. Harada, and S. Russell. Policy invariance under reward transformations: Theory and applications to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning*, pages 278–287, 1999. Morgan Kaufmann, San Francisco, CA.

21. T. J. Perkins and A. G. Barto. Lyapunov-constrained action sets for reinforcement learning. In C. Brodley and A. Danyluk, editors, *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 409–416, 2001. Morgan Kaufmann, San Francisco, CA.

22. T. J. Perkins and A. G. Barto. Lyapunov design for safe reinforcement learning. *Journal of Machine Learning Research*, 3:803–832, 2002.

23. B. Price and C. Boutilier. Implicit imitation in multiagent reinforcement learning. In I. Bratko and S. Dzeroski, editors, *Proceedings of the Sixteenth International Conference on Machine Learning*, pages 325–334, 1999. Morgan Kaufmann, San Francisco, CA.

24. J. C. Santamaria, R. S. Sutton, and A. Ram. Experiments with reinforcement learning in problems with continuous state and action spaces. *Adaptive Behavior*, 6:163–217, 1997.

25. S. Schaal. Learning from demonstration. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances In Neural Information Processing Systems 9*, pages 1040–1046, 1997. The MIT Press, Cambridge, MA.

26. S. Schaal. Is imitation learning the route to humanoid robots? *Trends in Cognitive Science*, 3:233–242, 1999.

27. J. S. Shamma. Linearization and gain-scheduling. In W. S. Levine, editor, *The Control Handbook*, pages 388–396. CRC Press, Boca Raton, FL, 1996.

28. S. P. Singh and R. S. Sutton. Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22(1–3):123–158, 1996.

29. W. D. Smart and L. P. Kaelbling. Effective reinforcement learning for mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3404–3410, 2002. IEEE, Piscataway, NJ.

30. R. S. Sutton. Learning to predict by the method of temporal differences. *Machine Learning*, 3:9–44, 1988.

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

32. C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3/4): 279–292, 1992.

33. R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

Appendix 1: Nomenclature

a	composite action	π^E	exploratory policy
a^A	actor action	π^S	supervisor policy
a^E	exploratory action	r	immediate reward
a^S	supervisor action	R	expected immediate reward
α	learning step size (actor)	s	current state
β	learning step size (critic)	s'	next state
δ	TD error	σ	exploration size
E	supervisory error	θ	CMAC weight vector (critic)
e	eligibility trace	V^π	value function under policy π
γ	discount factor	V	estimate of V^π
k	interpolation parameter	w	CMAC weight vector (actor)
λ	eligibility trace decay factor	Δw^{RL}	reinforcement learning update
π	composite policy	Δw^{SL}	supervised learning update
π^A	actor policy		

Appendix 2: Cerebellar Model Arithmetic Computer

A CMAC, or *cerebellar model arithmetic computer* [1], is a kind of artificial neural network inspired by the anatomy and physiology of the cerebellum. Much like radial basis function (RBF) networks, CMACs have processing units that are localized in different regions of an input space. In contrast with RBF networks—where each unit computes a scalar value based on how close an input is to the center of the unit—CMAC units instead compute a binary value for each input. In either case, the computed values affect how much the associated weight parameters contribute to the network's output. With CMAC units, their binary nature therefore determines whether a particular weight "participates" entirely or not at all in the output calculation.

Also in contrast to RBF networks—with radially symmetric processing units—CMAC units are hyper-rectangles arranged as a *tiling*, i.e., as a grid-like tessellation of the input space. This leads to computationally efficient implementations similar to lookup tables. Another advantage of CMACs is that multiple tilings, with each one offset from the rest, can be used to improve resolution, while relatively large hyper-rectangles can be used to improve generalization. And for modern RL algorithms, a function approximator with binary units allows one to take advantage of replacing eligibility traces [28], as we do with the algorithm in Figure 1.3. See [24] for more information about CMACs, including an empirical evaluation of their use for RL problems.