Supervised Learning Combined with an Actor-Critic Architecture

Michael T. Rosenstein
Andrew G. Barto

CMPSCI Technical Report 02-41

October 18, 2002

Department of Computer Science
University of Massachusetts
140 Governors Drive
Amherst, Massachusetts 01003

Abstract

To address the shortcomings of reinforcement learning (RL) a number of researchers have focused recently on ways to take advantage of structure in RL problems and on ways to make domain knowledge part of RL algorithms. In this paper we examine a *supervised* actor-critic architecture, whereby a supervisor adds structure to a learning problem and supervised learning makes that structure part of an actor-critic framework for reinforcement learning. We provide a steepest descent algorithm for real-valued actions such that the actor adjusts its policy in accordance with gradient information from both supervisor and critic. We also illustrate the approach with 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.

Supervised Learning Combined with an Actor-Critic Architecture

Michael T. Rosenstein Andrew G. Barto

CMPSCI Technical Report 02-41
Department of Computer Science
University of Massachusetts, Amherst
{mtr,barto}@cs.umass.edu

Abstract

To address the shortcomings of reinforcement learning (RL) a number of researchers have focused recently on ways to take advantage of structure in RL problems and on ways to make domain knowledge part of RL algorithms. In this paper we examine a *supervised* actor-critic architecture, whereby a supervisor adds structure to a learning problem and supervised learning makes that structure part of an actor-critic framework for reinforcement learning. We provide a steepest descent algorithm for real-valued actions such that the actor adjusts its policy in accordance with gradient information from both supervisor and critic. We also illustrate the approach with 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 Introduction

Reinforcement learning and supervised learning are often portrayed as distinct methods of learning from training examples. Reinforcement learning methods emphasize sequential dynamics and optimization of a scalar performance objective, with online exploration of the effects of actions. Supervised learning methods, on the other hand, emphasize static input-output mappings and minimization of a vector error signal, with no explicit dependence on how training examples are gathered. As discussed by Jordan and Rumelhart [12] reinforcement learning and supervised learning differ not so much in the *algorithms* employed but rather in the nature of the *problems* solved. The distinguishing feature is whether feedback from the environment serves as an evaluation signal or as an error signal, and in this paper, 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 [12] 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.

To overcome some of the difficulties with reinforcement learning alone, 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 [9, 16, 18], learning from demonstration [13, 14, 21, 25], or the use of carefully designed controllers [11, 19]. Approaches that explicitly model the role of a supervisor include ASK FOR HELP [7], RATLE [15], and the mentor framework [20]. 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 reinforcement learning 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 [29], 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 paper involves an actor-critic architecture for reinforcement learning [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 [27]. 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 can become computationally prohibitive, especially for real-valued actions as with the examples in Section 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 is still required for optimization, e.g., [28], 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.

2 Supervised Actor-Critic Architecture

The combination of supervised learning with actor-critic reinforcement learning was first suggested by Clouse and Utgoff [8] and independently by Benbrahim and Franklin [4]. The 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

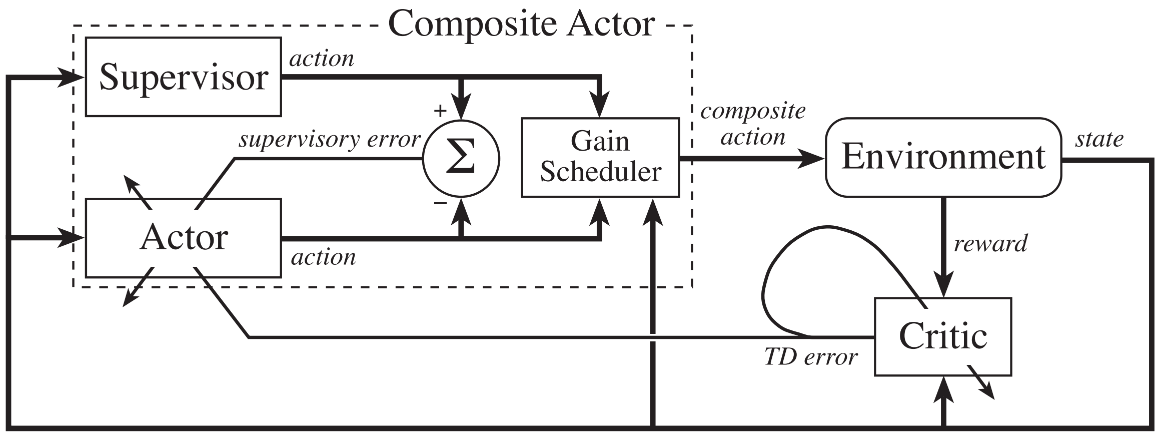

Figure 1: The supervised actor-critic architecture.

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 [10]. In effect, the central pattern generator learned not only from exploration, but from the guardians as well.

Figure 1 shows the supervised actor-critic architecture considered in this paper. 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'. The environment also provides an evaluation of s' called the immediate reward, r. The job of the critic is to observe states and rewards and to build a value function, $V(s)$, that accounts for both immediate and future rewards received under the composite policy. Temporal-difference (TD) methods [26] are commonly used to update $V(s)$ by an amount proportional to the TD error, δ, where

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

and $\gamma \in [0, 1]$ is a discount factor.

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,

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

where a^A and a^S are the actions from the actor and supervisor, respectively, a^E is an exploratory action, and $k \in [0, 1]$ is an interpolation parameter that 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.

[1]"Gain scheduling" refers to the construction of a global nonlinear controller by interpolation, or scheduling, of local linear controllers [23]. We use the term in a broader sense to mean the blending of two or more sources of control actions.

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

```
input
    critic value function, $V(s)$, parameterized by $\theta$
    actor policy, $\pi^A(s)$, parameterized by $w$
    exploration size, $\sigma$
    actor step size, $\alpha$
    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 $\pi^A(s)$
        $a^S \leftarrow$ action given by supervisor's unknown policy, $\pi^S(s)$
        $a^E \leftarrow N(0,\sigma)$
        $k \leftarrow$ interpolation parameter from gain scheduler
        $a \leftarrow k(a^A + a^E) + (1-k)a^S$
        $e \leftarrow \gamma\lambda e + \nabla_\theta V(s)$
        take action $a$, observe reward, $r$, and next state, $s'$
        $\delta \leftarrow r + \gamma V(s') - V(s)$
        $\theta \leftarrow \theta + \beta\delta e$
        $w \leftarrow w + \alpha\left[k\delta a^E + (1-k)(a^S - a^A)\right]\nabla_w \pi^A(s)$
        $s \leftarrow s'$
    until $s$ is terminal
```

Figure 2: The supervised actor-critic learning algorithm for deterministic policies and real-valued actions.

The parameter k also plays an important role for modifying the actor's policy, $\pi^A(s)$, which maps states to actions, a^A. We assume that π^A is given by a function approximator with the parameter vector w; for instance, w could be the weights of an artificial neural network. After each state transition, the parameters of π^A are updated according to the rule

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

where α is a step-size parameter. Eq. (1) 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, e.g., [10], and one from a quadratic error surface based on the supervisory error, $a^S - a^A$. Figure 2 gives a complete algorithm.

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 actor can set the value of k near 0 whenever it needs help from its supervisor, cf. [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. [15]. 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.

As mentioned above, the architecture shown in Figure 1 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 ref. [4] and ref. [8] influences the actor indirectly through its effects on the

[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.

environment as well as the TD error. Using our notation, the corresponding update equation [4, Eq. (1)] essentially becomes

$$w \leftarrow w + \alpha[k\delta a^E + (1-k)\delta(a^S - a^A)]\nabla_w \pi^A(s). \tag{2}$$

The key attribute of Eq. (2) 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. (2) 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) this ability depends primarily on the interpolation parameter, k.

3 Examples

In this section we present two examples that illustrate a gradual shift from full control of the environment by the supervisor to autonomous control by the actor. In both cases, the supervisor enables the composite actor in Figure 1 to solve the task *on the very first trial* whereas the task is virtually impossible to solve with RL alone. One task is a targeting problem with a heuristic controller that brings the system to target, although in a sub-optimal fashion. The second example involves a human supervisor that controls a simulated robot during a peg insertion task.

For both examples we used the learning algorithm in Figure 2 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 [26] 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. One advantage of tile coding schemes is that each tile acts as a binary feature, and so we easily modified the algorithm in Figure 2 to take advantage of replacing eligibility traces [24].

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 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 specifically, the equations of motion are

$$\dot{x} = C\,(\cos\phi - y),\ \dot{y} = C\,\sin\phi, \tag{3}$$

where ϕ is the ship's heading, and the water current acts along the horizontal direction, x, yet depends on 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 centered at the origin. A convenient feature of this test problem is that one can

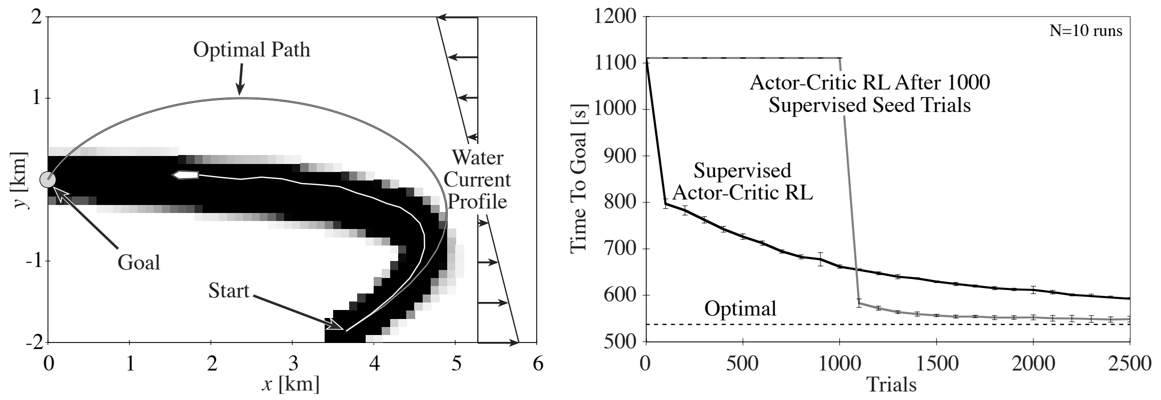

Figure 3: Ship steering task. The left panel shows the simulator after 1000 learning trials. The grayscale region indicates the level of autonomy, from $k = 0$ (white) to $k = 1$ (black). The right panel shows the effects of learning for 10 runs. For the supervised learning seed trials the initial position of the ship was chosen randomly from the region $0 \leq x \leq 6$, $-2 \leq y \leq 2$.

solve for the optimal policy analytically [6], and the dark curve in the left panel of Figure 3 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. (3) 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. 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 by 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. 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.

The right panel of Figure 3 shows the effects of learning for each of two cases. One case corresponds to the supervised actor-critic algorithm in Figure 2, 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 reinforcement learning alone, cf. [13, 14, 21, 25]. That is, $k = 0$ for the first 1000 trials, and $k = 1$ thereafter. Both cases show rapid improvement during the first 100 trials of reinforcement learning, followed by slower convergence toward optimality. The two-phase process appears to give much improved performance—if

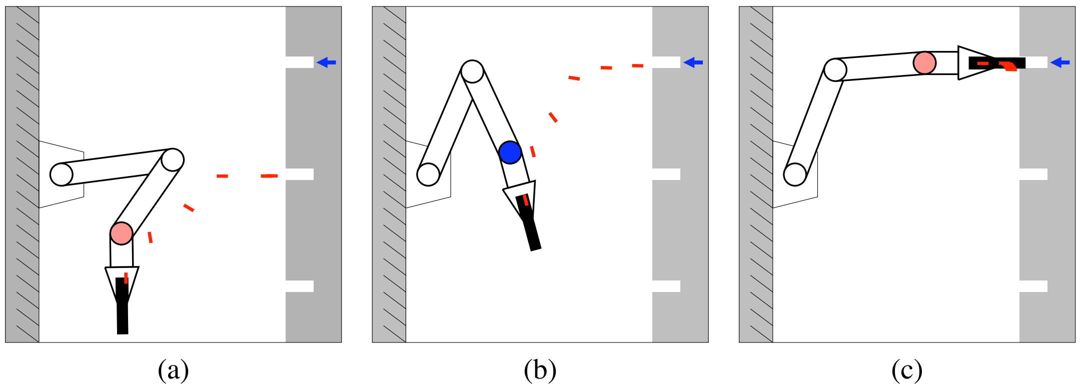
Figure 4: Human-robot interface for a simulated peg insertion task after 20 learning trials. The arrow marks the target slot, and small bars indicate predicted gripper positions under autonomous control by the actor. (a) After successful re-grasp of the peg, the actor begins movement toward the middle 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.

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, the roles become reversed with the two-phase process performing worse.

3.2 Peg Insertion Task

One goal of our ongoing work is to make the techniques described in this paper applicable for telerobotics, i.e., for remote operation of a robot by a human supervisor, possibly over great distances. 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 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 4 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 5-10 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 4(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.

4 Conclusions

Our results thus far suggest that the supervised actor-critic architecture is well-suited for telerobotic applications, 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 a real robot manipulator. Another difficulty is that input from the supervisor can quickly undo any progress made by the reinforcement learning 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., [17, 22]. 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, Roderic Grupen, 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.

References

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

[2] A. G. Barto. Reinforcement learning in motor ontrol. In M. A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks*, pages 809–813. The MIT Press, Cambridge, MA, 1995.

[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. Revised printing by Taylor & Francis Publishers (Bristol, PA) in 1981.

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

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

[11] 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.

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

[13] 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.

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

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

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

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

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

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

[25] 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.

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

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

[28] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In S. A. Solla, T. K. Leen, and K.-R. Muller, editors, *Advances In Neural Information Processing Systems 12*, pages 1057–1063, 2000. The MIT Press, Cambridge, MA.

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