Chapter 2

Reinforcement Learning

Andrew G. Barto

ABSTRACT Reinforcement learning refers to ways of improving performance through trial-and-error experience. Despite recent progress in developing artificial learning systems, including new learning methods for artificial neural networks, most of these systems learn under the tutelage of a knowledgeable "teacher" able to tell them how to respond to a set of training stimuli. But systems restricted to learning under these conditions are not adequate when it is costly, or even impossible, to obtain the required training examples. Reinforcement learning allows autonomous systems to learn from their experiences instead of exclusively from knowledgeable teachers. Although its roots are in experimental psychology, this chapter provides an overview of modern reinforcement learning research directed toward developing capable artificial learning systems.

1 Introduction

The term *reinforcement* comes from studies of animal learning in experimental psychology, where it refers to the occurrence of an event, in the proper relation to a response, that tends to increase the probability that the response will occur again in the same situation [Kim61]. Although the specific term "reinforcement learning" is not used by psychologists, it has been widely adopted by theorists in engineering and artificial intelligence to refer to a class of learning tasks and algorithms based on this principle of reinforcement. Mendel and McLaren, for example, used the term "reinforcement learning control" in their 1970 paper describing how this principle can be applied to control problems [MM70]. The simplest reinforcement learning methods are based on the commonsense idea that if an action is followed by a satisfactory state of affairs or an improvement in the state of affairs, then the tendency to produce that action is strengthened, i.e., reinforced. This basic idea follows Thorndike's [Tho11] classic 1911 "Law of Effect":

> Of several responses made to the same situation, those which are accompanied or closely followed by satisfaction to the animal will, other things being equal, be more firmly connected with the situation, so that, when it recurs, they will be more likely to recur; those which are accompanied or closely followed

> by discomfort to the animal will, other things being equal, have their connections with that situation weakened, so that, when it recurs, they will be less likely to occur. The greater the satisfaction or discomfort, the greater the strengthening or weakening of the bond.

Although this principle has generated controversy over the years, it remains influential because its general idea is supported by many experiments and it makes such good intuitive sense.

Reinforcement learning is usually formulated mathematically as an optimization problem with the objective of finding an action, or a strategy for producing actions, that is optimal in some well-defined way. Although in practice it is more important that a reinforcement learning system continue to improve than that it actually achieve optimal behavior, optimality objectives provide a useful categorization of reinforcement learning into three basic types, in order of increasing complexity: *nonassociative*, *associative*, and *sequential*. Nonassociative reinforcement learning involves determining which of a set of actions is best in bringing about a satisfactory state of affairs. In associative reinforcement learning, different actions are best in different situations. The objective is to form an optimal *associative mapping* between a set of stimuli and the actions having the best immediate consequences when executed in the situations signaled by those stimuli. Thorndike's Law of Effect refers to this kind of reinforcement learning. Sequential reinforcement learning retains the objective of forming an optimal associative mapping but is concerned with more complex problems in which the relevant consequences of an action are not available immediately after the action is taken. In these cases, the associative mapping represents a strategy, or policy, for acting over time. All of these types of reinforcement learning differ from the more commonly studied paradigm of supervised learning, or "learning with a teacher," in significant ways that I discuss in the course of this chapter.

This chapter is organized into three main sections, each addressing one of these three categories of reinforcement learning. For more detailed treatments, the reader should consult references [Bar92, BBS95, Sut92, Wer92, Kae96].

2 Nonassociative Reinforcement Learning

Figure 1 shows the basic components of a nonassociative reinforcement learning problem. The learning system's actions influence the behavior of some process, which might also be influenced by random or unknown factors (labeled "disturbances" in Figure 1). A *critic* sends the learning system a *reinforcement signal* whose value at any time is a measure of the "goodness" of the current process behavior. Using this information, the learning

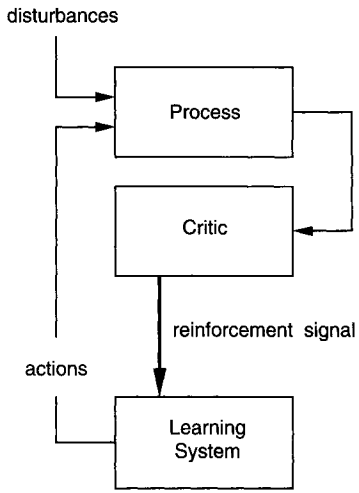

FIGURE 1. Nonassociative reinforcement learning. The learning system's actions influence the behavior of a process, which might also be influenced by random or unknown "disturbances." The critic evaluates the actions' immediate consequences on the process and sends the learning system a reinforcement signal.

system updates its action-generation rule, generates another action, and the process repeats.

An example of this type of problem has been extensively studied by theorists studying *learning automata* [NT89]. Suppose the learning system has m actions a_1, a_2, ..., a_m and that the reinforcement signal simply indicates "success" or "failure." Further, assume that the influence of the learning system's actions on the reinforcement signal can be modeled as a collection of success probabilities d_1, d_2, ..., d_m, where d_i is the probability of success given that the learning system has generated a_i (so that $1 - d_i$ is the probability that the critic signals failure). Each d_i can be any number between 0 and 1 (the d_i's do not have to sum to one), and the learning system has no initial knowledge of these values. The learning system's objective is to asymptotically maximize the probability of receiving "success," which is accomplished when it always performs the action a_j such that $d_j = \max\{d_i | i = 1, \ldots, m\}$. There are many variants of this task, some of which are better known as *m-armed bandit* problems [BF85].

One class of learning systems for this problem consists of *stochastic learning automata* [NT89]. Suppose that on each trial, or time step t, the learning system selects an action $a(t)$ from its set of m actions according to a probability vector $(p_1(t), \ldots, p_n(t))$, where $p_i(t) = Pr\{a(t) = a_i\}$. A stochastic learning automaton implements a commonsense notion of reinforcement learning: if action a_i is chosen on trial t and the critic's feedback is "success," then $p_i(t)$ is increased and the probabilities of the other actions are decreased; whereas if the critic indicates "failure," then $p_i(t)$ is decreased

and the probabilities of the other actions are appropriately adjusted. Many methods that have been studied are similar to the following *linear reward–penalty* (L_{R-P}) method:

If $a(t) = a_i$ and the critic says "success," then

$$\begin{aligned} p_i(t+1) &= p_i(t) + \alpha(1 - p_i(t)), \\ p_j(t+1) &= (1-\alpha)p_j(t), \quad j \neq i. \end{aligned}$$

If $a(t) = a_i$ and the critic says "failure," then

$$\begin{aligned} p_i(t+1) &= (1-\beta)p_i(t), \\ p_j(t+1) &= \frac{\beta}{m-1} + (1-\beta)p_j(t), \quad j \neq i, \end{aligned}$$

where $0 < \alpha < 1$, $0 \leq \beta < 1$.

The performance of a stochastic learning automaton is measured in terms of how the critic's signal tends to change over trials. The probability that the critic signals success on trial t is $M(t) = \sum_{i=1}^{m} p_i(t)d_i$. An algorithm is *optimal* if for all sets of success probabilities $\{d_i\}$,

$$\lim_{t\to\infty} E[M(t)] = d_j,$$

where $d_j = \max\{d_i | i = 1, \ldots, m\}$ and E is the expectation over all possible sequences of trials. An algorithm is said to be *ϵ-optimal* if for all sets of success probabilities and any $\epsilon > 0$ there exist algorithm parameters such that

$$\lim_{t\to\infty} E[M(t)] = d_j - \epsilon.$$

Although no stochastic learning automaton algorithm has been proved to be optimal, the L_{R-P} algorithm given above with $\beta = 0$ is *ϵ-optimal*, where α has to decrease as ϵ decreases. Additional results exist about the behavior of groups of stochastic learning automata forming *teams* (a single critic broadcasts its signal to all the team members) or playing *games* (there is a different critic for each automaton) [NT89].

Following are key observations about nonassociative reinforcement learning:

1. *Uncertainty* plays a key role in nonassociative reinforcement learning, as it does in reinforcement learning in general. For example, if the critic in the example above evaluated actions deterministically (i.e., $d_i = 1$ or 0 for each i), then the problem would be a much simpler optimization problem.

2. The critic is an abstract model of any process that evaluates the learning system's actions. The critic does not need to have direct access

to the actions or have any knowledge about the interior workings of the process influenced by those actions. In motor control, for example, judging the success of a reach or a grasp does not require access to the actions of all the internal components of the motor control system.

3. The reinforcement signal can be any signal evaluating the learning system's actions, and not just the success/failure signal described above. Often it takes on real values, and the objective of learning is to maximize its expected value. Moreover, the critic can use a variety of criteria in evaluating actions, which it can combine in various ways to form the reinforcement signal. Any value taken on by the reinforcement signal is often simply called a *reinforcement* (although this is at variance with traditional use of the term in psychology).

4. The critic's signal does not directly tell the learning system what action is best; it only evaluates the action taken. The critic also does not directly tell the learning system how to change its actions. These are key features distinguishing reinforcement learning from supervised learning, and we discuss them further below. Although the critic's signal is less informative than a training signal in supervised learning, reinforcement learning is not the same as the learning paradigm called *unsupervised learning* because unlike that form of learning, it is guided by external feedback.

5. Reinforcement learning algorithms are *selectional processes*. There must be *variety* in the action–generation process so that the consequences of alternative actions can be compared to select the best. Behavioral variety is called *exploration*; it is often generated through randomness (as in stochastic learning automata), but it need not be. Because it involves selection, nonassociative reinforcement learning is similar to natural selection in evolution. In fact, reinforcement learning in general has much in common with genetic approaches to search and problem solving [Gol89, Hol75].

6. Due to this selectional aspect, reinforcement learning is traditionally described as learning through "trial and error." However, one must take care to distinguish this meaning of "error" from the type of error signal used in supervised learning. The latter, usually a vector, tells the learning system the direction in which it should change each of its action components. A reinforcement signal is less informative. It would be better to describe reinforcement learning as learning through "trial and evaluation."

7. Nonassociative reinforcement learning is the simplest form of learning that involves the conflict between *exploitation* and *exploration*. In deciding which action to take, the learning system has to balance

two conflicting objectives: it has to use what it has already learned to obtain success (or, more generally, to obtain high evaluations), and it has to behave in new ways to learn more. The first is the need to *exploit* current knowledge; the second is the need to *explore* to acquire more knowledge. Because these needs ordinarily conflict, reinforcement learning systems have to somehow balance them. In control engineering, this is known as the conflict between control and identification. This conflict is absent from supervised and unsupervised learning, unless the learning system is also engaged in influencing which training examples it sees.

3 Associative Reinforcement Learning

Because its only input is the reinforcement signal, the learning system in Figure 1 cannot discriminate between different situations, such as different states of the process influenced by its actions. In an associative reinforcement learning problem, in contrast, the learning system receives stimulus patterns as input in addition to the reinforcement signal (Figure 2). The optimal action on any trial depends on the stimulus pattern present on that trial. To give a specific example, consider this generalization of the nonassociative task described above. Suppose that on trial t the learning system

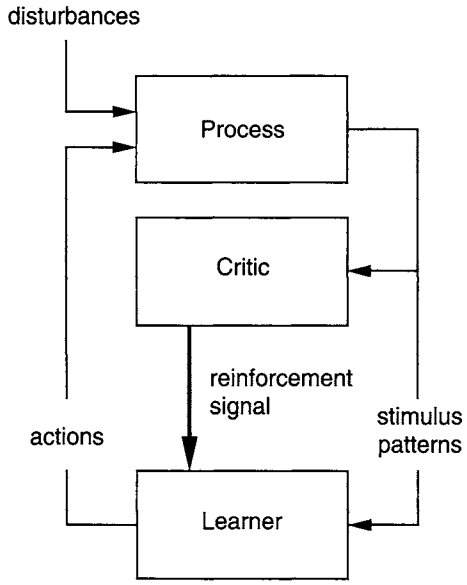

FIGURE 2. Associative reinforcement learning. The learning system receives stimulus patterns in addition to a reinforcement signal. Different actions can be optimal depending on the stimulus patterns.

senses stimulus pattern $x(t)$ and selects an action $a(t) = a_i$ through a process that can depend on $x(t)$. After this action is executed, the critic signals success with probability $d_i(x(t))$ and failure with probability $1 - d_i(x(t))$. The objective of learning is to maximize success probability, achieved when on each trial t the learning system executes the action $a(t) = a_j$, where a_j is the action such that $d_j(x(t)) = \max\{d_i(x(t))|i = 1, \ldots, m\}$.

The learning system's objective is thus to learn an optimal associative mapping from stimulus patterns to actions. Unlike supervised learning, examples of optimal actions are not provided during training; they have to be *discovered* through exploration by the learning system. Learning tasks like this are related to instrumental, or cued operant, tasks studied by animal learning theorists, and the stimulus patterns correspond to discriminative stimuli.

Several associative reinforcement learning rules for neuron-like units have been studied. Figure 3 shows a neuron-like unit receiving a stimulus pattern as input in addition to the critic's reinforcement signal. Let $x(t)$, $w(t)$, $a(t)$, and $r(t)$ respectively denote the stimulus vector, weight vector, action, and the resultant value of the reinforcement signal for trial t. Let $s(t)$ denote the weighted sum of the stimulus components at trial t:

$$s(t) = \sum_{i=1}^{n} w_i(t)x_i(t),$$

where $w_i(t)$ and $x_i(t)$ are respectively the ith components of the weight and stimulus vectors.

Associative Search Unit — One simple associative reinforcement learning rule is an extension of the Hebbian correlation learning rule. This

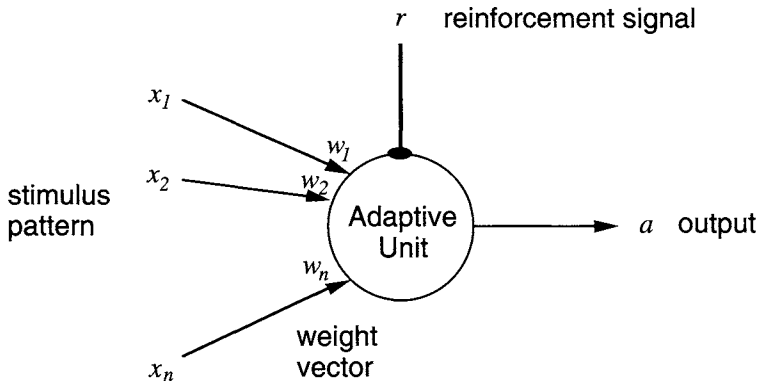

FIGURE 3. A neuron-like adaptive unit. Input pathways labeled x_1 through x_n carry nonreinforcing input signals, each of which has an associated weight w_i, $1 \leq i \leq n$; the pathway labeled r is a specialized input for delivering reinforcement; the unit's output pathway is labeled a.

rule was called the *associative search rule* by Barto, Sutton, and Brouwer [BSB81, BS81, BAS82] and was motivated by Klopf's [Klo72, Klo82] theory of the self-interested neuron. To exhibit variety in its behavior, the unit's output is a random variable depending on the activation level. One way to do this is as follows:

$$a(t) = \begin{cases} 1 & \text{with probability } p(t), \\ 0 & \text{with probability } 1 - p(t), \end{cases} \tag{1}$$

where $p(t)$, which must be between 0 and 1, is an increasing function (such as the logistic function) of $s(t)$. Thus, as the weighted sum increases (decreases), the unit becomes more (less) likely to fire (i.e., to produce an output of 1). The weights are updated according to the following rule:

$$\Delta w(t) = \eta\, r(t) a(t) x(t),$$

where $r(t)$ is $+1$ (success) or -1 (failure).

This is just the Hebbian correlation rule with the reinforcement signal acting as an additional modulatory factor. It is understood that $r(t)$ is the critic's evaluation of the action $a(t)$. In a more real-time version of the learning rule, there must necessarily be a time delay between an action and the resulting reinforcement. In this case, if the critic takes time τ to evaluate an action, the rule appears as follows, with t now acting as a time index instead of a trial number:

$$\Delta w(t) = \eta\, r(t) a(t-\tau) x(t-\tau), \tag{2}$$

where $\eta > 0$ is the learning rate parameter. Thus, if the unit fires in the presence of an input x, possibly just by chance, and this is followed by "success," the weights change so that the unit will be more likely to fire in the presence of x, and inputs similar to x, in the future. A failure signal makes it less likely to fire under these conditions. This rule, which implements the Law of Effect at the neuronal level, makes clear the three factors minimally required for associative reinforcement learning: a stimulus signal, x; the action produced in its presence, a; and the consequent evaluation, r.

Selective Bootstrap and Associative Reward–Penalty Units — Widrow, Gupta, and Maitra [WGM73] extended the Widrow/Hoff, or LMS, learning rule [WS85] so that it could be used in associative reinforcement learning problems. Since the LMS rule is a well-known rule for supervised learning, its extension to reinforcement learning helps illuminate one of the differences between supervised learning and associative reinforcement learning, which Widrow et al. [WGM73] called "learning with a critic." They called their extension of LMS the *selective bootstrap* rule. Unlike the associative search unit described above, a selective bootstrap unit's output is the usual deterministic threshold of the weighted sum:

$$a(t) = \begin{cases} 1 & \text{if } s(t) > 0, \\ 0 & \text{otherwise.} \end{cases}$$

In supervised learning, an LMS unit receives a training signal, $z(t)$, that directly specifies the desired action at trial t and updates its weights as follows:

$$\Delta w(t) = \eta[z(t) - s(t)]x(t). \tag{3}$$

In contrast, a selective bootstrap unit receives a reinforcement signal, $r(t)$, and updates its weights according to this rule:

$$\Delta w(t) = \begin{cases} \eta[a(t) - s(t)]x(t) & \text{if } r(t) = \text{"success"} \\ \eta[1 - a(t) - s(t)]x(t) & \text{if } r(t) = \text{"failure,"} \end{cases}$$

where it is understood that $r(t)$ evaluates $a(t)$. Thus, if $a(t)$ produces "success," the LMS rule is applied with $a(t)$ playing the role of the desired action. Widrow et al. [WGM73] called this "positive bootstrap adaptation": weights are updated as if the output actually produced was in fact the desired action. On the other hand, if $a(t)$ leads to "failure," the desired action is $1 - a(t)$, i.e., the action that was *not* produced. This is "negative bootstrap adaptation." The reinforcement signal switches the unit between positive and negative bootstrap adaptation, motivating the term "selective bootstrap adaptation." Widrow et al. [WGM73] showed how this unit was capable of learning a strategy for playing blackjack, where wins were successes and losses were failures. However, the learning ability of this unit is limited because it lacks variety in its behavior.

A closely related unit is the *associative reward–penalty* (A_{R-P}) unit of Barto and Anandan [BA85]. It differs from the selective bootstrap algorithm in two ways. First, the unit's output is a random variable like that of the associative search unit (Equation 1). Second, its weight–update rule is an *asymmetric* version of the selective bootstrap rule:

$$\Delta w(t) = \begin{cases} \eta[a(t) - s(t)]x(t) & \text{if } r(t) = \text{"success"} \\ \lambda\eta[1 - a(t) - s(t)]x(t) & \text{if } r(t) = \text{"failure,"} \end{cases}$$

where $0 \leq \lambda \leq 1$ and $\eta > 0$. This is a special case of a class of A_{R-P} rules for which Barto and Anandan [BA85] proved a convergence theorem giving conditions under which it asymptotically maximizes the probability of success in associative reinforcement learning tasks like those described above. The rule's asymmetry is important because its asymptotic performance improves as λ approaches zero.

One can see from the selective bootstrap and A_{R-P} units that a reinforcement signal is less informative than a signal specifying a desired action. It is also less informative than the error $z(t) - a(t)$ used by the LMS rule. Because this error is a signed quantity, it tells the unit *how*, i.e., in what direction, it should change its action. A reinforcement signal — by itself — does not convey this information. If the learner has only two actions, as in a selective bootstrap unit, it is easy to deduce, or at least estimate, the desired action from the reinforcement signal and the actual action. However,

if there are more than two actions, the situation is more difficult because the reinforcement signal does not provide information about actions that were not taken.

Stochastic Real-Valued Unit — One approach to associative reinforcement learning when there are more than two actions is illustrated by the *stochastic real-valued* (SRV) unit of Gullapalli [Gul90]. On any trial t, an SRV unit's output is a real number, $a(t)$, produced by applying a function f, such as the logistic function, to the weighted sum, $s(t)$, plus a random number $\texttt{noise}(t)$:

$$a(t) = f[s(t) + \texttt{noise}(t)].$$

The random number $\texttt{noise}(t)$ is selected according to a mean-zero Gaussian distribution with standard deviation $\sigma(t)$. Thus, $f[s(t)]$ gives the *expected* output on trial t, and the actual output varies about this value, with $\sigma(t)$ determining the amount of exploration the unit exhibits on trial t.

Before describing how the SRV unit determines $\sigma(t)$, we describe how it updates the weight vector $w(t)$. The weight-update rule requires an estimate of the amount of reinforcement expected for acting in the presence of stimulus $x(t)$. This is provided by a supervised-learning process that uses the LMS rule to adjust another weight vector, v, used to determine the reinforcement estimate $\hat{r}$:

$$\hat{r}(t) = \sum_{i=1}^{m} v_i(t) x_i(t),$$

with

$$\Delta v(t) = \eta[r(t) - \hat{r}(t)]x(t).$$

Given this $\hat{r}(t)$, $w(t)$ is updated as follows:

$$\Delta w(t) = \eta[r(t) - \hat{r}(t)] \left[\frac{\texttt{noise}(t)}{\sigma(t)}\right] x(t),$$

where $\eta > 0$ is a learning-rate parameter. Thus, if $\texttt{noise}(t)$ is positive, meaning that the unit's output is larger than expected, and the unit receives more than the expected reinforcement, the weights change to increase the expected output in the presence of $x(t)$; if it receives less than the expected reinforcement, the weights change to decrease the expected output. The reverse happens if $\texttt{noise}(t)$ is negative. Dividing by $\sigma(t)$ normalizes the weight change. Changing σ during learning changes the amount of exploratory behavior the unit exhibits.

Gullapalli [Gul90] suggests computing $\sigma(t)$ as a monotonically decreasing function of $\hat{r}(t)$. This implies that the amount of exploration for any stimulus vector decreases as the amount of reinforcement expected for acting in the presence of that stimulus vector increases. As learning proceeds, the SRV unit tends to act with increasing determinism in the presence of

stimulus vectors for which it has learned to achieve large reinforcement signals. This is somewhat like simulated annealing [KGV83] except that it is stimulus-dependent and is controlled by the progress of learning. SRV units have been used as output units of reinforcement learning networks in a number of applications (e.g., references [GGB92, GBG94]).

Weight Perturbation — For the units described above (except the selective bootstrap unit), behavioral variability is achieved by including random variation in the unit's output. Another approach is to randomly vary the weights. Following Alspector et al. [AMY+93], let δw be a vector of small perturbations, one for each weight, that are independently selected from some probability distribution. Letting J denote the function evaluating the system's behavior, the weights are updated as follows:

$$\Delta w = -\eta \left[\frac{J(w + \delta w) - J(w)}{\delta w} \right], \tag{4}$$

where $\eta > 0$ is a learning-rate parameter. This is a gradient descent learning rule that changes weights according to an estimate of the gradient of $\mathcal{E}$ with respect to the weights. Alspector et al. [AMY+93] say that the method *measures* the gradient instead of *calculating* it as the LMS and error backpropagation [RHW86] algorithms do. This approach has been proposed by several researchers for updating the weights of a unit, or of a network, during supervised learning, where J gives the error over the training examples. However, J can be any function evaluating the unit's behavior, including a reinforcement function (in which case, the sign of the learning rule would be changed to make it a gradient *ascent* rule).

Another weight perturbation method for neuron-like units is provided by Unnikrishnan and Venugopal's [KPU94] use of the *Alopex* algorithm, originally proposed by Harth and Tzanakou [HT74], for adjusting a unit's (or a network's) weights. A somewhat simplified version of the weight-update rule is the following:

$$\Delta w(t) = \eta d(t), \tag{5}$$

where η is the learning-rate parameter and $d(t)$ is a vector whose components, $d_i(t)$, are equal to either $+1$ or -1. After the first two iterations, in which they are assigned randomly, successive values are determined by

$$d_i(t) = \begin{cases} d_i(t-1) & \text{with probability } p(t), \\ -d_i(t-1) & \text{with probability } 1 - p(t). \end{cases}$$

Thus, $p(t)$ is the probability that the direction of the change in weight w_i from iteration t to iteration $t+1$ will be the same as the direction it changed from iteration $t-2$ to $t-1$, whereas $1 - p(t)$ is the probability that the weight will move in the opposite direction. The probability $p(t)$ is a function

of the change in the value of the objective function from iteration $t-1$ to t; specifically, $p(t)$ is a positive increasing function of $J(t)-J(t-1)$, where $J(t)$ and $J(t-1)$ are respectively the values of the function evaluating the behavior of the unit at iterations t and $t-1$. Consequently, if the unit's behavior has moved uphill by a large amount, as measured by J, from iteration $t-1$ to iteration t, then $p(t)$ will be large, so that the probability of the next step in weight space being in the same direction as the preceding step will be high. On the other hand, if the unit's behavior moved downhill, then the probability will be high that some of the weights will move in the opposite direction, i.e., that the step in weight space will be in some new direction.

Although weight perturbation methods are of interest as alternatives to error backpropagation for adjusting network weights in supervised learning problems, they utilize reinforcement learning principles by estimating performance through active exploration, in this case achieved by adding random perturbations to the weights. In contrast, the other methods described above — at least to a first approximation — use active exploration to estimate the gradient of the reinforcement function with respect to a unit's *output* instead of its weights. The gradient with respect to the weights can then be estimated by differentiating the known function by which the weights influence the unit's output. Both approaches — weight perturbation and unit-output perturbation — lead to learning methods for networks to which we now turn our attention.

Reinforcement Learning Networks — The neuron-like units described above can be readily used to form networks. The weight perturbation approach carries over directly to networks by simply letting w in Equations 4 and 5 be the vector consisting of all the network's weights. A number of researchers have achieved success using this approach in supervised learning problems. In these cases, one can think of each weight as facing a reinforcement learning task (which is in fact nonassociative), even though the network as a whole faces a supervised learning task. A significant advantage of this approach is that it applies to networks with arbitrary connection patterns, not just to feedforward networks.

Networks of A_{R-P} units have been used successfully in both supervised and associative reinforcement learning tasks ([Bar85, BJ87]), although only with feedforward connection patterns. For supervised learning, the output units learn just as they do in error backpropagation, but the hidden units learn according to the A_{R-P} rule. The reinforcement signal, which is defined to increase as the output error decreases, is simply *broadcast* to all the hidden units, which learn simultaneously. If the network as a whole faces an associative reinforcement learning task, all the units are A_{R-P} units, to which the reinforcement signal is uniformly broadcast (Figure 4). The units exhibit a kind of *statistical cooperation* in trying to increase their common reinforcement signal (or the probability of success if it is a success/failure

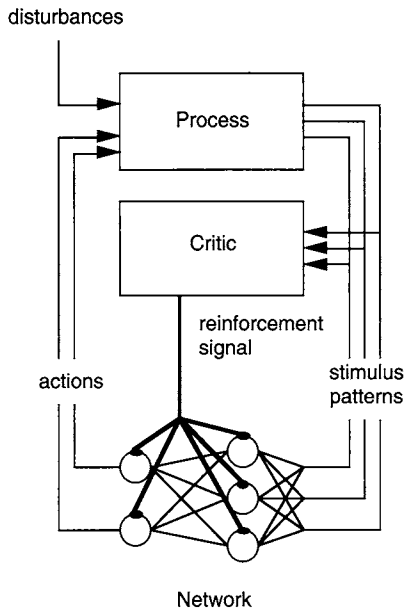

FIGURE 4. A network of associative reinforcement units. The reinforcement signal is broadcast to all the units.

signal) [Bar85]. Networks of associative search units and SRV units can be similarly trained, but these units do not perform well as hidden units in multilayer networks.

Methods for updating network weights fall within a spectrum of possibilities ranging from weight perturbation methods that do not take advantage of *any* of a network's structure to algorithms like error backpropagation, which take full advantage of network structure to compute gradients. Unit-output perturbation methods fall between these extremes by taking advantage of the structure of individual units but not of the network as a whole. Computational studies provide ample evidence that all of these methods can be effective, and each method has its own advantages, with perturbation methods usually sacrificing learning speed for generality and ease of implementation. Perturbation methods are also of interest due to their relative biological plausibility compared to error backpropagation.

Another way to use reinforcement learning units in networks is to use them only as output units, with hidden units being trained via error backpropagation. Weight changes of the output units determine the quantities that are backpropagated. This approach allows the function approximation success of the error backpropagation algorithm to be enlisted in associative reinforcement learning tasks (e.g. reference [GGB92]).

The error backpropagation algorithm can be used in another way in associative reinforcement learning problems. It is possible to train a multi-

layer network to form a model of the process by which the critic evaluates actions. The network's input consists of the stimulus pattern $x(t)$ as well as the current action vector $a(t)$, which is generated by another component of the system. The desired output is the critic's reinforcement signal, and training is accomplished by backpropagating the error

$$r(t) - \hat{r}(t),$$

where $\hat{r}(t)$ is the network's output at time t. After this model is trained sufficiently, it is possible to estimate the gradient of the reinforcement signal with respect to each component of the action vector by analytically differentiating the model's output with respect to its action inputs (which can be done efficiently by backpropagation). This gradient estimate is then used to update the parameters of the action-generation component. Jordan and Jacobs [JJ90] illustrate this approach. Note that the exploration required in reinforcement learning is conducted in the model-learning phase of this approach instead in the action-learning phase.

It should be clear from this discussion of reinforcement learning networks that there are many different approaches to solving reinforcement learning problems. Furthermore, although reinforcement learning *tasks* can be clearly distinguished from supervised and unsupervised learning tasks, it is more difficult to precisely define a class of reinforcement learning *algorithms*.

4 Sequential Reinforcement Learning

Sequential reinforcement requires improving the long-term consequences of an action, or of a strategy for performing actions, in addition to short-term consequences. In these problems, it can make sense to forgo short-term performance in order to achieve better performance over the long term. Tasks having these properties are examples of *optimal control problems*, sometimes called *sequential decision problems* when formulated in discrete time.

Figure 2, which shows the components of an associative reinforcement learning system, also applies to sequential reinforcement learning, where the box labeled "process" is a system being controlled. A sequential reinforcement learning system tries to influence the behavior of the process in order to maximize a measure of the total amount of reinforcement that will be received over time. In the simplest case, this measure is the sum of the future reinforcement values, and the objective is to learn an associative mapping that at time step t selects, as a function of the stimulus pattern $x(t)$, an action $a(t)$ that maximizes

$$\sum_{k=0}^{\infty} r(t+k),$$

where $r(t+k)$ is the reinforcement signal at step $t+k$. Such an associative mapping is called a *policy.*

Because this sum might be infinite in some problems, and because the learning system usually has control only over its expected value, researchers often consider the following *discounted sum* instead:

$$E\{r(t) + \gamma r(t+1) + \gamma^2 r(t+2) + \cdots\} = E\{\sum_{k=0}^{\infty} \gamma^k r(t+k)\}, \tag{6}$$

where E is the expectation over all possible future behavior patterns of the process. The discount factor determines the present value of future reinforcement: a reinforcement value received k time steps in the future is worth γ^k times what it would be worth if it were received now. If $0 \leq \gamma < 1$, this infinite discounted sum is finite as long as the reinforcement values are bounded. If $\gamma = 0$, the robot is "myopic" in being only concerned with maximizing immediate reinforcement; this is the associative reinforcement learning problem discussed above. As γ approaches one, the objective explicitly takes future reinforcement into account: the robot becomes more farsighted.

An important special case of this problem occurs when there is no immediate reinforcement until a goal state is reached. This is a *delayed reward* problem in which the learning system has to learn how to make the process enter a goal state. Sometimes the objective is to make it enter a goal state as quickly as possible. A key difficulty in these problems has been called the *temporal credit-assignment problem:* When a goal state is finally reached, which of the decisions made earlier deserve credit for the resulting reinforcement? A widely studied approach to this problem is to learn an *internal evaluation function* that is more informative than the evaluation function implemented by the external critic. An *adaptive critic* is a system that learns such an internal evaluation function.

Samuel's Checker Player — Samuel's [Sam59] checkers playing program has been a major influence on adaptive critic methods. The checkers player selects moves by using an evaluation function to compare the board configurations expected to result from various moves. The evaluation function assigns a score to each board configuration, and the system make the move expected to lead to the configuration with the highest score. Samuel used a method to improve the evaluation function through a process that compared the score of the current board position with the score of a board position likely to arise later in the game:

> We are attempting to make the score, calculated for the current board position, look like that calculated for the terminal board position of the chain of moves which most probably occur during actual play [Sam59].

As a result of this process of "backing up" board evaluations, the evaluation function should improve in its ability to evaluate long-term consequences of moves. In one version of Samuel's system, the evaluation function was represented as a weighted sum of numerical features, and the weights were adjusted based on an error derived by comparing evaluations of current and predicted board positions.

If the evaluation function can be made to score each board configuration according to its true promise of eventually leading to a win, then the best strategy for playing is to myopically select each move so that the next board configuration is the most highly scored. If the evaluation function is optimal in this sense, then it already takes into account all the possible future courses of play. Methods such as Samuel's that attempt to adjust the evaluation function toward this ideal optimal evaluation function are of great utility.

Adaptive Critic Unit and Temporal Difference Methods — An adaptive critic unit is a neuron-like unit that implements a method similar to Samuel's. The unit is as in Figure 3 except that its output at time step t is $P(t) = \sum_{i=1}^{n} w_i(t)x_i(t)$, so denoted because it is a prediction of the discounted sum of future reinforcement given in Equation 6. The adaptive critic learning rule rests on noting that correct predictions must satisfy a consistency condition, which is a special case of the Bellman optimality equation, relating predictions at adjacent time steps. Suppose that the predictions at any two successive time steps, say steps t and $t+1$, are correct. This means that

$$\begin{aligned} P(t) &= E\{r(t) + \gamma r(t+1) + \gamma^2 r(t+2) + \cdots\}, \\ P(t+1) &= E\{r(t+1) + \gamma r(t+2) + \gamma^2 r(t+3) + \cdots\}. \end{aligned}$$

Now notice that we can rewrite $P(t)$ as follows:

$$P(t) = E\{r(t) + \gamma[r(t+1) + \gamma r(t+2) + \cdots]\}.$$

But this is exactly the same as

$$P(t) = E\{r(t)\} + \gamma P(t+1).$$

An estimate of the error by which any two adjacent predictions fail to satisfy this consistency condition is called the *temporal difference (TD) error* [Sut88]:

$$r(t) + \gamma P(t+1) - P(t), \tag{7}$$

where $r(t)$ is used as an unbiased estimate of $E\{r(t)\}$. The term *temporal difference* comes from the fact that this error essentially depends on the difference between the critic's predictions at successive time steps.

The adaptive critic unit adjusts its weights according to the following learning rule:

$$\Delta w(t) = \eta[r(t) + \gamma P(t+1) - P(t)]x(t). \tag{8}$$

A subtlety here is that $P(t+1)$ should be computed using the weight vector $w(t)$, not $w(t+1)$. This rule changes the weights to decrease the magnitude of the TD error. Note that if $\gamma = 0$, Equation 8 is equivalent to the LMS learning rule (Equation 3). In analogy with the LMS rule, we can think of $r(t) + \gamma P(t+1)$ as the prediction target: it is the quantity that each $P(t)$ should match. The adaptive critic is therefore trying to predict the next reinforcement, $r(t)$, *plus its own next prediction* (discounted), $\gamma P(t+1)$. The adaptive critic is similar to Samuel's learning method in adjusting weights to make current predictions closer to later predictions.

Although this method is very simple computationally, it actually converges to the correct predictions of the discounted sum of future reinforcement if these correct predictions can be computed by a linear unit. This is shown by Sutton [Sut88], who discusses a more general class of methods, called *TD methods*, that include Equation 8 as a special case. It is also possible to learn nonlinear predictions using, for example, multilayer networks trained by back propagating the TD error. Using this approach, Tesauro [Tes92] produced a system that learned how to play expert-level backgammon.

Actor–Critic Architectures—In an actor–critic architecture, the predictions formed by an adaptive critic act as reinforcement for an associative reinforcement learning component, called the *actor* (Figure 5). To distinguish the adaptive critic's signal from the reinforcement signal sup-

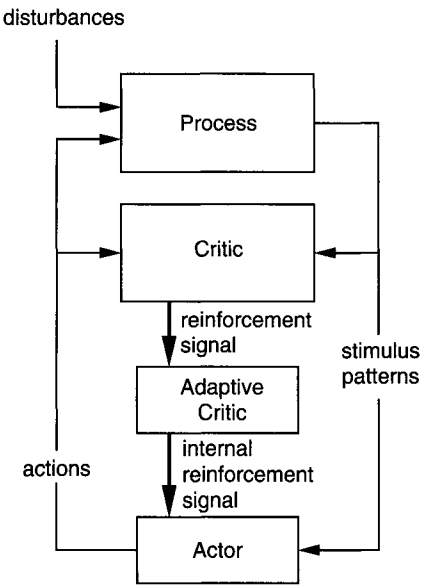

FIGURE 5. Actor–critic architecture. An adaptive critic provides an internal reinforcement signal to an *actor*, which learns a policy for controlling the process.

plied by the original, nonadaptive critic, we call it the *internal reinforcement signal.* The actor tries to maximize the *immediate* internal reinforcement signal, while the adaptive critic tries to predict total future reinforcement. To the extent that the adaptive critic's predictions of total future reinforcement are correct given the actor's current policy, the actor actually learns to increase the total amount of future reinforcement (as measured, for example, by Equation 6).

Barto, Sutton, and Anderson [BSA83] used this architecture for learning to balance a simulated pole mounted on a cart. The actor had two actions: application of a force of a fixed magnitude to the cart in the plus or minus direction. The nonadaptive critic only provided a signal of failure when the pole fell past a certain angle or the cart hit the end of the track. The stimulus patterns were vectors representing the state of the cart–pole system. The actor was an associative search unit as described above except that it used an *eligibility trace* [Klo82] in its weight-update rule:

$$\Delta w(t) = \eta\, \hat{r}(t) a(t) \bar{x}(t),$$

where $\hat{r}(t)$ is the internal reinforcement signal and $\bar{x}(t)$ is an exponentially-decaying trace of past input patterns. When a component of this trace is nonzero, the corresponding synapse is *eligible* for modification. This technique is used instead of the delayed stimulus pattern in Equation 2 to improve the rate of learning. It is assumed that $\hat{r}(t)$ evaluates the action $a(t)$. The internal reinforcement is the TD error used by the adaptive critic:

$$\hat{r}(t) = r(t) + \gamma P(t+1) - P(t).$$

This makes the original reinforcement signal, $r(t)$, available to the actor, as well as changes in the adaptive critic's predictions of future reinforcement, $\gamma P(t+1) - P(t)$.

Action-Dependent Adaptive Critics — Another approach to sequential reinforcement learning combines the actor and adaptive critic into a single component that learns separate predictions for each action. At each time step the action with the largest prediction is selected, except for a random exploration factor that causes other actions to be selected occasionally. An algorithm for learning action-dependent predictions of future reinforcement, called the *Q-learning* algorithm, was proposed by Watkins in 1989, who proved that it converges to the correct predictions under certain conditions [WD92]. The term *action-dependent adaptive critic* was first used by Lukes, Thompson, and Werbos [LTW90], who presented a similar idea. A little-known forerunner of this approach was presented by Bozinovski [Boz82].

For each pair (x, a) consisting of a process state, x, and and a possible action, a, let $Q(x, a)$ denote the total amount of reinforcement that will be produced over the future if action a is executed when the process is in

state x and optimal actions are selected thereafter. Q-learning is a simple on-line algorithm for estimating this function Q of state-action pairs. Let Q_t denote the estimate of Q at time step t. This is stored in a look-up table with an entry for each state-action pair. Suppose the learning system observes the process state $x(t)$, executes action $a(t)$, and receives the resulting immediate reinforcement $r(t)$. Then

$$\Delta Q_t(x,a) = \begin{cases} \eta(t)[r(t) + \gamma P(t+1) - Q_t(x,a)] & \text{if } x = x(t) \text{ and } a = a(t), \\ 0 & \text{otherwise,} \end{cases}$$

where $\eta(t)$ is a positive learning-rate parameter that depends on t and

$$P(t+1) = \max_{a \in A(t+1)} Q_t(x(t+1), a),$$

with $A(t+1)$ denoting the set of all actions available at $t+1$. If this set consists of a single action for all t, Q-learning reduces to a look-up-table version of the adaptive critic learning rule (Equation 8). Although the Q-learning convergence theorem requires look-up-table storage (and therefore finite state and action sets), many researchers have heuristically adapted Q-learning to more general forms of storage, including multilayer neural networks trained by backpropagation of the Q-learning error.

Dynamic Programming—Sequential reinforcement learning problems (in fact, all reinforcement learning problems) are examples of stochastic optimal control problems. Among the traditional methods for solving these problems are dynamic programming (DP) algorithms. As applied to optimal control, DP consists of methods for successively approximating optimal evaluation functions and optimal decision rules for both deterministic and stochastic problems. Bertsekas [Ber87] provides a good treatment of these methods. A basic operation in all DP algorithms is "backing up" evaluations in a manner similar to the operation used in Samuel's method and in the adaptive critic and Q-learning algorithms.

Recent reinforcement learning theory exploits connections with DP algorithms while emphasizing important differences. For an overview and guide to the literature, see [Bar92, BBS95, Sut92, Wer92, Kae96]. Following is a summary of key observations.

1. Because conventional dynamic programming algorithms require multiple exhaustive "sweeps" of the process state set (or a discretized approximation of it), they are not practical for problems with very large finite-state sets or high-dimensional continuous state spaces. Sequential reinforcement learning algorithms *approximate* DP algorithms in ways designed to reduce this computational complexity.

2. Instead of requiring exhaustive sweeps, sequential reinforcement learning algorithms operate on states as they occur in actual or simulated

experiences in controlling the process. It is appropriate to view them as *Monte Carlo* DP algorithms.

3. Whereas conventional DP algorithms require a complete and accurate model of the process to be controlled, sequential reinforcement learning algorithms do not require such a model. Instead of computing the required quantities (such as state evaluations) from a model, they estimate these quantities from experience. However, reinforcement learning methods can also take advantage of models to improve their efficiency.

4. Conventional DP algorithms require look-up-table storage of evaluations or actions for all states, which is impractical for large problems. Although this is also required to guarantee convergence of reinforcement learning algorithms, such as Q-learning, these algorithms can be adapted for use with more compact storage means, such as neural networks.

It is therefore accurate to view sequential reinforcement learning as a collection of heuristic methods providing computationally feasible approximations of DP solutions to stochastic optimal control problems. Emphasizing this view, Werbos [Wer92] uses the term *heuristic dynamic programming* for this class of methods.

5 Conclusion

The increasing interest in reinforcement learning is due to its applicability to learning by autonomous robotic agents. Although both supervised and unsupervised learning can play essential roles in reinforcement learning systems, these paradigms by themselves are not general enough for learning while acting in a dynamic and uncertain environment. Among the topics being addressed by current reinforcement learning research are extending the theory of sequential reinforcement learning to include generalizing function approximation methods; understanding how exploratory behavior is best introduced and controlled; sequential reinforcement learning when the process state cannot be observed; how problem-specific knowledge can be effectively incorporated into reinforcement learning systems; the design of modular and hierarchical architectures; and the relationship to brain reward mechanisms.

Acknowledgments: This chapter is an expanded and revised version of "Reinforcement Learning" by Andrew G. Barto, which appeared in the *Handbook of Brain Theory and Neural Networks*, M. A. Arbib, editor, pp. 804-809. MIT Press: Cambridge, Massachusetts, 1995.

6 REFERENCES

[AMY+93] J. Alspector, R. Meir, B. Yuhas, A. Jayakumar, and D. Lippe. A parallel gradient descent method for learning in analog VLSI neural networks. In S. J. Hanson, J. D. Cohen, and C. L. Giles, editors, *Advances in Neural Information Processing Systems 5*, pages 836–844, Morgan Kaufmann, San Mateo, California, 1993.

[BA85] A. G. Barto and P. Anandan. Pattern recognizing stochastic learning automata. *IEEE Transactions on Systems, Man, and Cybernetics*, 15:360–375, 1985.

[Bar85] A. G. Barto. Learning by statistical cooperation of self-interested neuron-like computing elements. *Human Neurobiology*, 4:229–256, 1985.

[Bar92] A. G. Barto. Reinforcement learning and adaptive critic methods. In D. A. White and D. A. Sofge, editors, *Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches*, pages 469–491. Van Nostrand Reinhold, New York, 1992.

[BAS82] A. G. Barto, C. W. Anderson, and R. S. Sutton. Synthesis of nonlinear control surfaces by a layered associative search network. *Biological Cybernetics*, 43:175–185, 1982.

[BBS95] A. G. Barto, S. J. Bradtke, and S. P. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72:81–138, 1995.

[Ber87] D. P. Bertsekas. *Dynamic Programming: Deterministic and Stochastic Models.* Prentice-Hall, Englewood Cliffs, New Jersey, 1987.

[BF85] D. A. Berry and B. Fristedt. *Bandit Problems.* Chapman and Hall, London, 1985.

[BJ87] A. G. Barto and M. I. Jordan. Gradient following without back-propagation in layered networks. In M. Caudill and C. Butler, editors, *Proceedings of the IEEE First Annual Conference on Neural Networks*, II-629 – II-636, San Diego, 1987.

[Boz82] S. Bozinovski. A self-learning system using secondary reinforcement. In R. Trappl, editor, *Cybernetics and Systems.* North-Holland, Amsterdam, 1982.

[BS81] A. G. Barto and R. S. Sutton. Landmark learning: An illustration of associative search. *Biological Cybernetics*, 42:1–8, 1981.

[BSA83] 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. Reprinted in J. A. Anderson and E. Rosenfeld, *Neurocomputing: Foundations of Research*, MIT Press, Cambridge, Massachusetts, 1988.

[BSB81] A. G. Barto, R. S. Sutton, and P. S. Brouwer. Associative search network: A reinforcement learning associative memory. *IEEE Transactions on Systems, Man, and Cybernetics*, 40:201–211, 1981.

[GBG94] V. Gullapalli, A. G. Barto, and R. A. Grupen. Learning admittance mappings for force-guided assembly. In *Proceedings of the 1994 International Conference on Robotics and Automation*, pages 2633–2638, 1994.

[GGB92] V. Gullapalli, R. A. Grupen, and A. G. Barto. Learning reactive admittance control. In *Proceedings of the 1992 IEEE Conference on Robotics and Automation*, pages 1475–1480. IEEE, Piscataway, New Jersey, 1992.

[Gol89] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, Reading, Massachusetts, 1989.

[Gul90] V. Gullapalli. A stochastic reinforcement algorithm for learning real-valued functions. *Neural Networks*, 3:671–692, 1990.

[Hol75] J. H. Holland. *Adaptation in Natural and Artificial Systems.* Univ. of Michigan Press, Ann Arbor, 1975.

[HT74] E. Harth and E. Tzanakou. Alopex: A stochastic method for determining visual receptive fields. *Vision Research*, 14:1475–1482, 1974.

[JJ90] M. I. Jordan and R. A. Jacobs. Learning to control an unstable system with forward modeling. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, Morgan Kaufmann, San Mateo, California, 1990.

[Kae96] L. P. Kaelbling, editor. *Special Issue on Reinforcement Learning*, volume 22. *Machine Learning*, 1996.

[KGV83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[Kim61] G. A. Kimble. *Hilgard and Marquis' Conditioning and Learning.* Appleton-Century-Crofts, New York, 1961.

[Klo72] A. H. Klopf. Brain function and adaptive systems—A heterostatic theory. Technical Report AFCRL-72-0164, Air Force Cambridge Research Laboratories, Bedford, MA, 1972. A summary appears in *Proceedings of the International Conference on Systems, Man, and Cybernetics*, IEEE Systems, Man, and Cybernetics Society, Dallas, Texas, 1974.

[Klo82] A. H. Klopf. *The Hedonistic Neuron: A Theory of Memory, Learning, and Intelligence.* Hemisphere, Washington, D.C., 1982.

[KPU94] K. P. Venugopal K. P. Unnikrishnan. Alopex: A correlation-based learning algorithm for feed-forward and recurrent neural networks. *Neural Computation*, 6:469–490, 1994.

[LTW90] G. Lukes, B. Thompson, and P. Werbos. Expectation driven learning with an associative memory. In *Proceedings of the International Joint Conference on Neural Networks*, pages I-521 to I-524. Lawrence Erlbaum, Hillsdale, New Jersey, 1990.

[MM70] J. M. Mendel and R. W. McLaren. Reinforcement learning control and pattern recognition systems. In J. M. Mendel and K. S. Fu, editors, *Adaptive, Learning and Pattern Recognition Systems: Theory and Applications*, pages 287–318. Academic Press, New York, 1970.

[NT89] K. Narendra and M. A. L. Thathachar. *Learning Automata: An Introduction.* Prentice-Hall, Englewood Cliffs, New Jersey, 1989.

[RHW86] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, volume 1: Foundations.* Bradford Books/MIT Press, Cambridge, Massachusetts, 1986.

[Sam59] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal on Research and Development*, pages 210–229, 1959. Reprinted in E. A. Feigenbaum and J. Feldman, editors, *Computers and Thought*, pages 71–105. McGraw-Hill, New York, 1963.

[Sut88] R. S. Sutton. Learning to predict by the method of temporal differences. *Machine Learning*, 3:9–44, 1988.

[Sut92] R. S. Sutton, editor. *A Special Issue of Machine Learning on Reinforcement Learning*, volume 8. *Machine Learning*, 1992.

Also published as *Reinforcement Learning*, Kluwer Academic Press, Boston, Massachusetts, 1992.

[Tes92] G. J. Tesauro. Practical issues in temporal difference learning. *Machine Learning*, 8:257–277, 1992.

[Tho11] E. L. Thorndike. *Animal Intelligence.* Hafner, Darien, Connecticut, 1911.

[WD92] C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.

[Wer92] P. J. Werbos. Approximate dynamic programming for real-time control and neural modeling. In D. A. White and D. A. Sofge, editors, *Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches*, pages 493–525. Van Nostrand Reinhold, New York, 1992.

[WGM73] B. Widrow, N. K. Gupta, and S. Maitra. Punish/reward: Learning with a critic in adaptive threshold systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 5:455–465, 1973.

[WS85] B. Widrow and S. D. Stearns. *Adaptive Signal Processing.* Prentice-Hall, Englewood Cliffs, New Jersey, 1985.