Encyclopedia of Cognitive Science, in press

Reinforcement Learning

Peter Dayan
Gatsby Computational Neuroscience Unit
University College London
17 Queen Square
London WC1N 3AR

Tel: +44 (0) 20 7679 1175
Fax: +44 (0) 20 7679 1173
Email: dayan@gatsby.ucl.ac.uk

Christopher JCH Watkins
Department of Computer Science
Royal Holloway, University of London
Egham
Surrey TW20 0EX

Tel: +44 (0) 1784 443419
Fax: +44 (0) 1784 439786
Email: chrisw@dcs.rhbnc.ac.uk

Table of Contents

    - Backgammon
    - Robots
    - Finance
- Summary and future directions

Processed using LaTeX2e

Glossary

Markov chain a model for a random process that evolves over time such that the states (like locations in a maze) occupied in the future are independent of the states in the past given the current state.

Markov decision problem (MDP) a model for a controlled random process in which an agent's choice of action determines the probabilities of transitions of a Markov chain and lead to rewards (or costs) that need to be maximised (or minimised).

policy a deterministic or stochastic scheme for choosing an action at every state or location.

reward an immediate, possibly stochastic, payoff that results from performing an action in a state. In an MDP, the immediate reward depends on the current state and action only. The agent's aim is to optimise a sum or average of (possibly discounted) future rewards.

value function a function defined over states, which gives an estimate of the total (possibly discounted) reward expected in the future, starting from each state, and following a particular policy.

discounting if rewards received in the far future are worth less than rewards received sooner, they are described as being discounted. Humans and animals appear to discount future rewards hyperbolically; exponential discounting is common in engineering and finance.

dynamic programming a collection of calculation techniques (notably policy and value iteration) for finding a policy that maximises reward or minimises costs.

temporal difference prediction error a measure of the inconsistency between estimates of the value function at two successive states. This prediction error can be used to improve the predictions and also to choose good actions.

Reinforcement Learning

Secondary

Computation

dynamic programming#value function#policy#actor-critic#Q-learning

Dayan, Peter

Peter Dayan

Gatsby Computational Neuroscience Unit, University College London, UK

Watkins, Christopher JCH

Christopher Watkins

Department of Computer Science, Royal Holloway College, UK

Reinforcement learning is the study of how animals and artificial systems can learn to optimize their behavior in the face of rewards and punishments. Reinforcement learning algorithms have been developed that are closely related to methods of dynamic programming, which is a general approach to optimal control. Reinforcement learning phenomena have been observed in psychological studies of animal behavior, and in neurobiological investigations of neuromodulation and addiction.

1 Introduction

One way in which animals acquire complex behaviors is by learning to obtain rewards and to avoid punishments. Reinforcement learning theory is a formal computational model of this type of learning. To see that some theory is needed, consider that in many environments animals need to perform unrewarded or unpleasant preparatory actions in order to obtain some later reward. For example, a mouse may need to leave the warmth and safety of its burrow to go on a cold and initially unrewarded search for food.

Is it possible to explain learning to take such an unpleasant decision in terms of learning to obtain a large but distant reward? What computational abilities does an agent need in principle for this type of learning? What capacity for episodic memory is necessary? Does an agent need to be able to plan ahead by envisioning future situations and actions? If an agent is doing something without immediate reward, how does it learn to recognize whether it is making progress towards a desirable goal? It is these questions that the theory of reinforcement learning (RL) seeks to answer.

RL theory comprises a formal framework for describing an agent interacting with its environment, together with a family of learning algorithms, and analyses of their performance.

2 The Reinforcement Learning Framework

In the standard framework for RL, a learning agent—an animal or perhaps a robot—repeatedly observes the state of its environment, and then chooses and performs an action. Performing the action changes the state of the world, and the agent also obtains an immediate numeric payoff as a result. Positive payoffs are rewards and negative payoffs are punishments. The agent must learn to choose actions so as to maximize a long term sum or average of the future payoffs it will receive.

The agent's payoffs are subjective, in the sense that they are the agent's own evaluation of its experience. The ability to evaluate its experience in this way is an essential part of the agent's prior knowledge. The payoffs define what the agent is trying to achieve; what the agent needs to learn is how to choose actions to obtain high payoffs. For this framework to be sensible, the agent must be able to visit the same or at least similar states on many occasions, to take advantage of learning.

A standard assumption is that the agent is able to observe those aspects of the state of the world that are relevant to deciding what to do. This assumption is convenient rather than plausible, since it leads to great simplification of the RL problem. We will not here consider the important problem of how the agent could learn what information it needs to observe.

In the conventional framework, the agent does *not* initially know what effect its actions have on the state of the world, nor what immediate payoffs its actions will produce. It particularly does not know what action is best to do. Rather, it tries out various actions at various states, and gradually learns which one is best at each state so as to maximize its long run payoff. The agent thus acquires what is known as a *closed-loop control policy,* or a rule for choosing an action according to the observed current state of the world.

From an engineering perspective, the most natural way to learn the best actions would be for the agent to try out various actions in various states, and so learn a predictive model of the effects its actions have on the state of the world and of what payoffs its actions produce. Given such a model, the agent could plan ahead by envisioning alternative courses of action and the states and payoffs that would result. However, this approach to learning does not seem at all plausible for animals. Planning ahead involves accurately envisioning alternative sequences of actions and their consequences: the animal would have to imagine what states of the world would result and what payoffs it would receive for different possible sequences of actions. Planning ahead is particularly difficult if actions may have stochastic effects, so that performing an action may lead to one of several different possible states.

One of the most surprising discoveries in RL is that there are simple learning algorithms by means of which an agent can learn an optimal policy without ever being able to predict the effects of its actions or the immediate payoffs they will produce, and without ever planning ahead. It is also surprising that, in principle, learning requires only a minimal amount of episodic memory: an agent can learn if it can consider together the last action it took, the state when it chose that action, the payoff received, and the current state.

The simplest RL algorithms require minimal ability for episodic memory of past states and actions, no prediction of effects of actions, and very simple computations. The speed of learning, and efficiency of use of experience, can be improved if the agent has greater abilities. A continuum of improvement is possible. If an agent has or constructs partial models of the effects of its actions, or if an agent can remember and process longer sequences of past states and actions, learning can be faster and the agent's experience can be used more efficiently.

2.1 Markov Decision Problems

Formally, an RL agent faces a *Markov decision problem* (MDP). An MDP has four components: states, actions, and transition and reward distributions.

The state at time t, for which we use the notation $x(t)$, characterizes the current situation of the agent in the world. For an agent in a maze, for instance, the relevant state would generally be the location of the agent. The action the agent takes at time t is called $a(t)$. Sometimes there may be little or no choice of actions at some state or states. One consequence of taking the action is the immediate payoff or reward which we call $r(x(t), a(t))$, or sometimes just $r(t)$. If the rewards are stochastic, then $\hat{r}(x(t), a(t))$ is its mean. The other consequence of taking the action is that the state changes. Such changes are characterized by a *transition* distribution, which allows them to be stochastic. In the simplest RL problems, state-transition distributions and the rewards depend *only* on the current state and the current action, and not on the history of previous actions and states. This restriction is called the Markov property, and ensures that the description of the current state contains all information relevant to choosing an action in the current state. The Markov property is critically necessary for the learning algorithms that will be described below.

We typically use a set of matrices $P_{xy}(a)$ to describe the transition structure of the world. Here, $P_{xy}(a)$ is the probability that the state changes from x to y given that action a is emitted.

As an example of a Markov decision process, consider a hypothetical experiment in which a rat presses levers to obtain food in a cage with a light. Suppose that if the light is off, pressing lever A turns on the light back on with a certain probability, and pressing lever B has no effect. When the light is on, pressing lever A has no effect, but pressing lever B delivers food with a certain probability, and turns the light off again. In this simple environment there are two relevant states: *light on* and *light off*. Lever A may cause a transition from *light off* to *light on*; in *light on*, lever B may yield a reward. The only information that the rat needs to decide what to do is whether the light is on or off. The optimal policy is simple: in *light off* press lever A; in *light on*, press lever B.

The goal for an agent in solving a Markov decision problem is to maximize its expected, long-term payoff, known as its *return*. A mathematically convenient way to formalize return is as a discounted sum of payoffs. That is, starting from state $x(0) = x$ at time $t = 0$, it should choose actions $a(0), a(1), \ldots$ to maximize

$$\left\langle \sum_{t=0}^{\infty} \gamma^t r(x(t), a(t)) \right\rangle_{x,r} \tag{1}$$

where the $\langle\rangle_{x,r}$ indicates that an average is taken over the stochasticity in the states and the payoffs. The factor $0 \leq \gamma < 1$ is called the *discount factor,* and controls the relative weighting of payoffs that happen sooner and those that happen later. The larger γ, the more important are distant payoffs, and, typically, the more difficult the optimization problem.

Other definitions of return are possible. We assume that the Markov decision problem is *absorbing,* so there is a state or set of states which define the end of a trial (like the goal of the maze), allowing a new trial to start. The infinite sum of equation 1 is effectively truncated at these states, which is equivalent to specifying degenerate transition matrices and reward distributions there. In this case, it is possible to learn to optimize the sum total payoff that the agent receives before reaching the terminal state. If the MDP is such that it is always possible to return to every state, it is possible to learn to optimize the average payoff. A policy that optimizes average payoff will also optimize discounted payoff for a value of γ sufficiently close to 1. RL learning algorithms can be adapted to use any of these definitions of return; we will consider only total discounted reward here. For convenience, we also assume that the problem is finite, *ie* there are only finitely many states and possible actions.

Conventionally, the agent starts off not knowing the transition matrices or reward distributions. Over the course of repeated exploration, has to work out a course of action that will optimize its return.

3 Dynamic Programming

The obvious problem with optimizing a goal such as that in expression 1 is that the action taken at time 0 can affect the rewards at time $t \gg 0$. In effect, one has to take into consideration all possible sequences of actions. However, the number of such sequences of length t typically grows exponentially as a function of t, making it impractical to consider each one in turn. Dynamic programming (Bellman, 1957) offers a set of methods for solving the optimization problem while (generally) avoiding this explosion, by taking advantage of the Markov property. The two main methods are called policy iteration and value iteration, and we discuss them in turn.

3.1 Policy iteration

A *policy* is an assignment of actions to state – *eg* a recipe that says push lever A if the light is off; and push lever B if the light is on. In general, policies can be stochastic, specifying the probability of performing each action at each state. It can be shown that the solution to the optimization problem of expression 1 can be cast in terms of a deterministic policy which is constant over time, so the agent performs the single same action every time it gets to a particular state. Policies without time dependence are called stationary. We will, for the moment, consider stationary deterministic policies. We use the notation $\pi(x)$ for the action that policy π recommends at state x.

In policy iteration, a policy is first *evaluated*, and then *improved*. Policy evaluation consists of working out the value of every state x under policy π, *ie* the the expected long term reward available starting from x and following policy π. That is

$$V^\pi(x) = \left\langle \sum_{t=0}^{\infty} \gamma^t r(t) \right\rangle_{x,r}$$

where the states follow from the transitions with $x(0) = x$ and $r(t) = r(x(t), \pi(x(t)))$. We can separate the first and subsequent terms in the sum,

$$V^\pi(x) = \hat{r}(x, \pi(x)) + \gamma \left\langle \sum_{t=0}^{\infty} \gamma^t r(t+1) \right\rangle_{x,r} \tag{2}$$

which, using the Markov property, is

$$= \hat{r}(x, \pi(x)) + \gamma \sum_y P_{xy}(\pi(x)) V^\pi(y) \ . \tag{3}$$

The second term in both these expressions is just γ times the expected infinite horizon return for the state at time $t = 1$, averaged over all the possibilities for this state. Equation 3 is a linear equation for the values $V^\pi(y)$, and so can be solved by a variety of numerical methods. Later, we see how to find these values without knowing the mean rewards or the transition distributions.

Policy improvement uses the values $V^\pi(x)$ to specify a policy π' that is guaranteed to be at least as good as π. The idea is to consider at state x the non-stationary policy which consists of taking action a (which may, or may not, be the action $\pi(x)$ that policy π recommends at x) and then following policy π thereafter. By the same reasoning as above, the expected value of this is

$$Q^\pi(x, a) = \hat{r}(x, a) + \gamma \sum_y P_{xy}(a) V^\pi(y) \tag{4}$$

The new policy is

$$\pi'(x) = \text{argmax}_a \left\{ Q^\pi(x, a) \right\} \tag{5}$$

from which it is obvious that $Q^\pi(x, \pi'(x)) \geq Q^\pi(x, \pi(x))$. Since the actions at all states are thus only improving, the overall policy can also only improve *simultaneously at every state.* If policy π' is the same as policy π, then it is an optimal policy (of which there may be more than one). The values $V^\pi(x)$ and $Q^\pi(x, a)$ associated with an optimal policy are called the optimal values or the optimal action values and are often written $V^*(x)$ and $Q^*(x, a)$. We will see later how to improve a policy without having explicitly to maximize $Q^\pi(x, a)$.

Since the problem is finite, and policy iteration improves the policy measurably on each step until it finds the optimal policy, policy iteration is bound to converge. Although, in the worst case, policy iteration can take an exponential number of steps, it is generally very quick.

3.2 Value iteration

Value iteration is the main alternative to policy iteration. For one version of this, a set of values $Q(x, a)$ (which are called Q-values) is updated simultaneously according to the formula

$$Q'(x, a) = \hat{r}(x, a) + \gamma \sum_y P_{xy}(a) \max_b Q(y, b) \tag{6}$$

Although the Q-values are not necessarily the Q^π-values associated with any policy π, as in equation 4, one can show that iterating equation 6 infinitely often will lead to the optimal Q values. Then, equation 5 can be used to find the associated optimal policy. The proof that value iteration works depends on a contraction property. That is, a particular measure (called the L_∞ norm) of the distance between $Q(x, a)$ and $Q^*(x, a)$ is greater than that between $Q'(x, a)$ and $Q^*(x, a)$ by a factor of at least $\gamma < 1$. Thus, the values converge exponentially quickly to the optimal values. The optimal policy can actually be derived from them even before convergence.

4 Temporal Differences

Implementing either policy or value iteration requires the agent to know the expected rewards and the transition matrices. If the agent does not know these, but rather just has to learn by interacting with the environment (*eg* by pulling the levers), then what can it do? Methods of temporal differences were invented to perform prediction and optimization in exactly these circumstances. There are two principal flavors of temporal difference methods, an actor-critic scheme (Barto, Sutton & Anderson, 1983), which parallels policy iteration, and has been suggested as being implemented in biological RL, and a method called Q-learning (Watkins, 1989), which parallels value iteration.

Methods of temporal differences were invented (Sutton, 1988; Sutton & Barto, 1987) to account for the behavior of animals in psychological experiments involving prediction; the links with dynamic programming were only made much later (Watkins, 1989; Barto, Sutton & Watkins, 1989).

4.1 Actor-critic learning

In actor-critic learning, the *actor* specifies a policy, which the *critic* evaluates. Consider first a fixed policy or actor π. The critic has to find a set of values $V^\pi(x)$ that satisfy equation 3 based on samples $\{x(t), r(t)\}$ from the transition and reward structure of the world (writing $r(t) = r(x(t), a(t))$ for convenience). It does this by maintaining and improving an approximation $V(x)$ to these quantities.

Two ideas underlie the temporal difference algorithm. One is to use averages from random samples to determine means; the second is to use a form of bootstrapping, substituting the incorrect estimates $V(y)$ as approximations for $V^\pi(y)$ in equation 3. First, consider the quantity

$$r(0) + \gamma V^\pi(x(1)) \tag{7}$$

which comes from the random reward and transition consequent on choosing action $\pi(x)$ at state $x(0) = x$. The *expected* value of this is

$$\Big\langle r(0) + \gamma V^\pi(x(1)) \Big\rangle_{r(0),x(1)} = \hat{r}(x, \pi(x)) + \gamma \sum_y P_{xy}(\pi(x)) V^\pi(y) \tag{8}$$

which is just $Q^\pi(x, \pi(x))$, the quantity on the left hand side of equation 3. Therefore,

$$r(0) + \gamma V^\pi(x(1)) - V(x) \tag{9}$$

could be used as a sampled *error* in the current estimate of the value of state x, namely $V(x)$, and an algorithm such as

$$V(x) \to V(x) + \epsilon \Big[r(0) + \gamma V^\pi(x(1)) - V(x) \Big]$$

when applied over the course of many trials might be expected to make $V(x) \approx V^\pi(x)$. Here, ϵ is a *learning rate.* In fact, this is a form of a stochastic, error-correcting, learning rule like the delta rule (see the DELTA RULE.). The same is true for all subsequent timesteps.

Of course, expression 7 contains $V^\pi(x(1))$, which is exactly the quantity that we are trying to estimate, only at state $x(1)$ rather than $x(0)$. The second key idea underlying temporal difference methods is to substitute the current approximation $V(x(1))$ for this quantity. In this case, the sampled error parallel to equation 9 is

$$\delta(0) = r(0) + \gamma V(x(1)) - V(x) \ , \tag{10}$$

which is called the *temporal difference.* The temporal difference term for subsequent timesteps is defined similarly

$$\delta(t) = r(t) + \gamma V(x(t+1)) - V(x(t)) \ , \tag{11}$$

The learning rule

$$V(x(t)) \to V(x(t)) + \epsilon \delta(t) \tag{12}$$

is called the temporal difference learning rule. It can be considered as a prediction error or a measure of the inconsistency between the estimates $V(x(t))$ and $V(x(t+1))$.

Despite the apparent approximations, circumstances are known under which $V(x) \to V^\pi(x)$ as the number of trials gets large, so implementing policy evaluation correctly. In fact, this is also true in the case that the policy is stochastic, that is $\pi_a(x)$ is a probability distribution over action a associated with state x.

The other half of policy iteration is policy improvement, patterned after equation 5. Here, the idea is to use a stochastic policy (rather than the deterministic ones that we have hitherto considered), which is defined by a set of parameters, called the action matrix M_{ax}. Then, rather than implement the full maximization of equation 5, we consider changing the parameters, using the answer from policy evaluation in order to increase the probability of actions that are associated with higher values of $Q^\pi(x, a)$. A natural way to parameterize a stochastic policy is to use a softmax function:

$$\pi_a(x) = \frac{\exp(M_{ax})}{\sum_b \exp(M_{bx})}$$

The normalization forces $\sum_a \pi_a(x) = 1$ for all states x. Here, the larger M_{ax} compared with M_{bx}, the more likely is action a at state x.

Consider the temporal difference δ after convergence, *ie* when $V(x) = V^\pi(x)$. In this case,

$$V(x) = V^\pi(x) = \sum_a \pi_a(x) Q^\pi(x, a)$$

is the average value of the actions specified by policy π. If action a is better than average, then $Q^\pi(x, a) > V(x)$; if action a is worse than average, then $Q^\pi(x, a) < V(x)$. Thus, π could be improved by changing the action matrix according to

$$M_{ax} \to M_{ax} + \epsilon \Big[(Q^\pi(x, a) - V(x) \Big] \tag{13}$$

However, if the agent takes action $a(0) = a$ at state $x(0) = x$, then

$$\delta(0) = Q^\pi(x, a) - V(x)$$

and so the actor learning rule is

$$M_{a(0)x(0)} \to M_{a(0)x(0)} + \epsilon\delta(0) \tag{14}$$

using just the same temporal difference term as in equation 12.

Normally, the action values are changed according to equation 14 before the critic has converged, and there is no proof that the combined estimation and optimization procedure is guaranteed to find an optimal policy. Nevertheless, it is found to work well in practice. Note that learning is based on following, and simultaneously trying to improve a policy.

4.2 Q-learning

Q-learning is a temporal difference version of value iteration. It applies to equation 6 the same two ideas underlying the actor-critic. That is, it maintains and improves values $Q(x, a)$, employing

$$r(0) + \gamma \max_b Q(x(1), b)$$

as a sampled version of the right hand side of equation 6. The overall update at time t is

$$Q(x(t), a(t)) \to Q(x(t), a(t)) + \epsilon \left[r(t) + \gamma \max_b Q(x(t+1), b) - Q(x(t), a(t)) \right]$$

As in value iteration, a policy can be defined at any time according to

$$\pi(x) = \text{argmax}_a Q(x, a).$$

Unlike the actor-critic scheme, circumstances are known under which Q-learning is guaranteed to converge on the optimal policy.

4.3 Exploration and Exploitation

One of the most important issues for the temporal difference learning algorithms is maintaining a balance between exploration and exploitation. That is, the agent must sometimes choose actions that it believes to be suboptimal in order to find out whether they might actually be good. This is particularly true in problems which change over time (which is true of most behavioral experiments), since actions that used to be good might become bad, and vice-versa.

The theorems proving that temporal difference methods work usually require much experimentation: all actions must be repeatedly tried in all states. In practice, it is common to choose policies that always embody exploration (such as choosing a random action some small fraction of the time, but otherwise the action currently believed to be best).

5 Extensions to Temporal Difference Learning

We presented a very simple form of temporal difference learning algorithm, but it has been extended in various ways. First, it is possible to learn the transition matrices and rewards and then use standard dynamic programming methods on the resulting estimated model of the environment (this is called a model-based method). It is also possible to integrate the learning of a model with the temporal difference learning methods, as a way of using the samples of transitions and rewards more effectively. There is substantial evidence that animals build models of their environments, although the way that they use this information to learn appropriate behaviors is not so clear.

Second, the idea behind equation 7 is to use $V^\pi(x(1))$ as a stochastic sample of $\sum_y P_{xy}(\pi(x))V^\pi(y)$ to replace all but the first reward term in the infinite sum of equation 2. One could equally well imagine collecting two actual steps of reward $r(0) + \gamma r(1)$, and using $V^\pi(x(2))$ as a sample of the all but the first *two* terms in equation 2. Similarly, one could consider all but the first three terms, *etc.* It turns out to be simple to weigh these different contributions in an exponentially decreasing manner, and this leads to an algorithm called TD(λ), where the value of λ is the weighting, taking the value $\lambda = 0$ for equation 7. The remaining bootstrap approximation is the same as before. The significance of λ is that it controls a trade-off between bias and variance. If λ is near 1, then the estimate is highly variable (since it depends on long sample paths), but not strongly biased (since real rewards from the reward distribution of the environment are considered). If λ is near 0, then the estimate is not very variable (since only short sample paths are involved), but it can be biased, because of the bootstrapping.

Third, we described the temporal difference algorithm using a unary or *table-lookup* representation in which we can separately manipulate the value $V(x)$ associated with each state $x(t)$. One can also consider parameterized representations of the values (and also the policies), for instance the linear form

$$V(x) = \mathbf{w} \cdot \boldsymbol{\phi}(x) \tag{15}$$

where $\mathbf{w}$ is a set of parameters, and $\boldsymbol{\phi}(x)$ is a population representation for state x. In this case, the temporal difference update rule of equation 12 is changed to one for the parameters

$$\mathbf{w} \to \mathbf{w} + \epsilon\delta(t)\boldsymbol{\phi}(x(t)) \ .$$

Again, this is quite similar to the delta rule. More sophisticated representational schemes than equation 15 can also be used, including neural networks, in which case $\nabla_{\mathbf{w}} V(x(t))$ should be used in place of $\boldsymbol{\phi}(x(t))$.

6 Formalizing learning in the brain

We have stressed that animals and artificial systems face similar problems in learning how to optimizing their behavior in the light of rewards and punishments. Although we have described temporal difference methods from the perspective of the engineering method of dynamic programming, they are also interesting as a way of formalizing behavioral and neurobiological experiments on animals. Briefly (see REINFORCEMENT LEARNING IN THE BRAIN), it has been postulated that a prediction error signal for appetitive events with some properties like that of the temporal difference signal $\delta(t)$ (equation 11) seems to be broadcast by dopaminergic cells in the ventral tegmental area, to control the learning of predictions, and by dopaminergic cells in the substantia nigra to control the learning of actions. The model is, as yet, quite incomplete, failing, for example, to specify the interaction between attention and learning, which is critical for accounting for the results of behavioral experiments.

References

Barto, AG, Sutton, RS & Anderson, CW (1983) Neuronlike elements that can solve difficult learning problems. *IEEE Transactions on Systems, Man, and Cybernetics* **13**:834-846.

Bellman, RE (1957) *Dynamic Programming*. Princeton, NJ:Princeton University Press.

Sutton, RS (1988) Learning to predict by the methods of temporal difference. *Machine Learning* **3**:9-44.

Sutton, RS & Barto, AG (1987) A temporal-difference model of classical conditioning. *Proceedings of the Ninth Annual Conference of the Cognitive Science Society*. Seattle, WA:???

Watkins, CJCH (1989) *Learning from Delayed Rewards*. PhD Thesis, University of Cambridge, Cambridge, UK.

Further Reading

Barto, AG, Sutton, RS & Watkins, CJCH (1990) Learning and sequential decision making. In M Gabriel & J Moore, editors, *Learning and Computational Neuroscience: Foundations of Adaptive Networks*. Cambridge, MA:MIT Press, Bradford Books.

Bertsekas, DP & Tsitsiklis, JN (1996) *Neuro-Dynamic Programming*. Belmont, MA:Athena Scientific.

Sutton, RS & Barto, AG (1998). *Reinforcement Learning: An Introduction.* Cambridge, MA: MIT Press.