A *synapse* is a structure generally at the termination of an axon branch that mediates the communication of one neuron to another. A synapse transmits information from the *presynaptic* neuron's axon to a dendrite or cell body of the *postsynaptic* neuron. With a few exceptions, synapses release a chemical *neurotransmitter* upon the arrival of an action potential from the presynaptic neuron. (The exceptions are cases of direct electric coupling between neurons, but these will not concern us here.) Neurotransmitter molecules released from the presynaptic side of the synapse diffuse across the *synaptic cleft*, the very small space between the presynaptic ending and the postsynaptic neuron, and then bind to receptors on the surface of the postsynaptic neuron to excite or inhibit its spike-generating activity, or to modulate its behavior in other ways. A particular neurotransmitter may bind to several different types of receptors, with each producing a different effect on the postsynaptic neuron. For example, there are at least five different receptor types by which the neurotransmitter dopamine can affect a postsynaptic neuron. Many different chemicals have been identified as neurotransmitters in animal nervous systems.

A neuron's *background* activity is its level of activity, usually its firing rate, when the neuron does not appear to be driven by synaptic input related to the task of interest to the experimenter, for example, when the neuron's activity is not correlated with a stimulus delivered to a subject as part of an experiment. Background activity can be irregular due to input from the wider network, or due to noise within the neuron or its synapses. Sometimes background activity is the result of dynamic processes intrinsic to the neuron. A neuron's *phasic* activity, in contrast to its background activity, consists of bursts of spiking activity usually caused by synaptic input. Activity that varies slowly and often in a graded manner, whether as background activity or not, is called a neuron's *tonic* activity.

The strength or effectiveness by which the neurotransmitter released at a synapse influences the postsynaptic neuron is the synapse's *efficacy*. One way a nervous system can change through experience is through changes in synaptic efficacies as a result of combinations of the activities of the presynaptic and postsynaptic neurons, and sometimes by the presence of a *neuromodulator*, which is a neurotransmitter having effects other than, or in addition to, direct fast excitation or inhibition.

Brains contain several different neuromodulation systems consisting of clusters of neurons with widely branching axonal arbors, with each system using a different neurotransmitter. Neuromodulation can alter the function of neural circuits, mediate motivation, arousal, attention, memory, mood, emotion, sleep, and body temperature. Important here is that a neuromodulatory system can distribute something like a scalar signal, such as a reinforcement signal, to alter the operation of synapses in widely distributed sites critical for learning.

The ability of synaptic efficacies to change is called *synaptic plasticity*. It is one of the primary mechanisms responsible for learning. The parameters, or weights, adjusted by learning algorithms correspond to synaptic efficacies. As we detail below, modulation of synaptic plasticity via the neuromodulator dopamine is a plausible mechanism for how the brain might implement learning algorithms like many of those described in this book.

15.2 Reward Signals, Reinforcement Signals, Values, and Prediction Errors

Links between neuroscience and computational reinforcement learning begin as parallels between signals in the brain and signals playing prominent roles in reinforcement learning theory and algorithms. In Chapter 3 we said that any problem of learning goal-directed behavior can be reduced to the three signals representing actions, states, and rewards. However, to explain links that have been made between neuroscience and reinforcement learning, we have to be less abstract than this and consider other reinforcement learning signals that correspond, in certain ways, to signals in the brain. In addition to reward signals, these include reinforcement signals (which we argue are different from reward signals), value signals, and signals conveying prediction errors. When we label a signal by its function in this way, we are doing it in the context of reinforcement learning theory in which the signal corresponds to a term in an equation or an algorithm. On the other hand, when we refer to a signal in the brain, we mean a physiological event such as a burst of action potentials or the secretion of a neurotransmitter. Labeling a neural signal by its function, for example calling the phasic activity of a dopamine neuron a reinforcement signal, means that the neural signal behaves like, and is conjectured to function like, the corresponding theoretical signal.

Uncovering evidence for these correspondences involves many challenges. Neural activity related to reward processing can be found in nearly every part of the brain, and it is difficult to interpret results unambiguously because representations of different reward-related signals tend to be highly correlated with one another. Experiments need to be carefully designed to allow one type of reward-related signal to be distinguished with any degree of certainty from others—or from an abundance of other signals not related to reward processing. Despite these difficulties, many experiments have been conducted with the aim of reconciling aspects of reinforcement learning theory and algorithms with neural signals, and some compelling links have been established. To prepare for examining these links, in the rest of this section we remind the reader of what various reward-related signals mean according to reinforcement learning theory.

In our Comments on Terminology at the end of the previous chapter, we said that R_t is like a reward signal in an animal's brain and not an object or event in the animal's environment. In reinforcement learning, the reward signal (along with an agent's environment) defines the problem a reinforcement learning agent is trying to solve. In this respect, R_t is like a signal in an animal's brain that distributes primary reward to sites throughout the brain. But it is unlikely that a unitary master reward signal like R_t exists in an animal's brain. It is best to think of R_t as an abstraction summarizing the overall effect of a multitude of neural signals generated by many systems in the brain that assess the rewarding or punishing qualities of sensations and states.

Reinforcement signals in reinforcement learning are different from reward signals. The function of a reinforcement signal is to direct the changes a learning algorithm makes in an agent's policy, value estimates, or environment models. For a TD method, for instance, the reinforcement signal at time t is the TD error $\delta_{t-1} = R_t + \gamma V(S_t) - V(S_{t-1})$.[1] The

[1] As we mentioned in Section 6.1, δ_t in our notation is defined to be $R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$, so δ_t

reinforcement signal for some algorithms could be just the reward signal, but for most of the algorithms we consider the reinforcement signal is the reward signal adjusted by other information, such as the value estimates in TD errors.

Estimates of state values or of action values, that is, V or Q, specify what is good or bad for the agent over the long run. They are predictions of the total reward an agent can expect to accumulate over the future. Agents make good decisions by selecting actions leading to states with the largest estimated state values, or by selecting actions with the largest estimated action values.

Prediction errors measure discrepancies between expected and actual signals or sensations. Reward prediction errors (RPEs) specifically measure discrepancies between the expected and the received reward signal, being positive when the reward signal is greater than expected, and negative otherwise. TD errors like (6.5) are special kinds of RPEs that signal discrepancies between current and earlier expectations of reward over the long-term. When neuroscientists refer to RPEs they generally (though not always) mean TD RPEs, which we simply call TD errors throughout this chapter. Also in this chapter, a TD error is generally one that does not depend on actions, as opposed to TD errors used in learning action-values by algorithms like Sarsa and Q-learning. This is because the most well-known links to neuroscience are stated in terms of action-free TD errors, but we do not mean to rule out possible similar links involving action-dependent TD errors. (TD errors for predicting signals other than rewards are useful too, but that case will not concern us here. See, for example, Modayil, White, and Sutton, 2014.)

One can ask many questions about links between neuroscience data and these theoretically-defined signals. Is an observed signal more like a reward signal, a value signal, a prediction error, a reinforcement signal, or something altogether different? And if it is an error signal, is it an RPE, a TD error, or a simpler error like the Rescorla–Wagner error (14.3)? And if it is a TD error, does it depend on actions like the TD error of Q-learning or Sarsa? As indicated above, probing the brain to answer questions like these is extremely difficult. But experimental evidence suggests that one neurotransmitter, specifically the neurotransmitter dopamine, signals RPEs, and further, that the phasic activity of dopamine-producing neurons in fact conveys TD errors (see Section 15.1 for a definition of phasic activity). This evidence led to the *reward prediction error hypothesis of dopamine neuron activity*, which we describe next.

15.3 The Reward Prediction Error Hypothesis

The *reward prediction error hypothesis of dopamine neuron activity* proposes that one of the functions of the phasic activity of dopamine-producing neurons in mammals is to deliver an error between an old and a new estimate of expected future reward to target areas throughout the brain. This hypothesis (though not in these exact words) was first explicitly stated by Montague, Dayan, and Sejnowski (1996), who showed how the TD error concept from reinforcement learning accounts for many features of the phasic

is not available until time $t+1$. The TD error *available* at t is actually $\delta_{t-1} = R_t + \gamma V(S_t) - V(S_{t-1})$. Because we are thinking of time steps as very small, or even infinitesimal, time intervals, one should not attribute undue importance to this one-step time shift.

activity of dopamine neurons in mammals. The experiments that led to this hypothesis were performed in the 1980s and early 1990s in the laboratory of neuroscientist Wolfram Schultz. Section 15.5 describes these influential experiments, Section 15.6 explains how the results of these experiments align with TD errors, and the Bibliographical and Historical Remarks section at the end of this chapter includes a guide to the literature surrounding the development of this influential hypothesis.

Montague et al. (1996) compared the TD errors of the TD model of classical conditioning with the phasic activity of dopamine-producing neurons during classical conditioning experiments. Recall from Section 14.2 that the TD model of classical conditioning is basically the semi-gradient-descent TD(λ) algorithm with linear function approximation. Montague et al. made several assumptions to set up this comparison. First, because a TD error can be negative but neurons cannot have a negative firing rate, they assumed that the quantity corresponding to dopamine neuron activity is $\delta_{t-1} + b_t$, where b_t is the background firing rate of the neuron. A negative TD error corresponds to a drop in a dopamine neuron's firing rate below its background rate.[2]

A second assumption was needed about the states visited in each classical conditioning trial and how they are represented as inputs to the learning algorithm. This is the same issue we discussed in Section 14.2.4 for the TD model. Montague et al. chose a complete serial compound (CSC) representation as shown in the left column of Figure 14.1, but where the sequence of short-duration internal signals continues until the onset of the US, which here is the arrival of a non-zero reward signal. This representation allows the TD error to mimic the fact that dopamine neuron activity not only predicts a future reward, but that it is also sensitive to *when* after a predictive cue that reward is expected to arrive. There has to be some way to keep track of the time between sensory cues and the arrival of reward. If a stimulus initiates a sequence of internal signals that continues after the stimulus ends, and if there is a different signal for each time step following the stimulus, then each time step after the stimulus is represented by a distinct state. Thus, the TD error, being state-dependent, can be sensitive to the timing of events within a trial.

In simulated trials with these assumptions about background firing rate and input representation, TD errors of the TD model are remarkably similar to dopamine neuron phasic activity. Previewing our description of details about these similarities in Section 15.5 below, the TD errors parallel the following features of dopamine neuron activity: (1) the phasic response of a dopamine neuron only occurs when a rewarding event is unpredicted; (2) early in learning, neutral cues that precede a reward do not cause substantial phasic dopamine responses, but with continued learning these cues gain predictive value and come to elicit phasic dopamine responses; (3) if an even earlier cue reliably precedes a cue that has already acquired predictive value, the phasic dopamine response shifts to the earlier cue, ceasing for the later cue; and (4) if after learning, the predicted rewarding event is omitted, a dopamine neuron's response decreases below its baseline level shortly after the expected time of the rewarding event.

[2] In the literature relating TD errors to the activity of dopamine neurons, their δ_t is the same as our $\delta_{t-1} = R_t + \gamma V(S_t) - V(S_{t-1})$.

Although not every dopamine neuron monitored in the experiments of Schultz and colleagues behaved in all of these ways, the striking correspondence between the activities of most of the monitored neurons and TD errors lends strong support to the reward prediction error hypothesis. There are situations, however, in which predictions based on the hypothesis do not match what is observed in experiments. The choice of input representation is critical to how closely TD errors match some of the details of dopamine neuron activity, particularly details about the timing of dopamine neuron responses. Different ideas, some of which we discuss below, have been proposed about input representations and other features of TD learning to make the TD errors fit the data better, though the main parallels appear with the CSC representation that Montague et al. used. Overall, the reward prediction error hypothesis has received wide acceptance among neuroscientists studying reward-based learning, and it has proven to be remarkably resilient in the face of accumulating results from neuroscience experiments.

To prepare for our description of the neuroscience experiments supporting the reward prediction error hypothesis, and to provide some context so that the significance of the hypothesis can be appreciated, we next present some of what is known about dopamine, the brain structures it influences, and how it is involved in reward-based learning.

15.4 Dopamine

Dopamine is produced as a neurotransmitter by neurons whose cell bodies lie mainly in two clusters of neurons in the midbrain of mammals: the substantia nigra pars compacta (SNpc) and the ventral tegmental area (VTA). Dopamine plays essential roles in many processes in the mammalian brain. Prominent among these are motivation, learning, action-selection, most forms of addiction, and the disorders schizophrenia and Parkinson's disease. Dopamine is called a neuromodulator because it performs many functions other than direct fast excitation or inhibition of targeted neurons. Although much remains unknown about dopamine's functions and details of its cellular effects, it is clear that it is fundamental to reward processing in the mammalian brain. Dopamine is not the only neuromodulator involved in reward processing, and its role in aversive situations—punishment—remains controversial. Dopamine also can function differently in non-mammals. But no one doubts that dopamine is essential for reward-related processes in mammals, including humans.

An early, traditional view is that dopamine neurons broadcast a reward signal to multiple brain regions implicated in learning and motivation. This view followed from a famous 1954 paper by James Olds and Peter Milner that described the effects of electrical stimulation on certain areas of a rat's brain. They found that electrical stimulation to particular regions acted as a very powerful reward in controlling the rat's behavior: "...the control exercised over the animal's behavior by means of this reward is extreme, possibly exceeding that exercised by any other reward previously used in animal experimentation" (Olds and Milner, 1954). Later research revealed that the sites at which stimulation was most effective in producing this rewarding effect excited dopamine pathways, either directly or indirectly, that ordinarily are excited by natural rewarding stimuli. Effects similar to these were also observed with human subjects. These observations strongly suggested that dopamine neuron activity signals reward.

But if the reward prediction error hypothesis is correct—even if it accounts for only some features of a dopamine neuron's activity—this traditional view of dopamine neuron activity is not entirely correct: phasic responses of dopamine neurons signal reward prediction errors, not reward itself. In reinforcement learning's terms, a dopamine neuron's phasic response at a time t corresponds to $\delta_{t-1} = R_t + \gamma V(S_t) - V(S_{t-1})$, not to R_t.

Reinforcement learning theory and algorithms help reconcile the reward-prediction-error view with the conventional notion that dopamine signals reward. In many of the algorithms we discuss in this book, δ functions as a reinforcement signal, meaning that it is the main driver of learning. For example, δ is the critical factor in the TD model of classical conditioning, and δ is the reinforcement signal for learning both a value function and a policy in an actor–critic architecture (Sections 13.5 and 15.7). Action-dependent forms of δ are reinforcement signals for Q-learning and Sarsa. The reward signal R_t is a crucial component of δ_{t-1}, but it is not the complete determinant of its reinforcing effect in these algorithms. The additional term $\gamma V(S_t) - V(S_{t-1})$ is the higher-order reinforcement part of δ_{t-1}, and even if reward occurs ($R_t \neq 0$), the TD error can be silent if the reward is fully predicted (which is fully explained in Section 15.6 below).

A closer look at Olds' and Milner's 1954 paper, in fact, reveals that it is mainly about the reinforcing effect of electrical stimulation in an instrumental conditioning task. Electrical stimulation not only energized the rats' behavior—through dopamine's effect on motivation—it also led to the rats quickly learning to stimulate themselves by pressing a lever, which they would do frequently for long periods of time. The activity of dopamine neurons triggered by electrical stimulation reinforced the rats' lever pressing.

More recent experiments using optogenetic methods clinch the role of phasic responses of dopamine neurons as reinforcement signals. These methods allow neuroscientists to precisely control the activity of selected neuron types at a millisecond timescale in awake behaving animals. Optogenetic methods introduce light-sensitive proteins into selected neuron types so that these neurons can be activated or silenced by means of flashes of laser light. The first experiment using optogenetic methods to study dopamine neurons showed that optogenetic stimulation producing phasic activation of dopamine neurons in mice was enough to condition the mice to prefer the side of a chamber where they received this stimulation as compared to the chamber's other side where they received no, or lower-frequency, stimulation (Tsai et al. 2009). In another example, Steinberg et al. (2013) used optogenetic activation of dopamine neurons to create artificial bursts of dopamine neuron activity in rats at the times when rewarding stimuli were expected but omitted—times when dopamine neuron activity normally pauses. With these pauses replaced by artificial bursts, responding was sustained when it would ordinarily decrease due to lack of reinforcement (in extinction trials), and learning was enabled when it would ordinarily be blocked due to the reward being already predicted (the blocking paradigm; Section 14.2.1).

Additional evidence for the reinforcing function of dopamine comes from optogenetic experiments with fruit flies, except in these animals dopamine's effect is the opposite of its effect in mammals: optically triggered bursts of dopamine neuron activity act just like electric foot shock in reinforcing avoidance behavior, at least for the population

of dopamine neurons activated (Claridge-Chang et al. 2009). Although none of these optogenetic experiments showed that phasic dopamine neuron activity is specifically like a TD error, they convincingly demonstrated that phasic dopamine neuron activity acts just like δ acts (or perhaps like *minus* δ acts in fruit flies) as the reinforcement signal in algorithms for both prediction (classical conditioning) and control (instrumental conditioning).

Dopamine neurons are particularly well suited to broadcasting a reinforcement signal to many areas of the brain. These neurons have huge axonal arbors, each releasing dopamine at 100 to 1,000 times more synaptic sites than reached by the axons of typical neurons. Shown to the right is the axonal arbor of a single dopamine neuron whose cell body is in the SNpc of a rat's brain. Each axon of a SNpc or VTA dopamine neuron makes roughly 500,000 synaptic contacts on the dendrites of neurons in targeted brain areas.

Axonal arbor of a single neuron producing dopamine as a neurotransmitter. These axons make synaptic contacts with a huge number of dendrites of neurons in targeted brain areas.

Adapted from *The Journal of Neuroscience*, Matsuda, Furuta, Nakamura, Hioki, Fujiyama, Arai, and Kaneko, volume 29, 2009, page 451.

If dopamine neurons broadcast a reinforcement signal like reinforcement learning's δ, then because this is a scalar signal, i.e., a single number, all dopamine neurons in both the SNpc and VTA would be expected to activate more-or-less identically so that they would act in near synchrony to send the same signal to all of the sites their axons target. Although it has been a common belief that dopamine neurons do act together like this, modern evidence is pointing to the more complicated picture that different subpopulations of dopamine neurons respond to input differently depending on the structures to which they send their signals and the different ways these signals act on their target structures. Dopamine has functions other than signaling RPEs, and even for dopamine neurons that do signal RPEs, it can make sense to send different RPEs to different structures depending on the roles these structures play in producing reinforced behavior. This is beyond what we treat in any detail in this book, but vector-valued RPE signals make sense from the perspective of reinforcement learning when decisions can be decomposed into separate sub-decisions, or more generally, as a way to address the *structural* version of the credit assignment problem: How do you distribute credit for success (or blame for failure) of a decision among the many component structures that could have been involved in producing it? We say a bit more about this in Section 15.10 below.

The axons of most dopamine neurons make synaptic contact with neurons in the frontal cortex and the basal ganglia, areas of the brain involved in voluntary movement, decision making, learning, and cognitive functions such as planning. Because most ideas relating

dopamine to reinforcement learning focus on the basal ganglia, and the connections from dopamine neurons are particularly dense there, we focus on the basal ganglia here. The basal ganglia are a collection of neuron groups, or nuclei, lying at the base of the forebrain. The main input structure of the basal ganglia is called the striatum. Essentially all of the cerebral cortex, among other structures, provides input to the striatum. The activity of cortical neurons conveys a wealth of information about sensory input, internal states, and motor activity. The axons of cortical neurons make synaptic contacts on the dendrites of the main input/output neurons of the striatum, called medium spiny neurons. Output from the striatum loops back via other basal ganglia nuclei and the thalamus to frontal areas of cortex, and to motor areas, making it possible for the striatum to influence movement, abstract decision processes, and reward processing. Two main subdivisions of the striatum are important for reinforcement learning: the dorsal striatum, primarily implicated in influencing action selection, and the ventral striatum, thought to be critical for different aspects of reward processing, including the assignment of affective value to sensations.

The dendrites of medium spiny neurons are covered with spines on whose tips the axons of neurons in the cortex make synaptic contact. Also making synaptic contact with these spines—in this case contacting the spine stems—are axons of dopamine neurons (Figure 15.1). This arrangement brings together presynaptic activity of cortical neurons,

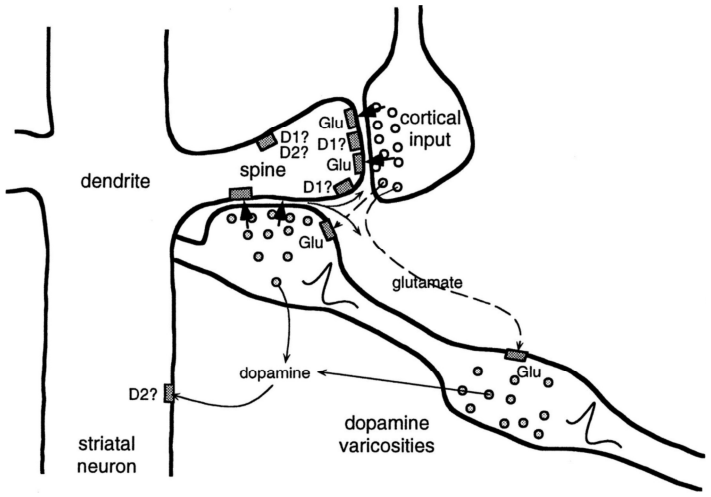


Figure 15.1: Spine of a striatal neuron showing input from both cortical and dopamine neurons. Axons of cortical neurons influence striatal neurons via corticostriatal synapses releasing the neurotransmitter glutamate at the tips of spines covering the dendrites of striatal neurons. An axon of a VTA or SNpc dopamine neuron is shown passing by the spine (from the lower right). "Dopamine varicosities" on this axon release dopamine at or near the spine stem, in an arrangement that brings together presynaptic input from cortex, postsynaptic activity of the striatal neuron, and dopamine, making it possible that several types of learning rules govern the plasticity of corticostriatal synapses. Each axon of a dopamine neuron makes synaptic contact with the stems of roughly 500,000 spines. Some of the complexity omitted from our discussion is shown here by other neurotransmitter pathways and multiple receptor types, such as D1 an D2 dopamine receptors by which dopamine can produce different effects at spines and other postsynaptic sites. From *Journal of Neurophysiology*, W. Schultz, vol. 80, 1998, page 10.

postsynaptic activity of medium spiny neurons, and input from dopamine neurons. What actually occurs at these spines is complex and not completely understood. Figure 15.1 hints at the complexity by showing two types of receptors for dopamine, receptors for glutamate—the neurotransmitter of the cortical inputs—and multiple ways that the various signals can interact. But evidence is mounting that changes in the efficacies of the synapses on the pathway from the cortex to the striatum, which neuroscientists call *corticostriatal synapses*, depend critically on appropriately-timed dopamine signals.

15.5 Experimental Support for the Reward Prediction Error Hypothesis

Dopamine neurons respond with bursts of activity to intense, novel, or unexpected visual and auditory stimuli that trigger eye and body movements, but very little of their activity is related to the movements themselves. This is surprising because degeneration of dopamine neurons is a cause of Parkinson's disease, whose symptoms include motor disorders, particularly deficits in self-initiated movement. Motivated by the weak relationship between dopamine neuron activity and stimulus-triggered eye and body movements, Romo and Schultz (1990) and Schultz and Romo (1990) took the first steps toward the reward prediction error hypothesis by recording the activity of dopamine neurons and muscle activity while monkeys moved their arms.

They trained two monkeys to reach from a resting hand position into a bin containing a bit of apple, a piece of cookie, or a raisin, when the monkey saw and heard the bin's door open. The monkey could then grab and bring the food to its mouth. After a monkey became good at this, it was trained on two additional tasks. The purpose of the first task was to see what dopamine neurons do when movements are self-initiated. The bin was left open but covered from above so that the monkey could not see inside but could reach in from below. No triggering stimuli were presented, and after the monkey reached for and ate the food morsel, the experimenter usually (though not always), silently and unseen by the monkey, replaced food in the bin by sticking it onto a rigid wire. Here too, the activity of the dopamine neurons Romo and Schultz monitored was not related to the monkey's movements, but a large percentage of these neurons produced phasic responses whenever the monkey first touched a food morsel. These neurons did not respond when the monkey touched just the wire or explored the bin when no food was there. This was good evidence that the neurons were responding to the food and not to other aspects of the task.

The purpose of Romo and Schultz's second task was to see what happens when movements are triggered by stimuli. This task used a different bin with a movable cover. The sight and sound of the bin opening triggered reaching movements to the bin. In this case, Romo and Schultz found that after some period of training, the dopamine neurons no longer responded to the touch of the food but instead responded to the sight and sound of the opening cover of the food bin. The phasic responses of these neurons had shifted from the reward itself to stimuli predicting the availability of the reward. In a followup study, Romo and Schultz found that most of the dopamine neurons whose activity they

monitored did not respond to the sight and sound of the bin opening outside the context of the behavioral task. These observations suggested that the dopamine neurons were responding neither to the initiation of a movement nor to the sensory properties of the stimuli, but were rather signaling an expectation of reward.

Schultz's group conducted many additional studies involving both SNpc and VTA dopamine neurons. A particular series of experiments was influential in suggesting that the phasic responses of dopamine neurons correspond to TD errors and not to simpler errors like those in the Rescorla–Wagner model (14.3). In the first of these experiments (Ljungberg, Apicella, and Schultz, 1992), monkeys were trained to depress a lever after a light was illuminated as a 'trigger cue' to obtain a drop of apple juice. As Romo and Schultz had observed earlier, many dopamine neurons initially responded to the reward—the drop of juice (Figure 15.2, top panel). But many of these neurons lost that reward response as training continued and developed responses instead to the illumination of the light that predicted the reward (Figure 15.2, middle panel). With continued training, lever pressing became faster while the number of dopamine neurons responding to the trigger cue decreased.

Following this study, the same monkeys were trained on a new task (Schultz, Apicella, and Ljungberg, 1993). Here the monkeys faced two levers, each with a light above it. Illuminating one of these lights was an 'instruction cue' indicating which of the two levers

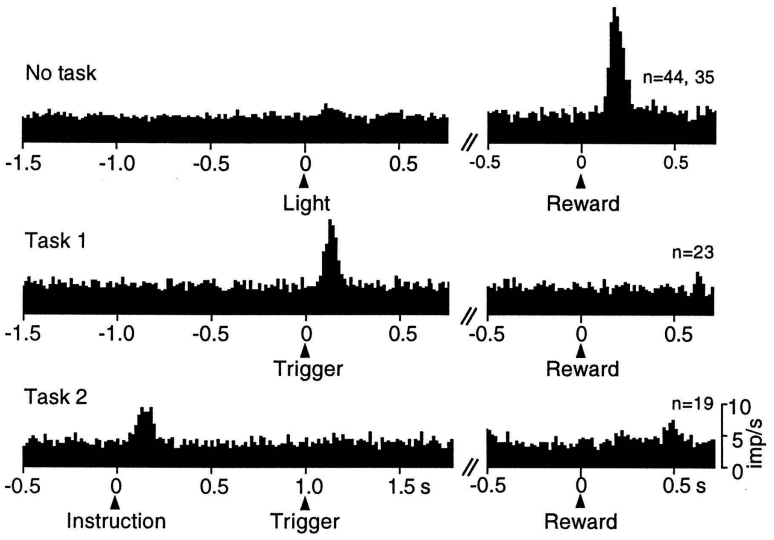


Figure 15.2: The response of dopamine neurons shifts from initial responses to primary reward to earlier predictive stimuli. These are plots of the number of action potentials produced by monitored dopamine neurons within small time intervals, averaged over all the monitored dopamine neurons (ranging from 23 to 44 neurons for these data). Top: dopamine neurons are activated by the unpredicted delivery of drop of apple juice. Middle: with learning, dopamine neurons developed responses to the reward-predicting trigger cue and lost responsiveness to the delivery of reward. Bottom: with the addition of an instruction cue preceding the trigger cue by 1 second, dopamine neurons shifted their responses from the trigger cue to the earlier instruction cue. From Schultz et al. (1995), MIT Press.

would produce a drop of apple juice. In this task, the instruction cue preceded the trigger cue of the previous task by a fixed interval of 1 second. The monkeys learned to withhold reaching until seeing the trigger cue, and dopamine neuron activity increased, but now the responses of the monitored dopamine neurons occurred almost exclusively to the earlier instruction cue and not to the trigger cue (Figure 15.2, bottom panel). Here again the number of dopamine neurons responding to the instruction cue was much reduced when the task was well learned. During learning across these tasks, dopamine neuron activity shifted from initially responding to the reward to responding to the earlier predictive stimuli, first progressing to the trigger stimulus then to the still earlier instruction cue. As responding moved earlier in time it disappeared from the later stimuli. This shifting of responses to earlier reward predictors, while losing responses to later predictors is a hallmark of TD learning (see, for example, Figure 14.2).

The task just described revealed another property of dopamine neuron activity shared with TD learning. The monkeys sometimes pressed the wrong key, that is, the key other than the instructed one, and consequently received no reward. In these trials, many of the dopamine neurons showed a sharp decrease in their firing rates below baseline shortly after the reward's usual time of delivery, and this happened without the availability of any external cue to mark the usual time of reward delivery (Figure 15.3). Somehow the

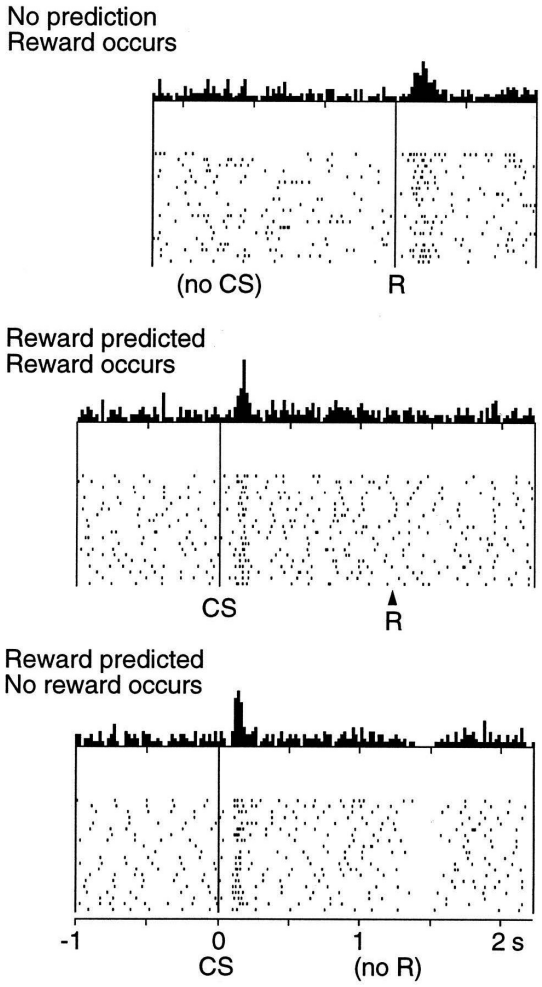


Figure 15.3: The response of dopamine neurons drops below baseline shortly after the time when an expected reward fails to occur. Top: dopamine neurons are activated by the unpredicted delivery of a drop of apple juice. Middle: dopamine neurons respond to a conditioned stimulus (CS) that predicts reward and do not respond to the reward itself. Bottom: when the reward predicted by the CS fails to occur, the activity of dopamine neurons drops below baseline shortly after the time the reward is expected to occur. At the top of each of these panels is shown the average number of action potentials produced by monitored dopamine neurons within small time intervals around the indicated times. The raster plots below show the activity patterns of the individual dopamine neurons that were monitored; each dot represents an action potential. From Schultz, Dayan, and Montague, A Neural Substrate of Prediction and Reward, *Science*, vol. 275, issue 5306, pages 1593-1598, March 14, 1997. Reprinted with permission from AAAS.

monkeys were internally keeping track of the timing of the reward. (Response timing is one area where the simplest version of TD learning needs to be modified to account for some of the details of the timing of dopamine neuron responses. We consider this issue in the following section.)

The observations from the studies described above led Schultz and his group to conclude that dopamine neurons respond to unpredicted rewards, to the earliest predictors of reward, and that dopamine neuron activity decreases below baseline if a reward, or a predictor of reward, does not occur at its expected time. Researchers familiar with reinforcement learning were quick to recognize that these results are strikingly similar to how the TD error behaves as the reinforcement signal in a TD algorithm. The next section explores this similarity by working through a specific example in detail.

15.6 TD Error/Dopamine Correspondence

This section explains the correspondence between the TD error δ and the phasic responses of dopamine neurons observed in the experiments just described. We examine how δ changes over the course of learning in a task something like the one described above where a monkey first sees an instruction cue and then a fixed time later has to respond correctly to a trigger cue in order to obtain reward. We use a simple idealized version of this task, but we go into a lot more detail than is usual because we want to emphasize the theoretical basis of the parallel between TD errors and dopamine neuron activity.

The first simplifying assumption is that the agent has already learned the actions required to obtain reward. Then its task is just to learn accurate predictions of future reward for the sequence of states it experiences. This is then a prediction task, or more technically, a policy-evaluation task: learning the value function for a fixed policy (Sections 4.1 and 6.1). The value function to be learned assigns to each state a value that predicts the return that will follow that state if the agent selects actions according to the given policy, where the return is the (possibly discounted) sum of all the future rewards. This is unrealistic as a model of the monkey's situation because the monkey would likely learn these predictions at the same time that it is learning to act correctly (as would a reinforcement learning algorithm that learns policies as well as value functions, such as an actor–critic algorithm), but this scenario is simpler to describe than one in which a policy and a value function are learned simultaneously.

Now imagine that the agent's experience divides into multiple trials, in each of which the same sequence of states repeats, with a distinct state occurring on each time step during the trial. Further imagine that the return being predicted is limited to the return over a trial, which makes a trial analogous to a reinforcement learning episode as we have defined it. In reality, of course, the returns being predicted are not confined to single trials, and the time interval between trials is an important factor in determining what an animal learns. This is true for TD learning as well, but here we assume that returns do not accumulate over multiple trials. Given this, then, a trial in experiments like those conducted by Schultz and colleagues is equivalent to an episode of reinforcement learning. (Though in this discussion, we will use the term trial instead of episode to relate better to the experiments.)

As usual, we also need to make an assumption about how states are represented as inputs to the learning algorithm, an assumption that influences how closely the TD error corresponds to dopamine neuron activity. We discuss this issue later, but for now we assume the same CSC representation used by Montague et al. (1996) in which there is a separate internal stimulus for each state visited at each time step in a trial. This reduces the process to the tabular case covered in the first part of this book. Finally, we assume that the agent uses TD(0) to learn a value function, V, stored in a lookup table initialized to be zero for all the states. We also assume that this is a deterministic task and that the discount factor, γ, is very nearly one so that we can ignore it.

Figure 15.4 shows the time courses of R, V, and δ at several stages of learning in this policy-evaluation task. The time axes represent the time interval over which a sequence of states is visited in a trial (where for clarity we omit showing individual states). The reward signal is zero throughout each trial except when the agent reaches the rewarding state, shown near the right end of the time line, when the reward signal becomes some positive number, say $R^\star$. The goal of TD learning is to predict the return for each state visited in a trial, which in this undiscounted case and given our assumption that predictions are confined to individual trials, is simply $R^\star$ for each state.

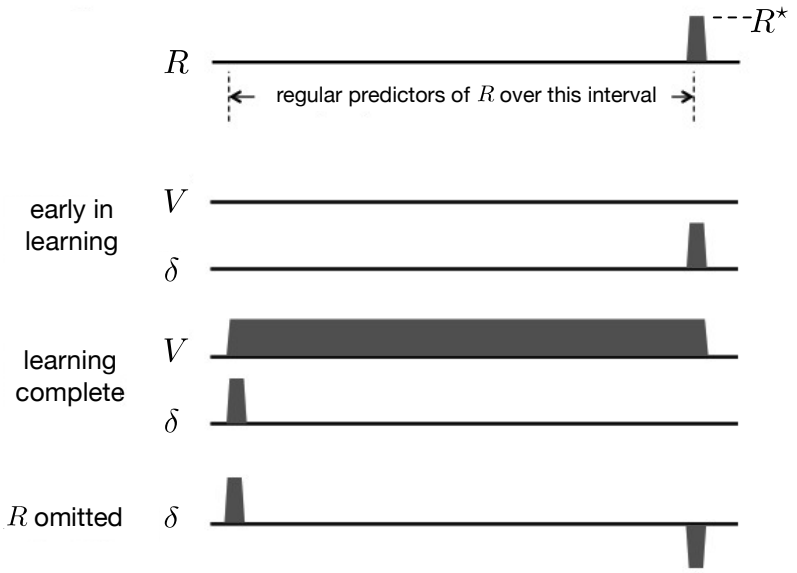


Figure 15.4: The behavior of the TD error δ during TD learning is consistent with features of the phasic activation of dopamine neurons. (Here δ is the TD error *available* at time t, i.e., δ_{t-1}). *Top*: a sequence of states, shown as an interval of regular predictors, is followed by a non-zero reward $R^\star$. *Early in learning*: the initial value function, V, and initial δ, which at first is equal to $R^\star$. *Learning complete*: the value function accurately predicts future reward, δ is positive at the earliest predictive state, and $\delta = 0$ at the time of the non-zero reward. $R^\star$ *omitted*: at the time the predicted reward is omitted, δ becomes negative. See text for a complete explanation of why this happens.

Preceding the rewarding state is a sequence of reward-predicting states, with the *earliest reward-predicting state* shown near the left end of the time line. This is like the state near the start of a trial, for example like the state marked by the instruction cue in a trial of the monkey experiment of Schultz et al. (1993) described above. It is the first state in a trial that reliably predicts that trial's reward. (Of course, in reality states visited on preceding trials are even earlier reward-predicting states, but because we are confining predictions to individual trials, these do not qualify as predictors of *this* trial's reward. Below we give a more satisfactory, though more abstract, description of an earliest reward-predicting state.) The *latest reward-predicting state* in a trial is the state immediately preceding the trial's rewarding state. This is the state near the far right end of the time line in Figure 15.4. Note that the rewarding state of a trial does not predict the return for that trial: the value of this state would come to predict the return over all the *following* trials, which here we are assuming to be zero in this episodic formulation.

Figure 15.4 shows the first-trial time courses of V and δ as the graphs labeled 'early in learning.' Because the reward signal is zero throughout the trial except when the rewarding state is reached, and all the V-values are zero, the TD error is also zero until it becomes $R^\star$ at the rewarding state. This follows because $\delta_{t-1} = R_t + V_t - V_{t-1} = R_t + 0 - 0 = R_t$, which is zero until it equals $R^\star$ when the reward occurs. Here V_t and V_{t-1} are respectively the estimated values of the states visited at times t and $t-1$ in a trial. The TD error at this stage of learning is analogous to a dopamine neuron responding to an unpredicted reward (e.g., a drop of apple juice) at the start of training.

Throughout this first trial and all successive trials, TD(0) updates occur at each state transition as described in Chapter 6. This successively increases the values of the reward-predicting states, with the increases spreading backwards from the rewarding state, until the values converge to the correct return predictions. In this case (because we are assuming no discounting) the correct predictions are equal to $R^\star$ for all the reward-predicting states. This can be seen in Figure 15.4 as the graph of V labeled 'learning complete' where the values of all the states from the earliest to the latest reward-predicting states all equal $R^\star$. The values of the states preceding the earliest reward-predicting state remain low (which Figure 15.4 shows as zero) because they are not reliable predictors of reward.

When learning is complete, that is, when V attains its correct values, the TD errors associated with transitions *from* any reward-predicting state are zero because the predictions are now accurate. This is because for a transition from a reward-predicting state to another reward-predicting state, we have $\delta_{t-1} = R_t + V_t - V_{t-1} = 0 + R^\star - R^\star = 0$, and for the transition from the latest reward-predicting state to the rewarding state, we have $\delta_{t-1} = R_t + V_t - V_{t-1} = R^\star + 0 - R^\star = 0$. On the other hand, the TD error on a transition from any state *to* the earliest reward-predicting state is positive because of the mismatch between this state's low value and the larger value of the following reward-predicting state. Indeed, if the value of a state preceding the earliest reward-predicting state were zero, then after the transition to the earliest reward-predicting state, we would have that $\delta_{t-1} = R_t + V_t - V_{t-1} = 0 + R^\star - 0 = R^\star$. The 'learning complete' graph of δ in Figure 15.4 shows this positive value at the earliest reward-predicting state, and zeros everywhere else.

The positive TD error upon transitioning to the earliest reward-predicting state is analogous to the persistence of dopamine responses to the earliest stimuli predicting reward. By the same token, when learning is complete, a transition from the latest reward-predicting state to the rewarding state produces a zero TD error because the latest reward-predicting state's value, being correct, cancels the reward. This parallels the observation that fewer dopamine neurons generate a phasic response to a fully predicted reward than to an unpredicted reward.

After learning, if the reward is suddenly omitted, the TD error goes negative at the usual time of reward because the value of the latest reward-predicting state is then too high: $\delta_{t-1} = R_t + V_t - V_{t-1} = 0 + 0 - R^\star = -R^\star$, as shown at the right end of the 'R omitted' graph of δ in Figure 15.4. This is like dopamine neuron activity decreasing below baseline at the time an expected reward is omitted as seen in the experiment of Schultz et al. (1993) described above and shown in Figure 15.3.

The idea of an *earliest reward-predicting state* deserves more attention. In the scenario described above, because experience is divided into trials, and we assumed that predictions are confined to individual trials, the earliest reward-predicting state is always the first state of a trial. Clearly this is artificial. A more general way to think of an earliest reward-predicting state is that it is an *unpredicted predictor* of reward, and there can be many such states. In an animal's life, many different states may precede an earliest reward-predicting state. However, because these states are more often followed by *other* states that do not predict reward, their reward-predicting powers, that is, their values, remain low. A TD algorithm, if operating throughout the animal's life, would update the values of these states too, but the updates would not consistently accumulate because, by assumption, none of these states reliably precedes an earliest reward-predicting state. If any of them did, they would be reward-predicting states as well. This might explain why with overtraining, dopamine responses decrease to even the earliest reward-predicting stimulus in a trial. With overtraining one would expect that even a formerly-unpredicted predictor state would become predicted by stimuli associated with earlier states: the animal's interaction with its environment both inside and outside of an experimental task would become commonplace. Upon breaking this routine with the introduction of a new task, however, one would see TD errors reappear, as indeed is observed in dopamine neuron activity.

The example described above explains why the TD error shares key features with the phasic activity of dopamine neurons when the animal is learning in a task similar to the idealized task of our example. But not every property of the phasic activity of dopamine neurons coincides so neatly with properties of δ. One of the most troubling discrepancies involves what happens when a reward occurs *earlier* than expected. We have seen that the omission of an expected reward produces a negative prediction error at the reward's expected time, which corresponds to the activity of dopamine neurons decreasing below baseline when this happens. If the reward arrives later than expected, it is then an unexpected reward and generates a positive prediction error. This happens with both TD errors and dopamine neuron responses. But when reward arrives earlier than expected, dopamine neurons do not do what the TD error does—at least with the CSC representation used by Montague et al. (1996) and by us in our example. Dopamine

neurons do respond to the early reward, which is consistent with a positive TD error because the reward is not predicted to occur then. However, at the later time when the reward is expected but omitted, the TD error is negative whereas, in contrast to this prediction, dopamine neuron activity does not drop below baseline in the way the TD model predicts (Hollerman and Schultz, 1998). Something more complicated is going on in the animal's brain than simply TD learning with a CSC representation.

Some of the mismatches between the TD error and dopamine neuron activity can be addressed by selecting suitable parameter values for the TD algorithm and by using stimulus representations other than the CSC representation. For instance, to address the early-reward mismatch just described, Suri and Schultz (1999) proposed a CSC representation in which the sequences of internal signals initiated by earlier stimuli are cancelled by the occurrence of a reward. Another proposal by Daw, Courville, and Touretzky (2006) is that the brain's TD system uses representations produced by statistical modeling carried out in sensory cortex rather than simpler representations based on raw sensory input. Ludvig, Sutton, and Kehoe (2008) found that TD learning with a microstimulus (MS) representation (Figure 14.1) fits the activity of dopamine neurons in the early-reward and other situations better than when a CSC representation is used. Pan, Schmidt, Wickens, and Hyland (2005) found that even with the CSC representation, prolonged eligibility traces improve the fit of the TD error to some aspects of dopamine neuron activity. In general, many fine details of TD-error behavior depend on subtle interactions between eligibility traces, discounting, and stimulus representations. Findings like these elaborate and refine the reward prediction error hypothesis without refuting its core claim that the phasic activity of dopamine neurons is well characterized as signaling TD errors.

On the other hand, there are other discrepancies between the TD theory and experimental data that are not so easily accommodated by selecting parameter values and stimulus representations (we mention some of these discrepancies in the Bibliographical and Historical Remarks section at the end of this chapter), and more mismatches are likely to be discovered as neuroscientists conduct ever more refined experiments. But the reward prediction error hypothesis has been functioning very effectively as a catalyst for improving our understanding of how the brain's reward system works. Intricate experiments have been designed to validate or refute predictions derived from the hypothesis, and experimental results have, in turn, led to refinement and elaboration of the TD error/dopamine hypothesis.

A remarkable aspect of these developments is that the reinforcement learning algorithms and theory that connect so well with properties of the dopamine system were developed from a computational perspective in total absence of any knowledge about the relevant properties of dopamine neurons—remember, TD learning and its connections to optimal control and dynamic programming were developed many years before any of the experiments were conducted that revealed the TD-like nature of dopamine neuron activity. This unplanned correspondence, despite not being perfect, suggests that the TD error/dopamine parallel captures something significant about brain reward processes.

In addition to accounting for many features of the phasic activity of dopamine neurons, the reward prediction error hypothesis links neuroscience to other aspects of reinforcement

learning, in particular, to learning algorithms that use TD errors as reinforcement signals. Neuroscience is still far from reaching complete understanding of the circuits, molecular mechanisms, and functions of the phasic activity of dopamine neurons, but evidence supporting the reward prediction error hypothesis, along with evidence that phasic dopamine responses are reinforcement signals for learning, suggest that the brain might implement something like an actor–critic algorithm in which TD errors play critical roles. Other reinforcement learning algorithms are plausible candidates too, but actor–critic algorithms fit the anatomy and physiology of the mammalian brain particularly well, as we describe in the following two sections.

15.7 Neural Actor–Critic

Actor–critic algorithms learn both policies and value functions. The 'actor' is the component that learns policies, and the 'critic' is the component that learns about whatever policy is currently being followed by the actor in order to 'criticize' the actor's action choices. The critic uses a TD algorithm to learn the state-value function for the actor's current policy. The value function allows the critic to critique the actor's action choices by sending TD errors, δ, to the actor. A positive δ means that the action was 'good' because it led to a state with a better-than-expected value; a negative δ means that the action was 'bad' because it led to a state with a worse-than-expected value. Based on these critiques, the actor continually updates its policy.

Two distinctive features of actor–critic algorithms are responsible for thinking that the brain might implement an algorithm like this. First, the two components of an actor–critic algorithm—the actor and the critic—suggest that two parts of the striatum—the dorsal and ventral subdivisions (Section 15.4), both critical for reward-based learning—may function respectively something like an actor and a critic. A second property of actor–critic algorithms that suggests a brain implementation is that the TD error has the dual role of being the reinforcement signal for both the actor and the critic, though it has a different influence on learning in each of these components. This fits well with several properties of the neural circuitry: axons of dopamine neurons target both the dorsal and ventral subdivisions of the striatum; dopamine appears to be critical for modulating synaptic plasticity in both structures; and how a neuromodulator such as dopamine acts on a target structure depends on properties of the target structure and not just on properties of the neuromodulator.

Section 13.5 presents actor–critic algorithms as policy gradient methods, but the actor–critic algorithm of Barto, Sutton, and Anderson (1983) was simpler and was presented as an artificial neural network (ANN). Here we describe an ANN implementation something like that of Barto et al., and we follow Takahashi, Schoenbaum, and Niv (2008) in giving a schematic proposal for how this ANN might be implemented by real neural networks in the brain. We postpone discussion of the actor and critic learning rules until Section 15.8, where we present them as special cases of the policy-gradient formulation and discuss what they suggest about how dopamine might modulate synaptic plasticity.

Figure 15.5a shows an implementation of an actor–critic algorithm as an ANN with component networks implementing the actor and the critic. The critic consists of a single neuron-like unit, V, whose output activity represents state values, and a component shown as the diamond labeled TD that computes TD errors by combining V's output with reward signals and with previous state values (as suggested by the loop from the TD diamond to itself). The actor network has a single layer of k actor units labeled A_i, $i = 1, \ldots, k$. The output of each actor unit is a component of a k-dimensional action vector. An alternative is that there are k separate actions, one commanded by each actor unit, that compete with one another to be executed, but here we will think of the entire A-vector as an action.

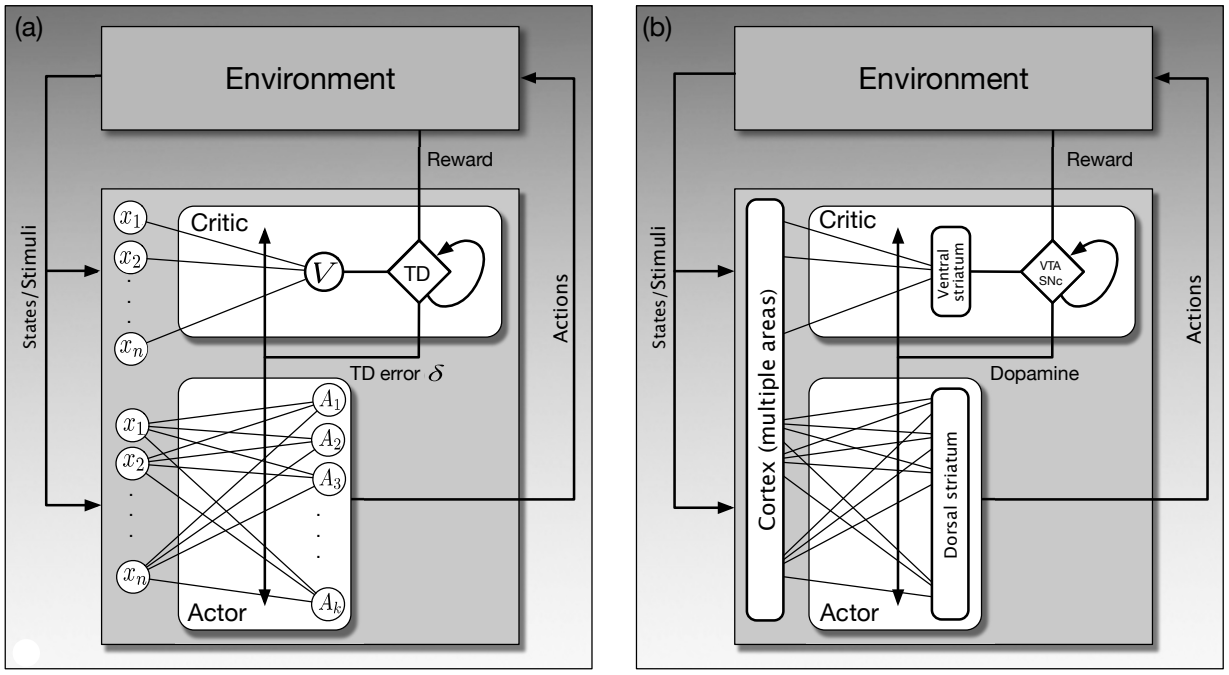


Figure 15.5: Actor–critic ANN and a hypothetical neural implementation. a) Actor–critic algorithm as an ANN. The actor adjusts a policy based on the TD error δ it receives from the critic; the critic adjusts state-value parameters using the same δ. The critic produces a TD error from the reward signal, R, and the current change in its estimate of state values. The actor does not have direct access to the reward signal, and the critic does not have direct access to the action. b) Hypothetical neural implementation of an actor–critic algorithm. The actor and the value-learning part of the critic are respectively placed in the dorsal and ventral subdivisions of the striatum. The TD error is transmitted by dopamine neurons located in the VTA and SNpc to modulate changes in synaptic efficacies of input from cortical areas to the ventral and dorsal striatum. Adapted from Frontiers in Neuroscience, vol. 2(1), 2008, Y. Takahashi, G. Schoenbaum, and Y. Niv, Silencing the critics: Understanding the effects of cocaine sensitization on dorsolateral and ventral striatum in the context of an Actor/Critic model.

Both the critic and actor networks receive input consisting of multiple features representing the state of the agent's environment. (Recall from Chapter 1 that the environment of a reinforcement learning agent includes components both inside and outside of the 'organism' containing the agent.) The figure shows these features as the circles labeled $x_1, x_2, \ldots, x_n$, shown twice just to keep the figure simple. A weight representing the efficacy of a synapse is associated with each connection from each feature x_i to the critic unit, V, and to each of the action units, A_i. The weights in the critic network parameterize the value function, and the weights in the actor network parameterize the policy. The networks learn as these weights change according to the critic and actor learning rules that we describe in the following section.

The TD error produced by circuitry in the critic is the reinforcement signal for changing the weights in both the critic and the actor networks. This is shown in Figure 15.5a by the line labeled 'TD error δ' extending across all of the connections in the critic and actor networks. This aspect of the network implementation, together with the reward prediction error hypothesis and the fact that the activity of dopamine neurons is so widely distributed by the extensive axonal arbors of these neurons, suggests that an actor–critic network something like this may not be too farfetched as a hypothesis about how reward-related learning might happen in the brain.

Figure 15.5b suggests—very schematically—how the ANN on the figure's left might map onto structures in the brain according to the hypothesis of Takahashi et al. (2008). The hypothesis puts the actor and the value-learning part of the critic respectively in the dorsal and ventral subdivisions of the striatum, the input structure of the basal ganglia. Recall from Section 15.4 that the dorsal striatum is primarily implicated in influencing action selection, and the ventral striatum is thought to be critical for different aspects of reward processing, including the assignment of affective value to sensations. The cerebral cortex, along with other structures, sends input to the striatum conveying information about stimuli, internal states, and motor activity.

In this hypothetical actor–critic brain implementation, the ventral striatum sends value information to the VTA and SNpc, where dopamine neurons in these nuclei combine it with information about reward to generate activity corresponding to TD errors (though exactly how dopaminergic neurons calculate these errors is not yet understood). The 'TD error δ' line in Figure 15.5a becomes the line labeled 'Dopamine' in Figure 15.5b, which represents the widely branching axons of dopamine neurons whose cell bodies are in the VTA and SNpc. Referring back to Figure 15.1, these axons make synaptic contact with the spines on the dendrites of medium spiny neurons, the main input/output neurons of both the dorsal and ventral divisions of the striatum. Axons of the cortical neurons that send input to the striatum make synaptic contact on the tips of these spines. According to the hypothesis, it is at these spines where changes in the efficacies of the synapses from cortical regions to the striatum are governed by learning rules that critically depend on a reinforcement signal supplied by dopamine.

An important implication of the hypothesis illustrated in Figure 15.5b is that the dopamine signal is not the 'master' reward signal like the scalar R_t of reinforcement learning. In fact, the hypothesis implies that one should not necessarily be able to probe the brain and record any signal like R_t in the activity of any single neuron.

Many interconnected neural systems generate reward-related information, with different structures being recruited depending on different types of rewards. Dopamine neurons receive information from many different brain areas, so the input to the SNpc and VTA labeled 'Reward' in Figure 15.5b should be thought of as vector of reward-related information arriving to neurons in these nuclei along multiple input channels. What the theoretical scalar reward signal R_t might correspond to, then, is the net contribution of all reward-related information to dopamine neuron activity. It is the result of a pattern of activity across many neurons in different areas of the brain.

Although the actor–critic neural implementation illustrated in Figure 15.5b may be correct on some counts, it clearly needs to be refined, extended, and modified to qualify as a full-fledged model of the function of the phasic activity of dopamine neurons. The Historical and Bibliographic Remarks section at the end of this chapter cites publications that discuss in more detail both empirical support for this hypothesis and places where it falls short. We now look in detail at what the actor and critic learning algorithms suggest about the rules governing changes in synaptic efficacies of corticostriatal synapses.

15.8 Actor and Critic Learning Rules

If the brain does implement something like the actor–critic algorithm—and assuming populations of dopamine neurons broadcast a common reinforcement signal to the corticostriatal synapses of both the dorsal and ventral striatum as illustrated in Figure 15.5b (which is likely an oversimplification as we mentioned above)—then this reinforcement signal affects the synapses of these two structures in different ways. The learning rules for the critic and the actor use the same reinforcement signal, the TD error δ, but its effect on learning is different for these two components. The TD error (combined with eligibility traces) tells the actor how to update action probabilities in order to reach higher-valued states. Learning by the actor is like instrumental conditioning using a Law-of-Effect-type learning rule (Section 1.7): the actor works to keep δ as positive as possible. On the other hand, the TD error (when combined with eligibility traces) tells the critic the direction and magnitude in which to change the parameters of the value function in order to improve its predictive accuracy. The critic works to reduce δ's magnitude to be as close to zero as possible using a learning rule like the TD model of classical conditioning (Section 14.2). The difference between the critic and actor learning rules is relatively simple, but this difference has a profound effect on learning and is essential to how the actor–critic algorithm works. The difference lies solely in the eligibility traces each type of learning rule uses.

More than one set of learning rules can be used in actor–critic neural networks like those in Figure 15.5b but, to be specific, here we focus on the actor–critic algorithm for continuing problems with eligibility traces presented in Section 13.6. On each transition from state S_t to state S_{t+1}, taking action A_t and receiving reward R_{t+1}, that algorithm computes the TD error (δ) and then updates the eligibility trace vectors ($\mathbf{z}_t^{\mathbf{w}}$ and $\mathbf{z}_t^{\boldsymbol{\theta}}$) and

the parameters for the critic and actor ($\mathbf{w}$ and $\boldsymbol{\theta}$), according to

$$\begin{aligned}
\delta_t &= R_{t+1} + \gamma \hat{v}(S_{t+1},\mathbf{w}) - \hat{v}(S_t,\mathbf{w}),\\
\mathbf{z}^{\mathbf{w}}_t &= \gamma\lambda^{\mathbf{w}}\mathbf{z}^{\mathbf{w}}_{t-1} + \nabla\hat{v}(S_t,\mathbf{w}),\\
\mathbf{z}^{\boldsymbol{\theta}}_t &= \gamma\lambda^{\boldsymbol{\theta}}\mathbf{z}^{\boldsymbol{\theta}}_{t-1} + \nabla \ln \pi(A_t|S_t,\boldsymbol{\theta}),\\
\mathbf{w} &\leftarrow \mathbf{w} + \alpha^{\mathbf{w}}\delta_t\mathbf{z}^{\mathbf{w}}_t,\\
\boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}}\delta\mathbf{z}^{\boldsymbol{\theta}}_t,
\end{aligned}$$

where $\gamma \in [0,1)$ is a discount-rate parameter, $\lambda^{\mathbf{w}} \in [0,1]$ and $\lambda^{\boldsymbol{\theta}} \in [0,1]$ are bootstrapping parameters for the critic and the actor respectively, and $\alpha^{\mathbf{w}} > 0$ and $\alpha^{\boldsymbol{\theta}} > 0$ are analogous step-size parameters.

Think of the approximate value function $\hat{v}$ as the output of a single linear neuron-like unit, called the *critic unit* and labeled V in Figure 15.5a. Then the value function is a linear function of the feature-vector representation of state s, $\mathbf{x}(s) = (x_1(s),\ldots,x_n(s))^\top$, parameterized by a weight vector $\mathbf{w} = (w_1,\ldots,w_n)^\top$:

$$\hat{v}(s,\mathbf{w}) = \mathbf{w}^\top\mathbf{x}(s). \tag{15.1}$$

Each $x_i(s)$ is like the presynaptic signal to a neuron's synapse whose efficacy is w_i. The weights of the critic are incremented according to the rule above by $\alpha^{\mathbf{w}}\delta_t\mathbf{z}^{\mathbf{w}}_t$, where the reinforcement signal, δ_t, corresponds to a dopamine signal being broadcast to all of the critic unit's synapses. The eligibility trace vector, $\mathbf{z}^{\mathbf{w}}_t$, for the critic unit is a trace (average of recent values) of $\nabla\hat{v}(S_t,\mathbf{w})$. Because $\hat{v}(s,\mathbf{w})$ is linear in the weights, $\nabla\hat{v}(S_t,\mathbf{w}) = \mathbf{x}(S_t)$.

In neural terms, this means that each synapse has its own eligibility trace, which is one component of the vector $\mathbf{z}^{\mathbf{w}}_t$. A synapse's eligibility trace accumulates according to the level of activity arriving at that synapse, that is, the level of presynaptic activity, represented here by the component of the feature vector $\mathbf{x}(S_t)$ arriving at that synapse. The trace otherwise decays toward zero at a rate governed by the fraction $\lambda^{\mathbf{w}}$. A synapse is *eligible for modification* as long as its eligibility trace is non-zero. How the synapse's efficacy is actually modified depends on the reinforcement signals that arrive while the synapse is eligible. We call eligibility traces like these of the critic unit's synapses *non-contingent eligibility traces* because they only depend on presynaptic activity and are not contingent in any way on postsynaptic activity.

The non-contingent eligibility traces of the critic unit's synapses mean that the critic unit's learning rule is essentially the TD model of classical conditioning described in Section 14.2. With the definition we have given above of the critic unit and its learning rule, the critic in Figure 15.5a is the same as the critic in the ANN actor–critic of Barto et al. (1983). Clearly, a critic like this consisting of just one linear neuron-like unit is the simplest starting point; this critic unit is a proxy for a more complicated neural network able to learn value functions of greater complexity.

The actor in Figure 15.5a is a one-layer network of k neuron-like actor units, each receiving at time t the same feature vector, $\mathbf{x}(S_t)$, that the critic unit receives. Each actor unit j, $j = 1,\ldots,k$, has its own weight vector, $\boldsymbol{\theta}_j$, but because the actor units are all identical, we describe just one of the units and omit the subscript. One way for these

units to follow the actor–critic algorithm given in the equations above is for each to be a *Bernoulli-logistic unit*. This means that the output of each actor unit at each time is a random variable, A_t, taking value 0 or 1. Think of value 1 as the neuron firing, that is, emitting an action potential. The weighted sum, $\boldsymbol{\theta}^\top \mathbf{x}(S_t)$, of a unit's input vector determines the unit's action probabilities via the exponential soft-max distribution (13.2), which for two actions is the logistic function:

$$\pi(1|s,\boldsymbol{\theta}) = 1 - \pi(0|s,\boldsymbol{\theta}) = \frac{1}{1+\exp(-\boldsymbol{\theta}^\top \mathbf{x}(s))}. \tag{15.2}$$

The weights of each actor unit are incremented, as above, by: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}}\, \delta_t\, \mathbf{z}_t^{\boldsymbol{\theta}}$, where δ again corresponds to the dopamine signal: the same reinforcement signal that is sent to all the critic unit's synapses. Figure 15.5a shows δ_t being broadcast to all the synapses of all the actor units (which makes this actor network a *team* of reinforcement learning agents, something we discuss in Section 15.10 below). The actor eligibility trace vector $\mathbf{z}_t^{\boldsymbol{\theta}}$ is a trace (average of recent values) of $\nabla \ln \pi(A_t|S_t,\boldsymbol{\theta})$. To understand this eligibility trace refer to Exercise 13.5, which defines this kind of unit and asks you to give a learning rule for it. That exercise asked you to express $\nabla \ln \pi(a|s,\boldsymbol{\theta})$ in terms of a, $\mathbf{x}(s)$, and $\pi(a|s,\boldsymbol{\theta})$ (for arbitrary state s and action a) by calculating the gradient. For the action and state actually occurring at time t, the answer is

$$\nabla \ln \pi(A_t|S_t,\boldsymbol{\theta}) = \big(A_t - \pi(1|S_t,\boldsymbol{\theta})\big)\mathbf{x}(S_t). \tag{15.3}$$

Unlike the non-contingent eligibility trace of a critic synapse that only accumulates the presynaptic activity $\mathbf{x}(S_t)$, the eligibility trace of an actor unit's synapse in addition depends on the activity of the actor unit itself. We call this a *contingent eligibility trace* because it is contingent on this postsynaptic activity. The eligibility trace at each synapse continually decays, but increments or decrements depending on the activity of the presynaptic neuron *and* whether or not the postsynaptic neuron fires. The factor $A_t - \pi(1|S_t,\boldsymbol{\theta})$ in (15.3) is positive when $A_t = 1$ and negative otherwise. *The postsynaptic contingency in the eligibility traces of actor units is the only difference between the critic and actor learning rules.* By keeping information about what actions were taken in what states, contingent eligibility traces allow credit for reward (positive δ), or blame for punishment (negative δ), to be apportioned among the policy parameters (the efficacies of the actor units' synapses) according to the contributions these parameters made to the units' outputs that could have influenced later values of δ. Contingent eligibility traces mark the synapses as to how they should be modified to alter the units' future responses to favor positive values of δ.

What do the critic and actor learning rules suggest about how efficacies of corticostriatal synapses change? Both learning rules are related to Donald Hebb's classic proposal that whenever a presynaptic signal participates in activating the postsynaptic neuron, the synapse's efficacy increases (Hebb, 1949). The critic and actor learning rules share with Hebb's proposal the idea that changes in a synapse's efficacy depend on the interaction of several factors. In the critic learning rule the interaction is between the reinforcement signal δ and eligibility traces that depend only on presynaptic signals. Neuroscientists call this a *two-factor learning rule* because the interaction is between two signals or

quantities. The actor learning rule, on the other hand, is a *three-factor learning rule* because, in addition to depending on δ, its eligibility traces depend on both presynaptic and postsynaptic activity. Unlike Hebb's proposal, however, the relative timing of the factors is critical to how synaptic efficacies change, with eligibility traces intervening to allow the reinforcement signal to affect synapses that were active in the recent past.

Some subtleties about signal timing for the actor and critic learning rules deserve closer attention. In defining the neuron-like actor and critic units, we ignored the small amount of time it takes synaptic input to effect the firing of a real neuron. When an action potential from the presynaptic neuron arrives at a synapse, neurotransmitter molecules are released that diffuse across the synaptic cleft to the postsynaptic neuron, where they bind to receptors on the postsynaptic neuron's surface; this activates molecular machinery that causes the postsynaptic neuron to fire (or to inhibit its firing in the case of inhibitory synaptic input). This process can take several tens of milliseconds. According to (15.1) and (15.2), though, the input to a critic and actor unit instantaneously produces the unit's output. Ignoring activation time like this is common in abstract models of Hebbian-style plasticity in which synaptic efficacies change according to a simple product of simultaneous pre- and postsynaptic activity. More realistic models must take activation time into account.

Activation time is especially important for a more realistic actor unit because it influences how contingent eligibility traces have to work in order to properly apportion credit for reinforcement to the appropriate synapses. The expression $\big(A_t - \pi(1|S_t, \boldsymbol{\theta})\big)\mathbf{x}(S_t)$ defining contingent eligibility traces for the actor unit's learning rule given above includes the postsynaptic factor $\big(A_t - \pi(1|S_t, \boldsymbol{\theta})\big)$ and the presynaptic factor $\mathbf{x}(S_t)$. This works because by ignoring activation time, the presynaptic activity $\mathbf{x}(S_t)$ participates in *causing* the postsynaptic activity appearing in $\big(A_t - \pi(1|S_t, \boldsymbol{\theta})\big)$. To assign credit for reinforcement correctly, the presynaptic factor defining the eligibility trace must be a cause of the postsynaptic factor that also defines the trace. Contingent eligibility traces for a more realistic actor unit would have to take activation time into account. (Activation time should not be confused with the time required for a neuron to receive a reinforcement signal influenced by that neuron's activity. The function of eligibility traces is to span this time interval which is generally much longer than the activation time. We discuss this further in the following section.)

There are hints from neuroscience for how this process might work in the brain. Neuroscientists have discovered a form of Hebbian plasticity called *spike-timing-dependent plasticity* (STDP) that lends plausibility to the existence of actor-like synaptic plasticity in the brain. STDP is a Hebbian-style plasticity, but changes in a synapse's efficacy depend on the relative timing of presynaptic and postsynaptic action potentials. The dependence can take different forms, but in the one most studied, a synapse increases in strength if spikes incoming via that synapse arrive shortly before the postsynaptic neuron fires. If the timing relation is reversed, with a presynaptic spike arriving shortly after the postsynaptic neuron fires, then the strength of the synapse decreases. STDP is a type of Hebbian plasticity that takes the activation time of a neuron into account, which is one of the ingredients needed for actor-like learning.

The discovery of STDP has led neuroscientists to investigate the possibility of a three-factor form of STDP in which neuromodulatory input must follow appropriately-timed pre- and postsynaptic spikes. This form of synaptic plasticity, called *reward-modulated STDP*, is much like the actor learning rule discussed here. Synaptic changes that would be produced by regular STDP only occur if there is neuromodulatory input within a time window after a presynaptic spike is closely followed by a postsynaptic spike. Evidence is accumulating that reward-modulated STDP occurs at the spines of medium spiny neurons of the dorsal striatum, with dopamine providing the neuromodulatory factor—the sites where actor learning takes place in the hypothetical neural implementation of an actor–critic algorithm illustrated in Figure 15.5b. Experiments have demonstrated reward-modulated STDP in which lasting changes in the efficacies of corticostriatal synapses occur only if a neuromodulatory pulse arrives within a time window that can last up to 10 seconds after a presynaptic spike is closely followed by a postsynaptic spike (Yagishita et al. 2014). Although the evidence is indirect, these experiments point to the existence of contingent eligibility traces having prolonged time courses. The molecular mechanisms producing these traces, as well as the much shorter traces that likely underly STDP, are not yet understood, but research focusing on time-dependent and neuromodulator-dependent synaptic plasticity is continuing.

The neuron-like actor unit that we have described here, with its Law-of-Effect-style learning rule, appeared in somewhat simpler form in the actor–critic network of Barto et al. (1983). That network was inspired by the "hedonistic neuron" hypothesis proposed by physiologist A. H. Klopf (1972, 1982). Not all the details of Klopf's hypothesis are consistent with what has been learned about synaptic plasticity, but the discovery of STDP and the growing evidence for a reward-modulated form of STDP suggest that Klopf's ideas may not have been far off the mark. We discuss Klopf's hedonistic neuron hypothesis next.

15.9 Hedonistic Neurons

In his hedonistic neuron hypothesis, Klopf (1972, 1982) conjectured that individual neurons seek to maximize the difference between synaptic input treated as rewarding and synaptic input treated as punishing by adjusting the efficacies of their synapses on the basis of rewarding or punishing consequences of their own action potentials. In other words, individual neurons can be trained with response-contingent reinforcement like an animal can be trained in an instrumental conditioning task. His hypothesis included the idea that rewards and punishments are conveyed to a neuron via the same synaptic input that excites or inhibits the neuron's spike-generating activity. (Had Klopf known what we know today about neuromodulatory systems, he might have assigned the reinforcing role to neuromodulatory input, but he wanted to avoid any centralized source of training information.) Synaptically-local traces of past pre- and postsynaptic activity had the key function in Klopf's hypothesis of making synapses *eligible*—the term he introduced—for modification by later reward or punishment. He conjectured that these traces are implemented by molecular mechanisms local to each synapse and therefore different from the electrical activity of both the pre- and the postsynaptic neurons. In

the Bibliographical and Historical Remarks section of this chapter we bring attention to some similar proposals made by others.

Klopf specifically conjectured that synaptic efficacies change in the following way. When a neuron fires an action potential, all of its synapses that were active in contributing to that action potential become eligible to undergo changes in their efficacies. If the action potential is followed within an appropriate time period by an increase of reward, the efficacies of all the eligible synapses increase. Symmetrically, if the action potential is followed within an appropriate time period by an increase of punishment, the efficacies of eligible synapses decrease. This is implemented by triggering an eligibility trace at a synapse upon a coincidence of presynaptic and postsynaptic activity (or more exactly, upon pairing of presynaptic activity with the postsynaptic activity that that presynaptic activity participates in causing)—what we call a contingent eligibility trace. This is essentially the three-factor learning rule of an actor unit described in the previous section.

The shape and time course of an eligibility trace in Klopf's theory reflects the durations of the many feedback loops in which the neuron is embedded, some of which lie entirely within the brain and body of the organism, while others extend out through the organism's external environment as mediated by its motor and sensory systems. His idea was that the shape of a synaptic eligibility trace is like a histogram of the durations of the feedback loops in which the neuron is embedded. The peak of an eligibility trace would then occur at the duration of the most prevalent feedback loops in which that neuron participates. The eligibility traces used by algorithms described in this book are simplified versions of Klopf's original idea, being exponentially (or geometrically) decreasing functions controlled by the parameters λ and γ. This simplifies simulations as well as theory, but we regard these simple eligibility traces as a placeholders for traces closer to Klopf's original conception, which would have computational advantages in complex reinforcement learning systems by refining the credit-assignment process.

Klopf's hedonistic neuron hypothesis is not as implausible as it may at first appear. A well-studied example of a single cell that seeks some stimuli and avoids others is the bacterium *Escherichia coli*. The movement of this single-cell organism is influenced by chemical stimuli in its environment, behavior known as chemotaxis. It swims in its liquid environment by rotating hairlike structures called flagella attached to its surface. (Yes, it rotates them!) Molecules in the bacterium's environment bind to receptors on its surface. Binding events modulate the frequency with which the bacterium reverses flagellar rotation. Each reversal causes the bacterium to tumble in place and then head off in a random new direction. A little chemical memory and computation causes the frequency of flagellar reversal to decrease when the bacterium swims toward higher concentrations of molecules it needs to survive (attractants) and increase when the bacterium swims toward higher concentrations of molecules that are harmful (repellants). The result is that the bacterium tends to persist in swimming up attractant gradients and tends to avoid swimming up repellant gradients.

The chemotactic behavior just described is called klinokinesis. It is a kind of trial-and-error behavior, although it is unlikely that learning is involved: the bacterium needs a modicum of short-term memory to detect molecular concentration gradients, but it probably does not maintain long-term memories. Artificial intelligence pioneer Oliver

Selfridge called this strategy "run and twiddle," pointing out its utility as a basic adaptive strategy: "keep going in the same way if things are getting better, and otherwise move around" (Selfridge, 1978, 1984). Similarly, one might think of a neuron "swimming" (not literally of course) in a medium composed of the complex collection of feedback loops in which it is embedded, acting to obtain one type of input signal and to avoid others. Unlike the bacterium, however, the neuron's synaptic strengths retain information about its past trial-and-error behavior. If this view of the behavior of a neuron (or just one type of neuron) is plausible, then the closed-loop nature of how the neuron interacts with its environment is important for understanding its behavior, where the neuron's environment consists of the rest of the animal together with the environment with which the animal as a whole interacts.

Klopf's hedonistic neuron hypothesis extended beyond the idea that individual neurons are reinforcement learning agents. He argued that many aspects of intelligent behavior can be understood as the result of the collective behavior of a population of self-interested hedonistic neurons interacting with one another in an immense society or economic system making up an animal's nervous system. Whether or not this view of nervous systems is useful, the collective behavior of reinforcement learning agents has implications for neuroscience. We take up this subject next.

15.10 Collective Reinforcement Learning

The behavior of populations of reinforcement learning agents is deeply relevant to the study of social and economic systems, and if anything like Klopf's hedonistic neuron hypothesis is correct, to neuroscience as well. The hypothesis described above about how an actor–critic algorithm might be implemented in the brain only narrowly addresses the implications of the fact that the dorsal and ventral subdivisions of the striatum, the respective locations of the actor and the critic according to the hypothesis, each contain millions of medium spiny neurons whose synapses undergo change modulated by phasic bursts of dopamine neuron activity.

The actor in Figure 15.5a is a single-layer network of k actor units. The actions produced by this network are vectors $(A_1, A_2, \cdots, A_k)^\top$ presumed to drive the animal's behavior. Changes in the efficacies of the synapses of all of these units depend on the reinforcement signal δ. Because actor units attempt to make δ as large as possible, δ effectively acts as a reward signal for them (so in this case reinforcement is the same as reward). Thus, each actor unit is itself a reinforcement learning agent—a hedonistic neuron if you will. Now, to make the situation as simple as possible, assume that each of these units receives the same reward signal at the same time (although, as indicated above, the assumption that dopamine is released at all the corticostriatal synapses under the same conditions and at the same times is likely an oversimplification).

What can reinforcement learning theory tell us about what happens when all members of a population of reinforcement learning agents learn according to a common reward signal? The field of *multi-agent reinforcement learning* considers many aspects of learning by populations of reinforcement learning agents. Although this field is beyond the scope of this book, we believe that some of its basic concepts and results are relevant to thinking

about the brain's diffuse neuromodulatory systems. In multi-agent reinforcement learning (and in game theory), the scenario in which all the agents try to maximize a common reward signal that they simultaneously receive is known as a *cooperative game* or a *team problem*.

What makes a team problem interesting and challenging is that the common reward signal sent to each agent evaluates the *pattern* of activity produced by the entire population, that is, it evaluates the *collective action* of the team members. This means that any individual agent has only limited ability to affect the reward signal because any single agent contributes just one component of the collective action evaluated by the common reward signal. Effective learning in this scenario requires addressing a *structural credit assignment problem*: which team members, or groups of team members, deserve credit for a favorable reward signal, or blame for an unfavorable reward signal? It is a *cooperative* game, or a team problem, because the agents are united in seeking to increase the same reward signal: there are no conflicts of interest among the agents. The scenario would be a *competitive game* if different agents receive different reward signals, where each reward signal again evaluates the collective action of the population, and the objective of each agent is to increase its own reward signal. In this case there might be conflicts of interest among the agents, meaning that actions that are good for some agents are bad for others. Even deciding what the best collective action should be is a non-trivial aspect of game theory. This competitive setting might be relevant to neuroscience too (for example, to account for heterogeneity of dopamine neuron activity), but here we focus only on the cooperative, or team, case.

How can each reinforcement learning agent in a team learn to "do the right thing" so that the collective action of the team is highly rewarded? An interesting result is that if each agent can learn effectively despite its reward signal being corrupted by a large amount of noise, and despite its lack of access to complete state information, then the population as a whole will learn to produce collective actions that improve as evaluated by the common reward signal, even when the agents cannot communicate with one another. Each agent faces its own reinforcement learning task in which its influence on the reward signal is deeply buried in the noise created by the influences of other agents. In fact, for any agent, all the other agents are part of its environment because its input, both the part conveying state information and the reward part, depends on how all the other agents are behaving. Furthermore, lacking access to the actions of the other agents, indeed lacking access to the parameters determining their policies, each agent can only partially observe the state of its environment. This makes each team member's learning task very difficult, but if each uses a reinforcement learning algorithm able to increase a reward signal even under these difficult conditions, teams of reinforcement learning agents can learn to produce collective actions that improve over time as evaluated by the team's common reward signal.

If the team members are neuron-like units, then each unit has to have the goal of increasing the amount of reward it receives over time, as the actor unit does that we described in Section 15.8. Each unit's learning algorithm has to have two essential features. First, it has to use contingent eligibility traces. Recall that a contingent eligibility trace, in neural terms, is initiated (or increased) at a synapse when its presynaptic input

participates in causing the postsynaptic neuron to fire. A non-contingent eligibility trace, in contrast, is initiated or increased by presynaptic input independently of what the postsynaptic neuron does. As explained in Section 15.8, by keeping information about what actions were taken in what states, contingent eligibility traces allow credit for reward, or blame for punishment, to be apportioned to an agent's policy parameters according to the contribution the values of these parameters made in determining the agent's action. By similar reasoning, a team member must remember its recent action so that it can either increase or decrease the likelihood of producing that action according to the reward signal that is subsequently received. The action component of a contingent eligibility trace implements this action memory. Because of the complexity of the learning task, however, contingent eligibility is merely a preliminary step in the credit assignment process: the relationship between a single team member's action and changes in the team's reward signal is a statistical correlation that has to be estimated over many trials. Contingent eligibility is an essential but preliminary step in this process.

Learning with non-contingent eligibility traces does not work at all in the team setting because it does not provide a way to correlate actions with consequent changes in the reward signal. Non-contingent eligibility traces are adequate for learning to predict, as the critic component of the actor–critic algorithm does, but they do not support learning to control, as the actor component must do. The members of a population of critic-like agents may still receive a common reinforcement signal, but they would all learn to predict the same quantity (which in the case of an actor–critic method, would be the expected return for the current policy). How successful each member of the population would be in learning to predict the expected return would depend on the information it receives, which could be very different for different members of the population. There would be no need for the population to produce differentiated patterns of activity. This is not a team problem as defined here.

A second requirement for collective learning in a team problem is that there has to be variability in the actions of the team members in order for the team to explore the space of collective actions. The simplest way for a team of reinforcement learning agents to do this is for each member to independently explore its own action space through persistent variability in its output. This will cause the team as a whole to vary its collective actions. For example, a team of the actor units described in Section 15.8 explores the space of collective actions because the output of each unit, being a Bernoulli-logistic unit, probabilistically depends on the weighted sum of its input vector's components. The weighted sum biases firing probability up or down, but there is always variability. Because each unit uses a REINFORCE policy gradient algorithm (Chapter 13), each unit adjusts its weights with the goal of maximizing the average reward rate it experiences while stochastically exploring its own action space. One can show, as Williams (1992) did, that a team of Bernoulli-logistic REINFORCE units implements a policy gradient algorithm *as a whole* with respect to average rate of the team's common reward signal, where the actions are the collective actions of the team.

Further, Williams (1992) showed that a team of Bernoulli-logistic units using REINFORCE ascends the average reward gradient when the units in the team are interconnected to form a multilayer ANN. In this case, the reward signal is broadcast to all the units in

the network, though reward may depend only on the collective actions of the network's output units. This means that a multilayer team of Bernoulli-logistic REINFORCE units learns like a multilayer network trained by the widely-used error backpropagation method, but in this case the backpropagation process is replaced by the broadcasted reward signal. In practice, the error backpropagation method is considerably faster, but the reinforcement learning team method is more plausible as a neural mechanism, especially in light of what is being learned about reward-modulated STDP as discussed in Section 15.8.

Exploration through independent exploration by team members is only the simplest way for a team to explore; more sophisticated methods are possible if the team members coordinate their actions to focus on particular parts of the collective action space, either by communicating with one another or by responding to common inputs. There are also mechanisms more sophisticated than contingent eligibility traces for addressing structural credit assignment, which is easier in a team problem when the set of possible collective actions is restricted in some way. An extreme case is a winner-take-all arrangement (for example, the result of lateral inhibition in the brain) that restricts collective actions to those to which only one, or a few, team members contribute. In this case the winners get the credit or blame for resulting reward or punishment.

Details of learning in cooperative games (or team problems) and non-cooperative game problems are beyond the scope of this book. The Bibliographical and Historical Remarks section at the end of this chapter cites a selection of the relevant publications, including extensive references to research on implications for neuroscience of collective reinforcement learning.

15.11 Model-based Methods in the Brain

Reinforcement learning's distinction between model-free and model-based algorithms is proving to be useful for thinking about animal learning and decision processes. Section 14.6 discusses how this distinction aligns with that between habitual and goal-directed animal behavior. The hypothesis discussed above about how the brain might implement an actor–critic algorithm is relevant only to an animal's habitual mode of behavior because the basic actor–critic method is model-free. What neural mechanisms are responsible for producing goal-directed behavior, and how do they interact with those underlying habitual behavior?

One way to investigate questions about the brain structures involved in these modes of behavior is to inactivate an area of a rat's brain and then observe what the rat does in an outcome-devaluation experiment (Section 14.6). Results from experiments like these indicate that the actor–critic hypothesis described above is too simple in placing the actor in the dorsal striatum. Inactivating one part of the dorsal striatum, the dorsolateral striatum (DLS), impairs habit learning, causing the animal to rely more on goal-directed processes. On the other hand, inactivating the dorsomedial striatum (DMS) impairs goal-directed processes, requiring the animal to rely more on habit learning. Results like these support the view that the DLS in rodents is more involved in model-free processes, whereas their DMS is more involved in model-based processes. Results of

studies with human subjects in similar experiments using functional neuroimaging, and with non-human primates, support the view that the analogous structures in the primate brain are differentially involved in habitual and goal-directed modes of behavior.

Other studies identify activity associated with model-based processes in the prefrontal cortex of the human brain, the front-most part of the frontal cortex implicated in executive function, including planning and decision making. Specifically implicated is the orbitofrontal cortex (OFC), the part of the prefrontal cortex immediately above the eyes. Functional neuroimaging in humans, and also recordings of the activities of single neurons in monkeys, reveals strong activity in the OFC related to the subjective reward value of biologically significant stimuli, as well as activity related to the reward expected as a consequence of actions. Although not free of controversy, these results suggest significant involvement of the OFC in goal-directed choice. It may be critical for the reward part of an animal's environment model.

Another structure involved in model-based behavior is the hippocampus, a structure critical for memory and spatial navigation. A rat's hippocampus plays a critical role in the rat's ability to navigate a maze in the goal-directed manner that led Tolman to the idea that animals use models, or cognitive maps, in selecting actions (Section 14.5). The hippocampus may also be a critical component of our human ability to imagine new experiences (Hassabis and Maguire, 2007; Ólafsdóttir, Barry, Saleem, Hassabis, and Spiers, 2015).

The findings that most directly implicate the hippocampus in planning—the process needed to enlist an environment model in making decisions—come from experiments that decode the activity of neurons in the hippocampus to determine what part of space hippocampal activity is representing on a moment-to-moment basis. When a rat pauses at a choice point in a maze, the representation of space in the hippocampus sweeps forward (and not backwards) along the possible paths the animal can take from that point (Johnson and Redish, 2007). Furthermore, the spatial trajectories represented by these sweeps closely correspond to the rat's subsequent navigational behavior (Pfeiffer and Foster, 2013). These results suggest that the hippocampus is critical for the state-transition part of an animal's environment model, and that it is part of a system that uses the model to simulate possible future state sequences to assess the consequences of possible courses of action: a form of planning.

The results described above add to a voluminous literature on neural mechanisms underlying goal-directed, or model-based, learning and decision making, but many questions remain unanswered. For example, how can areas as structurally similar as the DLS and DMS be essential components of modes of learning and behavior that are as different as model-free and model-based algorithms? Are separate structures responsible for (what we call) the transition and reward components of an environment model? Is all planning conducted at decision time via simulations of possible future courses of action as the forward sweeping activity in the hippocampus suggests? In other words, is all planning something like a rollout algorithm (Section 8.10)? Or are models sometimes engaged in the background to refine or recompute value information as illustrated by the Dyna architecture (Section 8.2)? How does the brain arbitrate between the use of the habit and goal-directed systems? Is there, in fact, a clear separation between the neural substrates of these systems?

The evidence is not pointing to a positive answer to this last question. Summarizing the situation, Doll, Simon, and Daw (2012) wrote that "model-based influences appear ubiquitous more or less wherever the brain processes reward information," and this is true even in the regions thought to be critical for model-free learning. This includes the dopamine signals themselves, which can exhibit the influence of model-based information in addition to the reward prediction errors thought to be the basis of model-free processes.

Continuing neuroscience research informed by reinforcement learning's model-free and model-based distinction has the potential to sharpen our understanding of habitual and goal-directed processes in the brain. A better grasp of these neural mechanisms may lead to algorithms combining model-free and model-based methods in ways that have not yet been explored in computational reinforcement learning.

15.12 Addiction

Understanding the neural basis of drug abuse is a high-priority goal of neuroscience with the potential to produce new treatments for this serious public health problem. One view is that drug craving is the result of the same motivation and learning processes that lead us to seek natural rewarding experiences that serve our biological needs. Addictive substances, by being intensely reinforcing, effectively co-opt our natural mechanisms of learning and decision making. This is plausible given that many—though not all—drugs of abuse increase levels of dopamine either directly or indirectly in regions around terminals of dopamine neuron axons in the striatum, a brain structure firmly implicated in normal reward-based learning (Section 15.7). But the self-destructive behavior associated with drug addiction is not characteristic of normal learning. What is different about dopamine-mediated learning when the reward is the result of an addictive drug? Is addiction the result of normal learning in response to substances that were largely unavailable throughout our evolutionary history, so that evolution could not select against their damaging effects? Or do addictive substances somehow interfere with normal dopamine-mediated learning?

The reward prediction error hypothesis of dopamine neuron activity and its connection to TD learning are the basis of a model due to Redish (2004) of some—but certainly not all—features of addiction. The model is based on the observation that administration of cocaine and some other addictive drugs produces a transient increase in dopamine. In the model, this dopamine surge is assumed to increase the TD error, δ, in a way that cannot be cancelled out by changes in the value function. In other words, whereas δ is reduced to the degree that a normal reward is predicted by antecedent events (Section 15.6), the contribution to δ due to an addictive stimulus does not decrease as the reward signal becomes predicted: drug rewards cannot be "predicted away." The model does this by preventing δ from ever becoming negative when the reward signal is due to an addictive drug, thus eliminating the error-correcting feature of TD learning for states associated with administration of the drug. The result is that the values of these states increase without bound, making actions leading to these states preferred above all others.

Addictive behavior is much more complicated than this result from Redish's model, but the model's main idea may be a piece of the puzzle. Or the model might be misleading. Dopamine appears not to play a critical role in all forms of addiction, and not everyone is equally susceptible to developing addictive behavior. Moreover, the model does not include the changes in many circuits and brain regions that accompany chronic drug taking, for example, changes that lead to a drug's diminishing effect with repeated use. It is also likely that addiction involves model-based processes. Still, Redish's model illustrates how reinforcement learning theory can be enlisted in the effort to understand a major health problem. In a similar manner, reinforcement learning theory has been influential in the development of the new field of computational psychiatry, which aims to improve understanding of mental disorders through mathematical and computational methods.

15.13 Summary

The neural pathways involved in the brain's reward system are complex and incompletely understood, but neuroscience research directed toward understanding these pathways and their roles in behavior is progressing rapidly. This research is revealing striking correspondences between the brain's reward system and the theory of reinforcement learning as presented in this book.

The *reward prediction error hypothesis of dopamine neuron activity* was proposed by scientists who recognized striking parallels between the behavior of TD errors and the activity of neurons that produce dopamine, a neurotransmitter essential in mammals for reward-related learning and behavior. Experiments conducted in the late 1980s and 1990s in the laboratory of neuroscientist Wolfram Schultz showed that dopamine neurons respond to rewarding events with substantial bursts of activity, called phasic responses, only if the animal does not expect those events, suggesting that dopamine neurons are signaling reward prediction errors instead of reward itself. Further, these experiments showed that as an animal learns to predict a rewarding event on the basis of preceding sensory cues, the phasic activity of dopamine neurons shifts to earlier predictive cues while decreasing to later predictive cues. This parallels the backing-up effect of the TD error as a reinforcement learning agent learns to predict reward.

Other experimental results firmly establish that the phasic activity of dopamine neurons is a reinforcement signal for learning that reaches multiple areas of the brain by means of profusely branching axons of dopamine producing neurons. These results are consistent with the distinction we make between a reward signal, R_t, and a reinforcement signal, which is the TD error δ_t in most of the algorithms we present. Phasic responses of dopamine neurons are reinforcement signals, not reward signals.

A prominent hypothesis is that the brain implements something like an actor–critic algorithm. Two structures in the brain (the dorsal and ventral subdivisions of the striatum), both of which play critical roles in reward-based learning, may function respectively like an actor and a critic. That the TD error is the reinforcement signal for both the actor and the critic fits well with the facts that dopamine neuron axons target both the dorsal and ventral subdivisions of the striatum; that dopamine appears to be

critical for modulating synaptic plasticity in both structures; and that the effect on a target structure of a neuromodulator such as dopamine depends on properties of the target structure and not just on properties of the neuromodulator.

The actor and the critic can be implemented by ANNs consisting of neuron-like units having learning rules based on the policy-gradient actor–critic method described in Section 13.5. Each connection in these networks is like a synapse between neurons in the brain, and the learning rules correspond to rules governing how synaptic efficacies change as functions of the activities of the presynaptic and the postsynaptic neurons, together with neuromodulatory input corresponding to input from dopamine neurons. In this setting, each synapse has its own eligibility trace that records past activity involving that synapse. The only difference between the actor and critic learning rules is that they use different kinds of eligibility traces: the critic unit's traces are *non-contingent* because they do not involve the critic unit's output, whereas the actor unit's traces are *contingent* because in addition to the actor unit's input, they depend on the actor unit's output. In the hypothetical implementation of an actor–critic system in the brain, these learning rules respectively correspond to rules governing plasticity of corticostriatal synapses that convey signals from the cortex to the principal neurons in the dorsal and ventral striatal subdivisions, synapses that also receive inputs from dopamine neurons.

The learning rule of an actor unit in the actor–critic network closely corresponds to *reward-modulated spike-timing-dependent plasticity.* In spike-timing-dependent plasticity (STDP), the relative timing of pre- and postsynaptic activity determines the direction of synaptic change. In reward-modulated STDP, changes in synapses in addition depend on a neuromodulator, such as dopamine, arriving within a time window that can last up to 10 seconds after the conditions for STDP are met. Evidence is accumulating that reward-modulated STDP occurs at corticostriatal synapses, where the actor's learning takes place in the hypothetical neural implementation of an actor–critic system, adds to the plausibility of the hypothesis that something like an actor–critic system exists in the brains of some animals.

The idea of synaptic eligibility and basic features of the actor learning rule derive from Klopf's hypothesis of the "hedonistic neuron" (Klopf, 1972, 1981). He conjectured that individual neurons seek to obtain reward and to avoid punishment by adjusting the efficacies of their synapses on the basis of rewarding or punishing consequences of their action potentials. A neuron's activity can affect its later input because the neuron is embedded in many feedback loops, some within the animal's nervous system and body and others passing through the animal's external environment. Klopf's idea of eligibility is that synapses are temporarily marked as eligible for modification if they participated in the neuron's firing (making this the contingent form of eligibility trace). A synapse's efficacy is modified if a reinforcing signal arrives while the synapse is eligible. We alluded to the chemotactic behavior of a bacterium as an example of a single cell that directs its movements in order to seek some molecules and to avoid others.

A conspicuous feature of the dopamine system is that fibers releasing dopamine project widely to multiple parts of the brain. Although it is likely that only some populations of dopamine neurons broadcast the same reinforcement signal, if this signal reaches the synapses of many neurons involved in actor-type learning, then the situation can

be modeled as a *team problem*. In this type of problem, each agent in a collection of reinforcement learning agents receives the same reinforcement signal, where that signal depends on the activities of all members of the collection, or team. If each team member uses a sufficiently capable learning algorithm, the team can learn collectively to improve performance of the entire team as evaluated by the globally-broadcast reinforcement signal, even if the team members do not directly communicate with one another. This is consistent with the wide dispersion of dopamine signals in the brain and provides a neurally plausible alternative to the widely-used error-backpropagation method for training multilayer networks.

The distinction between model-free and model-based reinforcement learning is helping neuroscientists investigate the neural bases of habitual and goal-directed learning and decision making. Research so far points to their being some brain regions more involved in one type of process than the other, but the picture remains unclear because model-free and model-based processes do not appear to be neatly separated in the brain. Many questions remain unanswered. Perhaps most intriguing is evidence that the hippocampus, a structure traditionally associated with spatial navigation and memory, appears to be involved in simulating possible future courses of action as part of an animal's decision-making process. This suggests that it is part of a system that uses an environment model for planning.

Reinforcement learning theory is also influencing thinking about neural processes underlying drug abuse. A model of some features of drug addiction is based on the reward prediction error hypothesis. It proposes that an addicting stimulant, such as cocaine, destabilizes TD learning to produce unbounded growth in the values of actions associated with drug intake. This is far from a complete model of addiction, but it illustrates how a computational perspective suggests theories that can be tested with further research. The new field of computational psychiatry similarly focuses on the use of computational models, some derived from reinforcement learning, to better understand mental disorders.

This chapter only touched the surface of how the neuroscience of reinforcement learning and the development of reinforcement learning in computer science and engineering have influenced one another. Most features of reinforcement learning algorithms owe their design to purely computational considerations, but some have been influenced by hypotheses about neural learning mechanisms. Remarkably, as experimental data has accumulated about the brain's reward processes, many of the purely computationally-motivated features of reinforcement learning algorithms are turning out to be consistent with neuroscience data. Other features of computational reinforcement learning, such as eligibility traces and the ability of teams of reinforcement learning agents to learn to act collectively under the influence of a globally-broadcast reinforcement signal, may also turn out to parallel experimental data as neuroscientists continue to unravel the neural basis of reward-based animal learning and behavior.

Bibliographical and Historical Remarks

The number of publications treating parallels between the neuroscience of learning and decision making and the approach to reinforcement learning presented in this book is enormous. We can cite only a small selection. Niv (2009), Dayan and Niv (2008), Gimcher (2011), Ludvig, Bellemare, and Pearson (2011), and Shah (2012) are good places to start.

Together with economics, evolutionary biology, and mathematical psychology, reinforcement learning theory is helping to formulate quantitative models of the neural mechanisms of choice in humans and non-human primates. With its focus on learning, this chapter only lightly touches upon the neuroscience of decision making. Glimcher (2003) introduced the field of "neuroeconomics," in which reinforcement learning contributes to the study of the neural basis of decision making from an economics perspective. See also Glimcher and Fehr (2013). The text on computational and mathematical modeling in neuroscience by Dayan and Abbott (2001) includes reinforcement learning's role in these approaches. Sterling and Laughlin (2015) examined the neural basis of learning in terms of general design principles that enable efficient adaptive behavior.

15.1 There are many good expositions of basic neuroscience. Kandel, Schwartz, Jessell, Siegelbaum, and Hudspeth (2013) is an authoritative and very comprehensive source.

15.2 Berridge and Kringelbach (2008) reviewed the neural basis of reward and pleasure, pointing out that reward processing has many dimensions and involves many neural systems. Space prevents discussion of the influential research of Berridge and Robinson (1998), who distinguish between the hedonic impact of a stimulus, which they call "liking," and the motivational effect, which they call "wanting." Hare, O'Doherty, Camerer, Schultz, and Rangel (2008) examined the neural basis of value-related signals from an economic perspective, distinguishing between goal values, decision values, and prediction errors. Decision value is goal value minus action cost. See also Rangel, Camerer, and Montague (2008), Rangel and Hare (2010), and Peters and Büchel (2010).

15.3 The reward prediction error hypothesis of dopamine neuron activity is most prominently discussed by Schultz, Dayan, and Montague (1997). The hypothesis was first explicitly put forward by Montague, Dayan, and Sejnowski (1996). As they stated the hypothesis, it referred to reward prediction errors (RPEs) but not specifically to TD errors; however, their development of the hypothesis made it clear that they were referring to TD errors. The earliest recognition of the TD-error/dopamine connection of which we are aware is that of Montague, Dayan, Nowlan, Pouget, and Sejnowski (1993), who proposed a TD-error-modulated Hebbian learning rule motivated by results on dopamine signaling from Schultz's group. The connection was also pointed out in an abstract by Quartz, Dayan, Montague, and Sejnowski (1992). Montague and Sejnowski (1994) emphasized the importance of prediction in the brain and outlined how predictive Hebbian learning modulated by TD errors could be implemented via a diffuse neuromodulatory system, such as the dopamine system. Friston, Tononi, Reeke, Sporns,

and Edelman (1994) presented a model of value-dependent learning in the brain in which synaptic changes are mediated by a TD-like error provided by a global neuromodulatory signal (although they did not single out dopamine). Montague, Dayan, Person, and Sejnowski (1995) presented a model of honeybee foraging using the TD error. The model is based on research by Hammer, Menzel, and colleagues (Hammer and Menzel, 1995; Hammer, 1997) showing that the neuromodulator octopamine acts as a reinforcement signal in the honeybee. Montague et al. (1995) pointed out that dopamine likely plays a similar role in the vertebrate brain. Barto (1995a) related the actor–critic architecture to basal-ganglionic circuits and discussed the relationship between TD learning and the main results from Schultz's group. Houk, Adams, and Barto (1995) suggested how TD learning and the actor–critic architecture might map onto the anatomy, physiology, and molecular mechanism of the basal ganglia. Doya and Sejnowski (1998) extended their earlier paper on a model of birdsong learning (Doya and Sejnowski, 1995) by including a TD-like error identified with dopamine to reinforce the selection of auditory input to be memorized. O'Reilly and Frank (2006) and O'Reilly, Frank, Hazy, and Watz (2007) argued that phasic dopamine signals are RPEs but not TD errors. In support of their theory they cited results with variable interstimulus intervals that do not match predictions of a simple TD model, as well as the observation that higher-order conditioning beyond second-order conditioning is rarely observed, while TD learning is not so limited. Dayan and Niv (2008) discussed "the good, the bad, and the ugly" of how reinforcement learning theory and the reward prediction error hypothesis align with experimental data. Glimcher (2011) reviewed the empirical findings that support the reward prediction error hypothesis and emphasized the significance of the hypothesis for contemporary neuroscience.

15.4 Graybiel (2000) is a brief primer on the basal ganglia. The experiments mentioned that involve optogenetic activation of dopamine neurons were conducted by Tsai, Zhang, Adamantidis, Stuber, Bonci, de Lecea, and Deisseroth (2009), Steinberg, Keiflin, Boivin, Witten, Deisseroth, and Janak (2013), and Claridge-Chang, Roorda, Vrontou, Sjulson, Li, Hirsh, and Miesenböck (2009). Fiorillo, Yun, and Song (2013), Lammel, Lim, and Malenka (2014), and Saddoris, Cacciapaglia, Wightmman, and Carelli (2015) are among studies showing that the signaling properties of dopamine neurons are specialized for different target regions. RPE-signaling neurons may belong to one among multiple populations of dopamine neurons having different targets and subserving different functions. Eshel, Tian, Bukwich, and Uchida (2016) found homogeneity of reward prediction error responses of dopamine neurons in the lateral VTA during classical conditioning in mice, though their results do not rule out response diversity across wider areas. Gershman, Pesaran, and Daw (2009) studied reinforcement learning tasks that can be decomposed into independent components with separate reward signals, finding evidence in human neuroimaging data suggesting that the brain exploits this kind of structure.

15.5 Schultz's 1998 survey article is a good entrée into the very extensive literature on reward predicting signaling of dopamine neurons. Berns, McClure, Pagnoni, and Montague (2001), Breiter, Aharon, Kahneman, Dale, and Shizgal (2001), Pagnoni, Zink, Montague, and Berns (2002), and O'Doherty, Dayan, Friston, Critchley, and Dolan (2003) described functional brain imaging studies supporting the existence of signals like TD errors in the human brain.

15.6 This section roughly follows Barto (1995a) in explaining how TD errors mimic the main results from Schultz's group on the phasic responses of dopamine neurons.

15.7 This section is largely based on Takahashi, Schoenbaum, and Niv (2008) and Niv (2009). To the best of our knowledge, Barto (1995a) and Houk, Adams, and Barto (1995) first speculated about possible implementations of actor–critic algorithms in the basal ganglia. On the basis of functional magnetic resonance imaging of human subjects while engaged in instrumental conditioning, O'Doherty, Dayan, Schultz, Deichmann, Friston, and Dolan (2004) suggested that the actor and the critic are most likely located respectively in the dorsal and ventral striatum. Gershman, Moustafa, and Ludvig (2014) focused on how time is represented in reinforcement learning models of the basal ganglia, discussing evidence for, and implications of, various computational approaches to time representation.

The hypothetical neural implementation of the actor–critic architecture described in this section includes very little detail about known basal ganglia anatomy and physiology. In addition to the more detailed hypothesis of Houk, Adams, and Barto (1995), a number of other hypotheses include more specific connections to anatomy and physiology and are claimed to explain additional data. These include hypotheses proposed by Suri and Schultz (1998, 1999), Brown, Bullock, and Grossberg (1999), Contreras-Vidal and Schultz (1999), Suri, Bargas, and Arbib (2001), O'Reilly and Frank (2006), and O'Reilly, Frank, Hazy, and Watz (2007). Joel, Niv, and Ruppin (2002) critically evaluated the anatomical plausibility of several of these models and present an alternative intended to accommodate some neglected features of basal ganglionic circuitry.

15.8 The actor learning rule discussed here is more complicated than the one in the early actor–critic network of Barto et al. (1983). Actor-unit eligibility traces in that network were traces of just $A_t \times \mathbf{x}(S_t)$ instead of the full $(A_t - \pi(1|S_t, \boldsymbol{\theta}))\mathbf{x}(S_t)$. That work did not benefit from the policy-gradient theory presented in Chapter 13 or the contributions of Williams (1986, 1992), who showed how an ANN of Bernoulli-logistic units could implement a policy-gradient method.

Reynolds and Wickens (2002) proposed a three-factor rule for synaptic plasticity in the corticostriatal pathway in which dopamine modulates changes in corticostriatal synaptic efficacy. They discussed the experimental support for this kind of learning rule and its possible molecular basis. The definitive demonstration of spike-timing-dependent plasticity (STDP) is attributed to Markram, Lübke, Frotscher, and Sakmann (1997), with evidence from earlier experiments

by Levy and Steward (1983) and others that the relative timing of pre- and postsynaptic spikes is critical for inducing changes in synaptic efficacy. Rao and Sejnowski (2001) suggested how STDP could be the result of a TD-like mechanism at synapses with non-contingent eligibility traces lasting about 10 milliseconds. Dayan (2002) commented that this would require an error as in Sutton and Barto's (1981a) early model of classical conditioning and not a true TD error. Representative publications from the extensive literature on reward-modulated STDP are Wickens (1990), Reynolds and Wickens (2002), and Calabresi, Picconi, Tozzi and Di Filippo (2007). Pawlak and Kerr (2008) showed that dopamine is necessary to induce STDP at the corticostriatal synapses of medium spiny neurons. See also Pawlak, Wickens, Kirkwood, and Kerr (2010). Yagishita, Hayashi-Takagi, Ellis-Davies, Urakubo, Ishii, and Kasai (2014) found that dopamine promotes spine enlargement of the medium spiny neurons of mice only during a time window of from 0.3 to 2 seconds after STDP stimulation. Izhikevich (2007) proposed and explored the idea of using STDP timing conditions to trigger contingent eligibility traces. Frémaux, Sprekeler, and Gerstner (2010) proposed theoretical conditions for successful learning by rules based on reward-modulated STDP.

15.9 Klopf's hedonistic neuron hypothesis (Klopf 1972, 1982) inspired our actor–critic algorithm implemented as an ANN with a single neuron-like unit, called the actor unit, implementing a Law-of-Effect-like learning rule (Barto, Sutton, and Anderson, 1983). Ideas related to Klopf's synaptically-local eligibility have been proposed by others. Crow (1968) proposed that changes in the synapses of cortical neurons are sensitive to the consequences of neural activity. Emphasizing the need to address the time delay between neural activity and its consequences in a reward-modulated form of synaptic plasticity, he proposed a contingent form of eligibility, but associated with entire neurons instead of individual synapses. According to his hypothesis, a wave of neuronal activity

> leads to a short-term change in the cells involved in the wave such that they are picked out from a background of cells not so activated. ... such cells are rendered sensitive by the short-term change to a reward signal ... in such a way that if such a signal occurs before the end of the decay time of the change the synaptic connexions between the cells are made more effective. (Crow, 1968)

Crow argued against previous proposals that reverberating neural circuits play this role by pointing out that the effect of a reward signal on such a circuit would "...establish the synaptic connexions leading to the reverberation (that is to say, those involved in activity at the time of the reward signal) and not those on the path which led to the adaptive motor output." Crow further postulated that reward signals are delivered via a "distinct neural fiber system," presumably the one into which Olds and Milner (1954) tapped, that would transform synaptic connections "from a short into a long-term form."

In another farsighted hypothesis, Miller proposed a Law-of-Effect-like learning rule that includes synaptically-local contingent eligibility traces:

> ... it is envisaged that in a particular sensory situation neurone B, by chance, fires a 'meaningful burst' of activity, which is then translated into motor acts, which then change the situation. It must be supposed that the meaningful burst has an influence, *at the neuronal level*, on all of its own synapses which are active at the time ... thereby making a preliminary selection of the synapses to be strengthened, though not yet actually strengthening them. ...The strengthening signal ... makes the final selection ... and accomplishes the definitive change in the appropriate synapses. (Miller, 1981, p. 81)

Miller's hypothesis also included a critic-like mechanism, which he called a "sensory analyzer unit," that worked according to classical conditioning principles to provide reinforcement signals to neurons so that they would learn to move from lower- to higher-valued states, thus anticipating the use of the TD error as a reinforcement signal in the actor–critic architecture. Miller's idea not only parallels Klopf's (with the exception of its explicit invocation of a distinct "strengthening signal"), it also anticipated the general features of reward-modulated STDP.

A related though different idea, which Seung (2003) called the "hedonistic synapse," is that synapses individually adjust the probability that they release neurotransmitter in the manner of the Law of Effect: if reward follows release, the release probability increases, and decreases if reward follows failure to release. This is essentially the same as the learning scheme Minsky used in his 1954 Princeton PhD dissertation, where he called the synapse-like learning element a SNARC (Stochastic Neural-Analog Reinforcement Calculator). Contingent eligibility is involved in these ideas too, although it is contingent on the activity of an individual synapse instead of the postsynaptic neuron. Also related is the proposal of Unnikrishnan and Venugopal (1994) that uses the correlation-based method of Harth and Tzanakou (1974) to adjust ANN weights.

Frey and Morris (1997) proposed the idea of a "synaptic tag" for the induction of long-lasting strengthening of synaptic efficacy. Though not unlike Klopf's eligibility, their tag was hypothesized to consist of a temporary strengthening of a synapse that could be transformed into a long-lasting strengthening by subsequent neuron activation. The model of O'Reilly and Frank (2006) and O'Reilly, Frank, Hazy, and Watz (2007) uses working memory to bridge temporal intervals instead of eligibility traces. Wickens and Kotter (1995) discuss possible mechanisms for synaptic eligibility. He, Huertas, Hong, Tie, Hell, Shouval, Kirkwood (2015) provide evidence supporting the existence of contingent eligibility traces in synapses of cortical neurons with time courses like those of the eligibility traces Klopf postulated.

The metaphor of a neuron using a learning rule related to bacterial chemotaxis was discussed by Barto (1989). Koshland's extensive study of bacterial chemotaxis was in part motivated by similarities between features of bacteria and features of neurons (Koshland, 1980). See also Berg (1975). Shimansky (2009) proposed a

synaptic learning rule somewhat similar to Seung's mentioned above in which each synapse individually acts like a chemotactic bacterium. In this case a collection of synapses "swims" toward attractants in the high-dimensional space of synaptic weight values. Montague, Dayan, Person, and Sejnowski (1995) proposed a chemotactic-like model of the bee's foraging behavior involving the neuromodulator octopamine.

15.10 Research on the behavior of reinforcement learning agents in team and game problems has a long history roughly occurring in three phases. To the best of our knowledge, the first phase began with investigations by the Russian mathematician and physicist M. L. Tsetlin. A collection of his work was published as Tsetlin (1973) after his death in 1966. Our Sections 1.7 and 4.8 refer to his study of learning automata in connection to bandit problems. The Tsetlin collection also includes studies of learning automata in team and game problems, which led to later work in this area using stochastic learning automata as described by Narendra and Thathachar (1974, 1989), Viswanathan and Narendra (1974), Lakshmivarahan and Narendra (1982), Narendra and Wheeler (1983), and Thathachar and Sastry (2002). Thathachar and Sastry (2011) is a more recent comprehensive account. These studies were mostly restricted to non-associative learning automata, meaning that they did not address associative, or contextual, bandit problems (Section 2.9).

The second phase began with the extension of learning automata to the associative, or contextual, case. Barto, Sutton, and Brouwer (1981) and Barto and Sutton (1981b) experimented with associative stochastic learning automata in single-layer ANNs to which a global reinforcement signal was broadcast. The learning algorithm was an associative extension of the Alopex algorithm of Harth and Tzanakou (1974). Barto et al. called neuron-like elements implementing this kind of learning *associative search elements* (ASEs). Barto and Anandan (1985) introduced an associative reinforcement learning algorithm called the *associative reward-penalty* (A_{R-P}) algorithm. They proved a convergence result by combining theory of stochastic learning automata with theory of pattern classification. Barto (1985, 1986) and Barto and Jordan (1987) described results with teams of A_{R-P} units connected into multi-layer ANNs, showing that they could learn nonlinear functions, such as XOR and others, with a globally-broadcast reinforcement signal. Barto (1985) extensively discussed this approach to ANNs and how this type of learning rule is related to others in the literature at that time. Williams (1992) mathematically analyzed and broadened this class of learning rules and related their use to the error backpropagation method for training multilayer ANNs. Williams (1988) described several ways that backpropagation and reinforcement learning can be combined for training ANNs. Williams (1992) showed that a special case of the A_{R-P} algorithm is a REINFORCE algorithm, although better results were obtained with the general A_{R-P} algorithm (Barto, 1985).

The third phase of interest in teams of reinforcement learning agents was influenced by increased understanding of the role of dopamine as a widely broadcast neuromodulator and speculation about the existence of reward-modulated STDP. Much more so than earlier research, this research considers details of synaptic plasticity and other constraints from neuroscience. Publications include the following (chronologically and alphabetically): Bartlett and Baxter (1999, 2000), Xie and Seung (2004), Baras and Meir (2007), Farries and Fairhall (2007), Florian (2007), Izhikevich (2007), Pecevski, Maass, and Legenstein (2008), Legenstein, Pecevski, and Maass (2008), Kolodziejski, Porr, and Wörgötter (2009), Urbanczik and Senn (2009), and Vasilaki, Frémaux, Urbanczik, Senn, and Gerstner (2009). Nowé, Vrancx, and De Hauwere (2012) reviewed more recent developments in the wider field of multi-agent reinforcement learning

15.11 Yin and Knowlton (2006) reviewed findings from outcome-devaluation experiments with rodents supporting the view that habitual and goal-directed behavior (as psychologists use the phrase) are respectively most associated with processing in the dorsolateral striatum (DLS) and the dorsomedial striatum (DMS). Results of functional imaging experiments with human subjects in the outcome-devaluation setting by Valentin, Dickinson, and O'Doherty (2007) suggest that the orbitofrontal cortex (OFC) is an important component of goal-directed choice. Single unit recordings in monkeys by Padoa-Schioppa and Assad (2006) support the role of the OFC in encoding values guiding choice behavior. Rangel, Camerer, and Montague (2008) and Rangel and Hare (2010) reviewed findings from the perspective of neuroeconomics about how the brain makes goal-directed decisions. Pezzulo, van der Meer, Lansink, and Pennartz (2014) reviewed the neuroscience of internally generated sequences and presented a model of how these mechanisms might be components of model-based planning. Daw and Shohamy (2008) proposed that while dopamine signaling connects well to habitual, or model-free, behavior, other processes are involved in goal-directed, or model-based, behavior. Data from experiments by Bromberg-Martin, Matsumoto, Hong, and Hikosaka (2010) indicate that dopamine signals contain information pertinent to both habitual and goal-directed behavior. Doll, Simon, and Daw (2012) argued that there may not a clear separation in the brain between mechanisms that subserve habitual and goal-directed learning and choice.

15.12 Keiflin and Janak (2015) reviewed connections between TD errors and addiction. Nutt, Lingford-Hughes, Erritzoe, and Stokes (2015) critically evaluated the hypothesis that addiction is due to a disorder of the dopamine system. Montague, Dolan, Friston, and Dayan (2012) outlined the goals and early efforts in the field of computational psychiatry, and Adams, Huys, and Roiser (2015) reviewed more recent progress.

Chapter 16

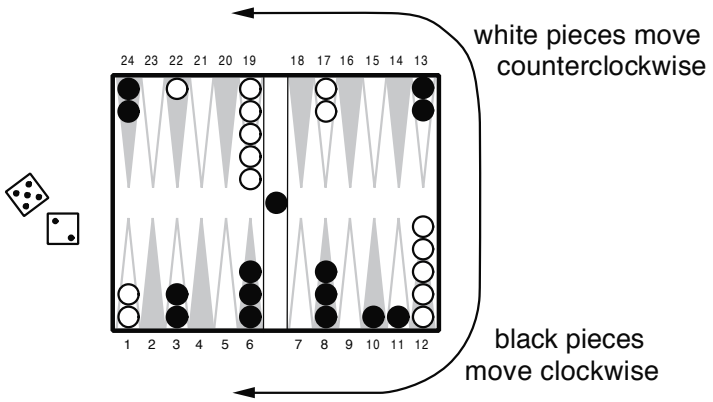
Applications and Case Studies

In this chapter we present a few case studies of reinforcement learning. Several of these are substantial applications of potential economic significance. One, Samuel's checkers player, is primarily of historical interest. Our presentations are intended to illustrate some of the trade-offs and issues that arise in real applications. For example, we emphasize how domain knowledge is incorporated into the formulation and solution of the problem. We also highlight the representation issues that are so often critical to successful applications. The algorithms used in some of these case studies are substantially more complex than those we have presented in the rest of the book. Applications of reinforcement learning are still far from routine and typically require as much art as science. Making applications easier and more straightforward is one of the goals of current research in reinforcement learning.

16.1 TD-Gammon

One of the most impressive applications of reinforcement learning to date is that by Gerald Tesauro to the game of backgammon (Tesauro, 1992, 1994, 1995, 2002). Tesauro's program, *TD-Gammon*, required little backgammon knowledge, yet learned to play extremely well, near the level of the world's strongest grandmasters. The learning algorithm in TD-Gammon was a straightforward combination of the TD(λ) algorithm and nonlinear function approximation using a multilayer artificial neural network (ANN) trained by backpropagating TD errors.

Backgammon is a major game in the sense that it is played throughout the world, with numerous tournaments and regular world championship matches. It is in part a game of chance, and it is a popular vehicle for waging significant sums of money. There are probably more professional backgammon players than there are professional chess players. The game is played with 15 white and 15 black pieces on a board of 24 locations, called *points*. To the right on the next page is shown a typical position early in the game, seen from the perspective of the white player. White here has just rolled the dice and obtained a 5 and a 2. This means that he can move one of his pieces 5 steps and one



A backgammon position

(possibly the same piece) 2 steps. For example, he could move two pieces from the 12 point, one to the 17 point, and one to the 14 point. White's objective is to advance all of his pieces into the last quadrant (points 19–24) and then off the board. The first player to remove all his pieces wins. One complication is that the pieces interact as they pass each other going in different directions. For example, if it were black's move, he could use the dice roll of 2 to move a piece from the 24 point to the 22 point, "hitting" the white piece there. Pieces that have been hit are placed on the "bar" in the middle of the board (where we already see one previously hit black piece), from whence they reenter the race from the start. However, if there are two pieces on a point, then the opponent cannot move to that point; the pieces are protected from being hit. Thus, white cannot use his 5–2 dice roll to move either of his pieces on the 1 point, because their possible resulting points are occupied by groups of black pieces. Forming contiguous blocks of occupied points to block the opponent is one of the elementary strategies of the game.

Backgammon involves several further complications, but the above description gives the basic idea. With 30 pieces and 24 possible locations (26, counting the bar and off-the-board) it should be clear that the number of possible backgammon positions is enormous, far more than the number of memory elements one could have in any physically realizable computer. The number of moves possible from each position is also large. For a typical dice roll there might be 20 different ways of playing. In considering future moves, such as the response of the opponent, one must consider the possible dice rolls as well. The result is that the game tree has an effective branching factor of about 400. This is far too large to permit effective use of the conventional heuristic search methods that have proved so effective in games like chess and checkers.

On the other hand, the game is a good match to the capabilities of TD learning methods. Although the game is highly stochastic, a complete description of the game's state is available at all times. The game evolves over a sequence of moves and positions until finally ending in a win for one player or the other, ending the game. The outcome can be interpreted as a final reward to be predicted. On the other hand, the theoretical results we have described so far cannot be usefully applied to this task. The number of states is so large that a lookup table cannot be used, and the opponent is a source of uncertainty and time variation.

TD-Gammon used a nonlinear form of TD(λ). The estimated value, $\hat{v}(s,\mathbf{w})$, of any state (board position) s was meant to estimate the probability of winning starting from state s. To achieve this, rewards were defined as zero for all time steps except those on which the game is won. To implement the value function, TD-Gammon used a standard multilayer ANN, much like that shown to the right on the next page. (The real

network had two additional units in its final layer to estimate the probability of each player's winning in a special way called a "gammon" or "backgammon.") The network consisted of a layer of input units, a layer of hidden units, and a final output unit. The input to the network was a representation of a backgammon position, and the output was an estimate of the value of that position.

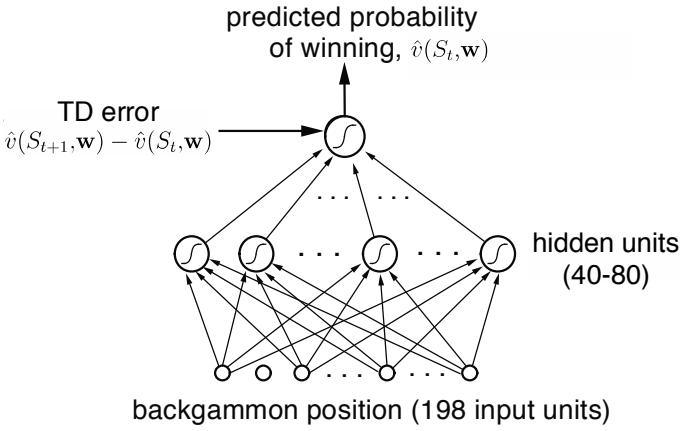


Figure 16.1: The TD-Gammon ANN

In the first version of TD-Gammon, TD-Gammon 0.0, backgammon positions were represented to the network in a relatively direct way that involved little backgammon knowledge. It did, however, involve substantial knowledge of how ANNs work and how information is best presented to them. It is instructive to note the exact representation Tesauro chose. There were a total of 198 input units to the network. For each point on the backgammon board, four units indicated the number of white pieces on the point. If there were no white pieces, then all four units took on the value zero. If there was one piece, then the first unit took on the value 1. This encoded the elementary concept of a "blot," i.e., a piece that can be hit by the opponent. If there were two or more pieces, then the second unit was set to 1. This encoded the basic concept of a "made point" on which the opponent cannot land. If there were exactly three pieces on the point, then the third unit was set to 1. This encoded the basic concept of a "single spare," i.e., an extra piece in addition to the two pieces that made the point. Finally, if there were more than three pieces, the fourth unit was set to a value proportionate to the number of additional pieces beyond three. Letting n denote the total number of pieces on the point, if $n > 3$, then the fourth unit took on the value $(n-3)/2$. This encoded a linear representation of "multiple spares" at the given point.

With four units for white and four for black at each of the 24 points, that made a total of 192 units. Two additional units encoded the number of white and black pieces on the bar (each took the value $n/2$, where n is the number of pieces on the bar), and two more encoded the number of black and white pieces already successfully removed from the board (these took the value $n/15$, where n is the number of pieces already borne off). Finally, two units indicated in a binary fashion whether it was white's or black's turn to move. The general logic behind these choices should be clear. Basically, Tesauro tried to represent the position in a straightforward way, while keeping the number of units relatively small. He provided one unit for each conceptually distinct possibility that seemed likely to be relevant, and he scaled them to roughly the same range, in this case between 0 and 1.

Given a representation of a backgammon position, the network computed its estimated value in the standard way. Corresponding to each connection from an input unit to a hidden unit was a real-valued weight. Signals from each input unit were multiplied by

their corresponding weights and summed at the hidden unit. The output, $h(j)$, of hidden unit j was a nonlinear sigmoid function of the weighted sum:

$$h(j) = \sigma\left(\sum_i w_{ij} x_i\right) = \frac{1}{1 + e^{-\sum_i w_{ij} x_i}},$$

where x_i is the value of the ith input unit and w_{ij} is the weight of its connection to the jth hidden unit (all the weights in the network together make up the parameter vector $\mathbf{w}$). The output of the sigmoid is always between 0 and 1, and has a natural interpretation as a probability based on a summation of evidence. The computation from hidden units to the output unit was entirely analogous. Each connection from a hidden unit to the output unit had a separate weight. The output unit formed the weighted sum and then passed it through the same sigmoid nonlinearity.

TD-Gammon used the semi-gradient form of the TD(λ) algorithm described in Section 12.2, with the gradients computed by the error backpropagation algorithm (Rumelhart, Hinton, and Williams, 1986). Recall that the general update rule for this case is

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha\Big[R_{t+1} + \gamma\hat{v}(S_{t+1},\mathbf{w}_t) - \hat{v}(S_t,\mathbf{w}_t)\Big]\mathbf{z}_t, \tag{16.1}$$

where $\mathbf{w}_t$ is the vector of all modifiable parameters (in this case, the weights of the network) and $\mathbf{z}_t$ is a vector of eligibility traces, one for each component of $\mathbf{w}_t$, updated by

$$\mathbf{z}_t \doteq \gamma\lambda\mathbf{z}_{t-1} + \nabla\hat{v}(S_t,\mathbf{w}_t),$$

with $\mathbf{z}_0 \doteq \mathbf{0}$. The gradient in this equation can be computed efficiently by the backpropagation procedure. For the backgammon application, in which $\gamma = 1$ and the reward is always zero except upon winning, the TD error portion of the learning rule is usually just $\hat{v}(S_{t+1},\mathbf{w}) - \hat{v}(S_t,\mathbf{w})$, as suggested in Figure 16.1.

To apply the learning rule we need a source of backgammon games. Tesauro obtained an unending sequence of games by playing his learning backgammon player against itself. To choose its moves, TD-Gammon considered each of the 20 or so ways it could play its dice roll and the corresponding positions that would result. The resulting positions are *afterstates* as discussed in Section 6.8. The network was consulted to estimate each of their values. The move was then selected that would lead to the position with the highest estimated value. Continuing in this way, with TD-Gammon making the moves for both sides, it was possible to easily generate large numbers of backgammon games. Each game was treated as an episode, with the sequence of positions acting as the states, $S_0, S_1, S_2, \ldots$. Tesauro applied the nonlinear TD rule (16.1) fully incrementally, that is, after each individual move.

The weights of the network were set initially to small random values. The initial evaluations were thus entirely arbitrary. Because the moves were selected on the basis of these evaluations, the initial moves were inevitably poor, and the initial games often lasted hundreds or thousands of moves before one side or the other won, almost by accident. After a few dozen games however, performance improved rapidly.

After playing about 300,000 games against itself, TD-Gammon 0.0 as described above learned to play approximately as well as the best previous backgammon computer

programs. This was a striking result because all the previous high-performance computer programs had used extensive backgammon knowledge. For example, the reigning champion program at the time was, arguably, *Neurogammon*, another program written by Tesauro that used an ANN but not TD learning. Neurogammon's network was trained on a large training corpus of exemplary moves provided by backgammon experts, and, in addition, started with a set of features specially crafted for backgammon. Neurogammon was a highly tuned, highly effective backgammon program that decisively won the World Backgammon Olympiad in 1989. TD-Gammon 0.0, on the other hand, was constructed with essentially zero backgammon knowledge. That it was able to do as well as Neurogammon and all other approaches is striking testimony to the potential of self-play learning methods.

The tournament success of TD-Gammon 0.0 with zero expert backgammon knowledge suggested an obvious modification: add the specialized backgammon features but keep the self-play TD learning method. This produced TD-Gammon 1.0. TD-Gammon 1.0 was clearly substantially better than all previous backgammon programs and found serious competition only among human experts. Later versions of the program, TD-Gammon 2.0 (40 hidden units) and TD-Gammon 2.1 (80 hidden units), were augmented with a selective two-ply search procedure. To select moves, these programs looked ahead not just to the positions that would immediately result, but also to the opponent's possible dice rolls and moves. Assuming the opponent always took the move that appeared immediately best for him, the expected value of each candidate move was computed and the best was selected. To save computer time, the second ply of search was conducted only for candidate moves that were ranked highly after the first ply, about four or five moves on average. Two-ply search affected only the moves selected; the learning process proceeded exactly as before. The final versions of the program, TD-Gammon 3.0 and 3.1, used 160 hidden units and a selective three-ply search. TD-Gammon illustrates the combination of learned value functions and decision-time search as in heuristic search and MCTS methods. In follow-on work, Tesauro and Galperin (1997) explored trajectory sampling methods as an alternative to full-width search, which reduced the error rate of live play by large numerical factors (4x–6x) while keeping the think time reasonable at ∼5–10 seconds per move.

During the 1990s, Tesauro was able to play his programs in a significant number of games against world-class human players. A summary of the results is given in Table 16.1.

Program	**Hidden Units**	**Training Games**	**Opponents**	**Results**
TD-Gammon 0.0	40	300,000	other programs	tied for best
TD-Gammon 1.0	80	300,000	Robertie, Magriel, ...	−13 pts / 51 games
TD-Gammon 2.0	40	800,000	various Grandmasters	−7 pts / 38 games
TD-Gammon 2.1	80	1,500,000	Robertie	−1 pt / 40 games
TD-Gammon 3.0	80	1,500,000	Kazaros	+6 pts / 20 games

Table 16.1: Summary of TD-Gammon Results

Based on these results and analyses by backgammon grandmasters (Robertie, 1992; see Tesauro, 1995), TD-Gammon 3.0 appeared to play at close to, or possibly better than, the playing strength of the best human players in the world. Tesauro reported in a subsequent article (Tesauro, 2002) the results of an extensive rollout analysis of the move decisions and doubling decisions of TD-Gammon relative to top human players. The conclusion was that TD-Gammon 3.1 had a "lopsided advantage" in piece-movement decisions, and a "slight edge" in doubling decisions, over top humans.

TD-Gammon had a significant impact on the way the best human players play the game. For example, it learned to play certain opening positions differently than was the convention among the best human players. Based on TD-Gammon's success and further analysis, the best human players now play these positions as TD-Gammon does (Tesauro, 1995). The impact on human play was greatly accelerated when several other self-teaching ANN backgammon programs inspired by TD-Gammon, such as Jellyfish, Snowie, and GNUBackgammon, became widely available. These programs enabled wide dissemination of new knowledge generated by the ANNs, resulting in great improvements in the overall caliber of human tournament play (Tesauro, 2002).

16.2 Samuel's Checkers Player

An important precursor to Tesauro's TD-Gammon was the seminal work of Arthur Samuel (1959, 1967) in constructing programs for learning to play checkers. Samuel was one of the first to make effective use of heuristic search methods and of what we would now call temporal-difference learning. His checkers players are instructive case studies in addition to being of historical interest. We emphasize the relationship of Samuel's methods to modern reinforcement learning methods and try to convey some of Samuel's motivation for using them.

Samuel first wrote a checkers-playing program for the IBM 701 in 1952. His first *learning* program was completed in 1955 and was demonstrated on television in 1956. Later versions of the program achieved good, though not expert, playing skill. Samuel was attracted to game-playing as a domain for studying machine learning because games are less complicated than problems "taken from life" while still allowing fruitful study of how heuristic procedures and learning can be used together. He chose to study checkers instead of chess because its relative simplicity made it possible to focus more strongly on learning.

Samuel's programs played by performing a lookahead search from each current position. They used what we now call heuristic search methods to determine how to expand the search tree and when to stop searching. The terminal board positions of each search were evaluated, or "scored," by a value function, or "scoring polynomial," using linear function approximation. In this and other respects Samuel's work seems to have been inspired by the suggestions of Shannon (1950). In particular, Samuel's program was based on Shannon's minimax procedure to find the best move from the current position. Working backward through the search tree from the scored terminal positions, each position was given the score of the position that would result from the best move, assuming that the machine would always try to maximize the score, while the opponent would always try to

minimize it. Samuel called this the "backed-up score" of the position. When the minimax procedure reached the search tree's root—the current position—it yielded the best move under the assumption that the opponent would be using the same evaluation criterion, shifted to its point of view. Some versions of Samuel's programs used sophisticated search control methods analogous to what are known as "alpha-beta" cutoffs (e.g., see Pearl, 1984).

Samuel used two main learning methods, the simplest of which he called *rote learning*. It consisted simply of saving a description of each board position encountered during play together with its backed-up value determined by the minimax procedure. The result was that if a position that had already been encountered were to occur again as a terminal position of a search tree, the depth of the search was effectively amplified because this position's stored value cached the results of one or more searches conducted earlier. One initial problem was that the program was not encouraged to move along the most direct path to a win. Samuel gave it a "a sense of direction" by decreasing a position's value a small amount each time it was backed up a level (called a ply) during the minimax analysis. "If the program is now faced with a choice of board positions whose scores differ only by the ply number, it will automatically make the most advantageous choice, choosing a low-ply alternative if winning and a high-ply alternative if losing" (Samuel, 1959, p. 80). Samuel found this discounting-like technique essential to successful learning. Rote learning produced slow but continual improvement that was most effective for opening and endgame play. His program became a "better-than-average novice" after learning from many games against itself, a variety of human opponents, and from book games in a supervised learning mode.

Rote learning and other aspects of Samuel's work strongly suggest the essential idea of temporal-difference learning—that the value of a state should equal the value of likely following states. Samuel came closest to this idea in his second learning method, his "learning by generalization" procedure for modifying the parameters of the value function. Samuel's method was the same in concept as that used much later by Tesauro in TD-Gammon. He played his program many games against another version of itself and performed an update after each move. The idea of Samuel's update is suggested by the backup diagram in Figure 16.2. Each open circle represents a position where the program moves next, an *on-move* position, and each solid circle represents a position where the opponent moves next. An update was made to the value of each on-move position after a move by each side, resulting in a second on-move position. The update was toward the minimax value of a search launched from the second on-move position. Thus, the overall effect was that of a backing-up over one full move of real events and then a search over possible events, as suggested by Figure 16.2. Samuel's actual algorithm was significantly more complex than this for computational reasons, but this was the basic idea.

Samuel did not include explicit rewards. Instead, he fixed the weight of the most important feature, the *piece advantage* feature, which measured the number of pieces the program had relative to how many its opponent had, giving higher weight to kings, and including refinements so that it was better to trade pieces when winning than when losing. Thus, the goal of Samuel's program was to improve its piece advantage, which in checkers is highly correlated with winning.

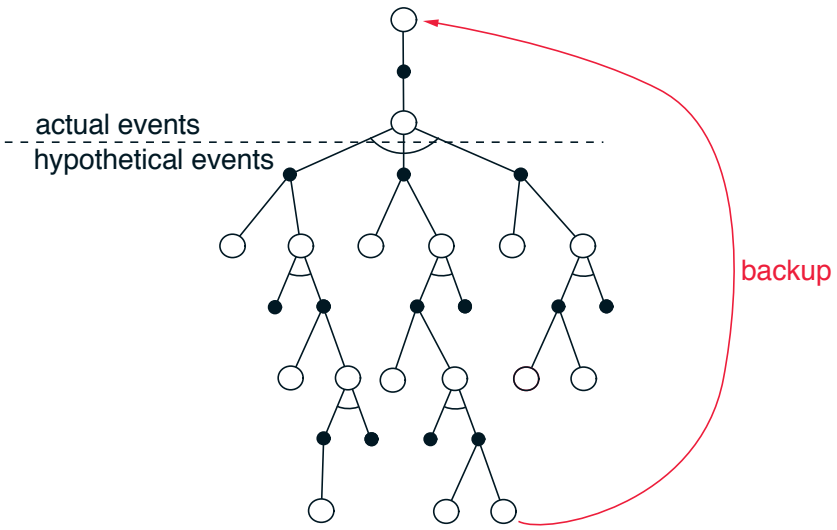


Figure 16.2: The backup diagram for Samuel's checkers player.

However, Samuel's learning method may have been missing an essential part of a sound temporal-difference algorithm. Temporal-difference learning can be viewed as a way of making a value function consistent with itself, and this we can clearly see in Samuel's method. But also needed is a way of tying the value function to the true value of the states. We have enforced this via rewards and by discounting or giving a fixed value to the terminal state. But Samuel's method included no rewards and no special treatment of the terminal positions of games. As Samuel himself pointed out, his value function could have become consistent merely by giving a constant value to all positions. He hoped to discourage such solutions by giving his piece-advantage term a large, nonmodifiable weight. But although this may decrease the likelihood of finding useless evaluation functions, it does not prohibit them. For example, a constant function could still be attained by setting the modifiable weights so as to cancel the effect of the nonmodifiable one.

Because Samuel's learning procedure was not constrained to find useful evaluation functions, it should have been possible for it to become worse with experience. In fact, Samuel reported observing this during extensive self-play training sessions. To get the program improving again, Samuel had to intervene and set the weight with the largest absolute value back to zero. His interpretation was that this drastic intervention jarred the program out of local optima, but another possibility is that it jarred the program out of evaluation functions that were consistent but had little to do with winning or losing the game.

Despite these potential problems, Samuel's checkers player using the generalization learning method approached "better-than-average" play. Fairly good amateur opponents characterized it as "tricky but beatable" (Samuel, 1959). In contrast to the rote-learning version, this version was able to develop a good middle game but remained weak in opening and endgame play. This program also included an ability to search through sets of features to find those that were most useful in forming the value function. A later version (Samuel, 1967) included refinements in its search procedure, such as alpha-beta pruning,

extensive use of a supervised learning mode called "book learning," and hierarchical lookup tables called signature tables (Griffith, 1966) to represent the value function instead of linear function approximation. This version learned to play much better than the 1959 program, though still not at a master level. Samuel's checkers-playing program was widely recognized as a significant achievement in artificial intelligence and machine learning.

16.3 Watson's Daily-Double Wagering

IBM WATSON[1] is the system developed by a team of IBM researchers to play the popular TV quiz show *Jeopardy!*.[2] It gained fame in 2011 by winning first prize in an exhibition match against human champions. Although the main technical achievement demonstrated by WATSON was its ability to quickly and accurately answer natural language questions over broad areas of general knowledge, its winning *Jeopardy!* performance also relied on sophisticated decision-making strategies for critical parts of the game. Tesauro, Gondek, Lechner, Fan, and Prager (2012, 2013) adapted Tesauro's TD-Gammon system described above to create the strategy used by WATSON in "Daily-Double" (DD) wagering in its celebrated winning performance against human champions. These authors report that the effectiveness of this wagering strategy went well beyond what human players are able to do in live game play, and that it, along with other advanced strategies, was an important contributor to WATSON's impressive winning performance. Here we focus only on DD wagering because it is the component of WATSON that owes the most to reinforcement learning.

Jeopardy! is played by three contestants who face a board showing 30 squares, each of which hides a clue and has a dollar value. The squares are arranged in six columns, each corresponding to a different category. A contestant selects a square, the host reads the square's clue, and each contestant may choose to respond to the clue by sounding a buzzer ("buzzing in"). The first contestant to buzz in gets to try responding to the clue. If this contestant's response is correct, their score increases by the dollar value of the square; if their response is not correct, or if they do not respond within five seconds, their score decreases by that amount, and the other contestants get a chance to buzz in to respond to the same clue. One or two squares (depending on the game's current round) are special DD squares. A contestant who selects one of these gets an exclusive opportunity to respond to the square's clue and has to decide—before the clue is revealed—on how much to wager, or bet. The bet has to be greater than five dollars but not greater than the contestant's current score. If the contestant responds correctly to the DD clue, their score increases by the bet amount; otherwise it decreases by the bet amount. At the end of each game is a "Final Jeopardy" (FJ) round in which each contestant writes down a sealed bet and then writes an answer after the clue is read. The contestant with the highest score after three rounds of play (where a round consists of revealing all 30 clues) is the winner. The game has many other details, but these are enough to appreciate

[1] Registered trademark of IBM Corp.

[2] Registered trademark of Jeopardy Productions Inc.

the importance of DD wagering. Winning or losing often depends on a contestant's DD wagering strategy.

Whenever WATSON selected a DD square, it chose its bet by comparing action values, $\hat{q}(s, bet)$, that estimated the probability of a win from the current game state, s, for each round-dollar legal bet. Except for some risk-abatement measures described below, WATSON selected the bet with the maximum action value. Action values were computed whenever a betting decision was needed by using two types of estimates that were learned before any live game play took place. The first were estimated values of the afterstates (Section 6.8) that would result from selecting each legal bet. These estimates were obtained from a state-value function, $\hat{v}(\cdot,\mathbf{w})$, defined by parameters $\mathbf{w}$, that gave estimates of the probability of a win for WATSON from any game state. The second estimates used to compute action values gave the "in-category DD confidence," p_{DD}, which estimated the likelihood that WATSON would respond correctly to the as-yet unrevealed DD clue.

Tesauro et al. used the reinforcement learning approach of TD-Gammon described above to learn $\hat{v}(\cdot,\mathbf{w})$: a straightforward combination of nonlinear TD(λ) using a multilayer ANN with weights $\mathbf{w}$ trained by backpropagating TD errors during many simulated games. States were represented to the network by feature vectors specifically designed for *Jeopardy!*. Features included the current scores of the three players, how many DDs remained, the total dollar value of the remaining clues, and other information related to the amount of play left in the game. Unlike TD-Gammon, which learned by self-play, WATSON's $\hat{v}$ was learned over millions of simulated games against carefully-crafted models of human players. In-category confidence estimates were conditioned on the number of right responses r and wrong responses w that WATSON gave in previously-played clues in the current category. The dependencies on (r, w) were estimated from WATSON's actual accuracies over many thousands of historical categories.

With the previously learned value function $\hat{v}$ and in-category DD confidence p_{DD}, WATSON computed $\hat{q}(s, bet)$ for each legal round-dollar bet as follows:

$$\hat{q}(s, bet) = p_{DD} \times \hat{v}(S_W + bet,\ldots) + (1 - p_{DD}) \times \hat{v}(S_W - bet,\ldots), \tag{16.2}$$

where S_W is WATSON's current score, and $\hat{v}$ gives the estimated value for the game state after WATSON's response to the DD clue, which is either correct or incorrect. Computing an action value this way corresponds to the insight from Exercise 3.19 that an action value is the expected next state value given the action (except that here it is the expected next *afterstate* value because the full next state of the entire game depends on the next square selection).

Tesauro et al. found that selecting bets by maximizing action values incurred "a frightening amount of risk," meaning that if WATSON's response to the clue happened to be wrong, the loss could be disastrous for its chances of winning. To decrease the downside risk of a wrong answer, Tesauro et al. adjusted (16.2) by subtracting a small fraction of the standard deviation over WATSON's correct/incorrect afterstate evaluations. They further reduced risk by prohibiting bets that would cause the wrong-answer afterstate value to decrease below a certain limit. These measures slightly reduced WATSON's expectation of winning, but they significantly reduced downside risk, not only in terms of average risk per DD bet, but even more so in extreme-risk scenarios where a risk-neutral WATSON would bet most or all of its bankroll.

Why was the TD-Gammon method of self-play not used to learn the critical value function $\hat{v}$? Learning from self-play in *Jeopardy!* would not have worked very well because WATSON was so different from any human contestant. Self-play would have led to exploration of state space regions that are not typical for play against human opponents, particularly human champions. In addition, unlike backgammon, *Jeopardy!* is a game of imperfect information because contestants do not have access to all the information influencing their opponents' play. In particular, *Jeopardy!* contestants do not know how much confidence their opponents have for responding to clues in the various categories. Self-play would have been something like playing poker with someone who is holding the same cards that you hold.

As a result of these complications, much of the effort in developing WATSON's DD-wagering strategy was devoted to creating good models of human opponents. The models did not address the natural language aspect of the game, but were instead stochastic process models of events that can occur during play. Statistics were extracted from an extensive fan-created archive of game information from the beginning of the show to the present day. The archive includes information such as the ordering of the clues, right and wrong contestant answers, DD locations, and DD and FJ bets for nearly 300,000 clues. Three models were constructed: an Average Contestant model (based on all the data), a Champion model (based on statistics from games with the 100 best players), and a Grand Champion model (based on statistics from games with the 10 best players). In addition to serving as opponents during learning, the models were used to assess the benefits produced by the learned DD-wagering strategy. WATSON's win rate in simulation when it used a baseline heuristic DD-wagering strategy was 61%; when it used the learned values and a default confidence value, its win rate increased to 64%; and with live in-category confidence, it was 67%. Tesauro et al. regarded this as a significant improvement, given that the DD wagering was needed only about 1.5 to 2 times in each game.

Because WATSON had only a few seconds to bet, as well as to select squares and decide whether or not to buzz in, the computation time needed to make these decisions was a critical factor. The ANN implementation of $\hat{v}$ allowed DD bets to be made quickly enough to meet the time constraints of live play. However, once games could be simulated fast enough through improvements in the simulation software, near the end of a game it was feasible to estimate the value of bets by averaging over many Monte-Carlo trials in which the consequence of each bet was determined by simulating play to the game's end. Selecting endgame DD bets in live play based on Monte-Carlo trials instead of the ANN significantly improved WATSON's performance because errors in value estimates in endgames could seriously affect its chances of winning. Making all the decisions via Monte-Carlo trials might have led to better wagering decisions, but this was simply impossible given the complexity of the game and the time constraints of live play.

Although its ability to quickly and accurately answer natural language questions stands out as WATSON's major achievement, all of its sophisticated decision strategies contributed to its impressive defeat of human champions. According to Tesauro et al. (2012):

> ... it is plainly evident that our strategy algorithms achieve a level of quantitative precision and real-time performance that exceeds human capabilities.

This is particularly true in the cases of DD wagering and endgame buzzing, where humans simply cannot come close to matching the precise equity and confidence estimates and complex decision calculations performed by Watson.

16.4 Optimizing Memory Control

Most computers use dynamic random access memory (DRAM) as their main memory because of its low cost and high capacity. The job of a DRAM memory controller is to efficiently use the interface between the processor chip and an off-chip DRAM system to provide the high-bandwidth and low-latency data transfer necessary for high-speed program execution. A memory controller needs to deal with dynamically changing patterns of read/write requests while adhering to a large number of timing and resource constraints required by the hardware. This is a formidable scheduling problem, especially with modern processors with multiple cores sharing the same DRAM.

İpek, Mutlu, Martínez, and Caruana (2008) (also Martínez and İpek, 2009) designed a reinforcement learning memory controller and demonstrated that it can significantly improve the speed of program execution over what was possible with conventional controllers at the time of their research. They were motivated by limitations of existing state-of-the-art controllers that used policies that did not take advantage of past scheduling experience and did not account for long-term consequences of scheduling decisions. İpek et al.'s project was carried out by means of simulation, but they designed the controller at the detailed level of the hardware needed to implement it—including the learning algorithm—directly on a processor chip.

Accessing DRAM involves a number of steps that have to be done according to strict time constraints. DRAM systems consist of multiple DRAM chips, each containing multiple rectangular arrays of storage cells arranged in rows and columns. Each cell stores a bit as the charge on a capacitor. Because the charge decreases over time, each DRAM cell needs to be recharged—refreshed—every few milliseconds to prevent memory content from being lost. This need to refresh the cells is why DRAM is called "dynamic."

Each cell array has a row buffer that holds a row of bits that can be transferred into or out of one of the array's rows. An *activate* command "opens a row," which means moving the contents of the row whose address is indicated by the command into the row buffer. With a row open, the controller can issue *read* and *write* commands to the cell array. Each read command transfers a word (a short sequence of consecutive bits) in the row buffer to the external data bus, and each write command transfers a word in the external data bus to the row buffer. Before a different row can be opened, a *precharge* command must be issued which transfers the (possibly updated) data in the row buffer back into the addressed row of the cell array. After this, another activate command can open a new row to be accessed. Read and write commands are *column commands* because they sequentially transfer bits into or out of columns of the row buffer; multiple bits can be transferred without re-opening the row. Read and write commands to the currently-open row can be carried out more quickly than accessing a different row, which would involve additional *row commands*: precharge and activate; this is sometimes

referred to as "row locality." A memory controller maintains a *memory transaction queue* that stores memory-access requests from the processors sharing the memory system. The controller has to process requests by issuing commands to the memory system while adhering to a large number of timing constraints.

A controller's policy for scheduling access requests can have a large effect on the performance of the memory system, such as the average latency with which requests can be satisfied and the throughput the system is capable of achieving. The simplest scheduling strategy handles access requests in the order in which they arrive by issuing all the commands required by the request before beginning to service the next one. But if the system is not ready for one of these commands, or executing a command would result in resources being underutilized (e.g., due to timing constraints arising from servicing that one command), it makes sense to begin servicing a newer request before finishing the older one. Policies can gain efficiency by reordering requests, for example, by giving priority to read requests over write requests, or by giving priority to read/write commands to already open rows. The policy called First-Ready, First-Come-First-Serve (FR-FCFS), gives priority to column commands (read and write) over row commands (activate and precharge), and in case of a tie gives priority to the oldest command. FR-FCFS was shown to outperform other scheduling policies in terms of average memory-access latency under conditions commonly encountered (Rixner, 2004).

Figure 16.3 is a high-level view of İpek et al.'s reinforcement learning memory controller. They modeled the DRAM access process as an MDP whose states are the contents of the transaction queue and whose actions are commands to the DRAM system: *precharge*, *activate*, *read*, *write*, and *NoOp*. The reward signal is 1 whenever the action is *read* or *write*, and otherwise it is 0. State transitions were considered to be stochastic because the next state of the system not only depends on the scheduler's command, but also on aspects of the system's behavior that the scheduler cannot control, such as the workloads of the processor cores accessing the DRAM system.

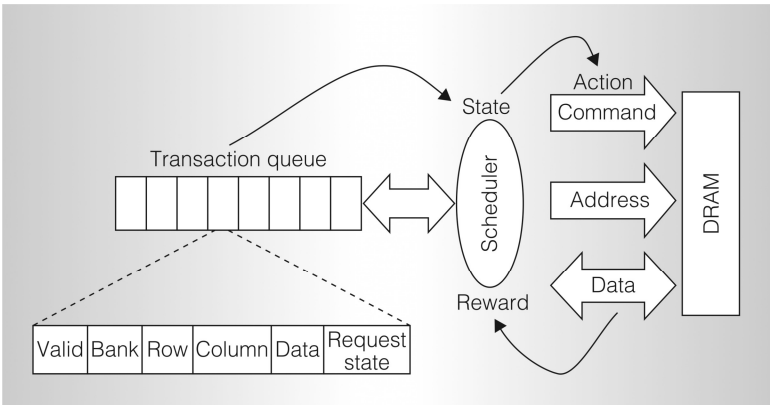


Figure 16.3: High-level view of the reinforcement learning DRAM controller. The scheduler is the reinforcement learning agent. Its environment is represented by features of the transaction queue, and its actions are commands to the DRAM system. ©2009 IEEE. Reprinted, with permission, from J. F. Martínez and E. İpek, Dynamic multicore resource management: A machine learning approach, *Micro, IEEE, 29*(5), p. 12.

Critical to this MDP are constraints on the actions available in each state. Recall from Chapter 3 that the set of available actions can depend on the state: $A_t \in \mathcal{A}(S_t)$, where A_t is the action at time step t and $\mathcal{A}(S_t)$ is the set of actions available in state S_t. In this application, the integrity of the DRAM system was assured by not allowing actions that would violate timing or resource constraints. Although İpek et al. did not make it explicit, they effectively accomplished this by pre-defining the sets $\mathcal{A}(S_t)$ for all possible states S_t.

These constraints explain why the MDP has a *NoOp* action and why the reward signal is 0 except when a *read* or *write* command is issued. *NoOp* is issued when it is the sole legal action in a state. To maximize utilization of the memory system, the controller's task is to drive the system to states in which either a *read* or a *write* action can be selected: only these actions result in sending data over the external data bus, so it is only these that contribute to the throughput of the system. Although *precharge* and *activate* produce no immediate reward, the agent needs to select these actions to make it possible to later select the rewarded *read* and *write* actions.

The scheduling agent used Sarsa (Section 6.4) to learn an action-value function. States were represented by six integer-valued features. To approximate the action-value function, the algorithm used linear function approximation implemented by tile coding with hashing (Section 9.5.4). The tile coding had 32 tilings, each storing 256 action values as 16-bit fixed point numbers. Exploration was ε-greedy with $\varepsilon = 0.05$.

State features included the number of read requests in the transaction queue, the number of write requests in the transaction queue, the number of write requests in the transaction queue waiting for their row to be opened, and the number of read requests in the transaction queue waiting for their row to be opened that are the oldest issued by their requesting processors. (The other features depended on how the DRAM interacts with cache memory, details we omit here.) The selection of the state features was based on İpek et al.'s understanding of factors that impact DRAM performance. For example, balancing the rate of servicing reads and writes based on how many of each are in the transaction queue can help avoid stalling the DRAM system's interaction with cache memory. The authors in fact generated a relatively long list of potential features, and then pared them down to a handful using simulations guided by stepwise feature selection.

An interesting aspect of this formulation of the scheduling problem as an MDP is that the features input to the tile coding for defining the action-value function were different from the features used to specify the action-constraint sets $\mathcal{A}(S_t)$. Whereas the tile coding input was derived from the contents of the transaction queue, the constraint sets depended on a host of other features related to timing and resource constraints that had to be satisfied by the hardware implementation of the entire system. In this way, the action constraints ensured that the learning algorithm's exploration could not endanger the integrity of the physical system, while learning was effectively limited to a "safe" region of the much larger state space of the hardware implementation.

Because an objective of this work was that the learning controller could be implemented on a chip so that learning could occur online while a computer is running, hardware implementation details were important considerations. The design included two five-stage pipelines to calculate and compare two action values at every processor clock cycle, and

to update the appropriate action value. This included accessing the tile coding which was stored on-chip in static RAM. For the configuration İpek et al. simulated, which was a 4GHz 4-core chip typical of high-end workstations at the time of their research, there were 10 processor cycles for every DRAM cycle. Considering the cycles needed to fill the pipes, up to 12 actions could be evaluated in each DRAM cycle. İpek et al. found that the number of legal commands for any state was rarely greater than this, and that performance loss was negligible if enough time was not always available to consider all legal commands. These and other clever design details made it feasible to implement the complete controller and learning algorithm on a multi-processor chip.

İpek et al. evaluated their learning controller in simulation by comparing it with three other controllers: (1) the FR-FCFS controller mentioned above that produces the best on-average performance, (2) a conventional controller that processes each request in order, and (3) an unrealizable ideal controller, called the Optimistic controller, able to sustain 100% DRAM throughput if given enough demand by ignoring all timing and resource constraints, but otherwise modeling DRAM latency (as row buffer hits) and bandwidth. They simulated nine memory-intensive parallel workloads consisting of scientific and data-mining applications. Figure 16.4 shows the performance (the inverse of execution time normalized to the performance of FR-FCFS) of each controller for the nine applications, together with the geometric mean of their performances over the applications. The learning controller, labeled RL in the figure, improved over that of FR-FCFS by from 7% to 33% over the nine applications, with an average improvement of 19%. Of course, no realizable controller can match the performance of Optimistic, which ignores all timing and resource constraints, but the learning controller's performance closed the gap with Optimistic's upper bound by an impressive 27%.

Because the rationale for on-chip implementation of the learning algorithm was to allow the scheduling policy to adapt online to changing workloads, İpek et al. analyzed the impact of online learning compared to a previously-learned fixed policy. They trained

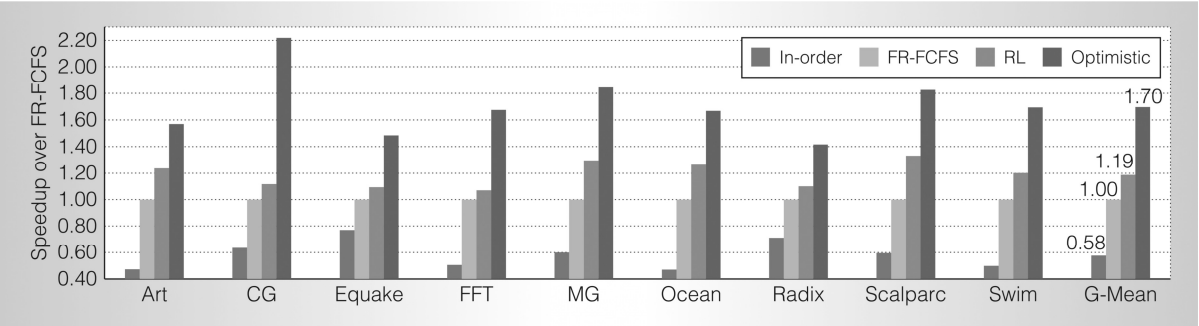


Figure 16.4: Performances of four controllers over a suite of 9 simulated benchmark applications. The controllers are: the simplest 'in-order' controller, FR-FCFS, the learning controller RL, and the unrealizable Optimistic controller which ignores all timing and resource constraints to provide a performance upper bound. Performance, normalized to that of FR-FCFS, is the inverse of execution time. At far right is the geometric mean of performances over the 9 benchmark applications for each controller. Controller RL comes closest to the ideal performance. ©2009 IEEE. Reprinted, with permission, from J. F. Martínez and E. İpek, Dynamic multicore resource management: A machine learning approach, *Micro, IEEE, 29*(5), p. 13.

their controller with data from all nine benchmark applications and then held the resulting action values fixed throughout the simulated execution of the applications. They found that the average performance of the controller that learned online was 8% better than that of the controller using the fixed policy, leading them to conclude that online learning is an important feature of their approach.

This learning memory controller was never committed to physical hardware because of the large cost of fabrication. Nevertheless, İpek et al. could convincingly argue on the basis of their simulation results that a memory controller that learns online via reinforcement learning has the potential to improve performance to levels that would otherwise require more complex and more expensive memory systems, while removing from human designers some of the burden required to manually design efficient scheduling policies. Mukundan and Martínez (2012) took this project forward by investigating learning controllers with additional actions, other performance criteria, and more complex reward functions derived using genetic algorithms. They considered additional performance criteria related to energy efficiency. The results of these studies surpassed the earlier results described above and significantly surpassed the 2012 state-of-the-art for all of the performance criteria they considered. The approach is especially promising for developing sophisticated power-aware DRAM interfaces.

16.5 Human-level Video Game Play

One of the greatest challenges in applying reinforcement learning to real-world problems is deciding how to represent and store value functions and/or policies. Unless the state set is finite and small enough to allow exhaustive representation by a lookup table—as in many of our illustrative examples—one must use a parameterized function approximation scheme. Whether linear or nonlinear, function approximation relies on features that have to be readily accessible to the learning system and able to convey the information necessary for skilled performance. Most successful applications of reinforcement learning owe much to sets of features carefully handcrafted based on human knowledge and intuition about the specific problem to be tackled.

A team of researchers at Google DeepMind developed an impressive demonstration that a deep multi-layer ANN can automate the feature design process (Mnih et al., 2013, 2015). Multi-layer ANNs have been used for function approximation in reinforcement learning ever since the 1986 popularization of the backpropagation algorithm as a method for learning internal representations (Rumelhart, Hinton, and Williams, 1986; see Section 9.7). Striking results have been obtained by coupling reinforcement learning with backpropagation. The results obtained by Tesauro and colleagues with TD-Gammon and Watson discussed above are notable examples. These and other applications benefited from the ability of multi-layer ANNs to learn task-relevant features. However, in all the examples of which we are aware, the most impressive demonstrations required the network's input to be represented in terms of specialized features handcrafted for the given problem. This is vividly apparent in the TD-Gammon results. TD-Gammon 0.0, whose network input was essentially a "raw" representation of the backgammon board, meaning that it involved very little knowledge of backgammon, learned to play approximately as well as

the best previous backgammon computer programs. Adding specialized backgammon features produced TD-Gammon 1.0 which was substantially better than all previous backgammon programs and competed well against human experts.

Mnih et al. developed a reinforcement learning agent called *deep Q-network* (DQN) that combined Q-learning with a *deep convolutional* ANN, a many-layered, or deep, ANN specialized for processing spatial arrays of data such as images. We describe deep convolutional ANNs in Section 9.7. By the time of Mnih et al.'s work with DQN, deep ANNs, including deep convolutional ANNs, had produced impressive results in many applications, but they had not been widely used in reinforcement learning.

Mnih et al. used DQN to show how a reinforcement learning agent can achieve a high level of performance on any of a collection of different problems without having to use different problem-specific feature sets. To demonstrate this, they let DQN learn to play 49 different Atari 2600 video games by interacting with a game emulator. DQN learned a different policy for each of the 49 games (because the weights of its ANN were reset to random values before learning on each game), but it used the same raw input, network architecture, and parameter values (e.g., step size, discount rate, exploration parameters, and many more specific to the implementation) for all the games. DQN achieved levels of play at or beyond human level on a large fraction of these games. Although the games were alike in being played by watching streams of video images, they varied widely in other respects. Their actions had different effects, they had different state-transition dynamics, and they needed different policies for learning high scores. The deep convolutional ANN learned to transform the raw input common to all the games into features specialized for representing the action values required for playing at the high level DQN achieved for most of the games.

The Atari 2600 is a home video game console that was sold in various versions by Atari Inc. from 1977 to 1992. It introduced or popularized many arcade video games that are now considered classics, such as Pong, Breakout, Space Invaders, and Asteroids. Although much simpler than modern video games, Atari 2600 games are still entertaining and challenging for human players, and they have been attractive as testbeds for developing and evaluating reinforcement learning methods (Diuk, Cohen, Littman, 2008; Naddaf, 2010; Cobo, Zang, Isbell, and Thomaz, 2011; Bellemare, Veness, and Bowling, 2013). Bellemare, Naddaf, Veness, and Bowling (2012) developed the publicly available Arcade Learning Environment (ALE) to encourage and simplify using Atari 2600 games to study learning and planning algorithms.

These previous studies and the availability of ALE made the Atari 2600 game collection a good choice for Mnih et al.'s demonstration, which was also influenced by the impressive human-level performance that TD-Gammon was able to achieve in backgammon. DQN is similar to TD-Gammon in using a multi-layer ANN as the function approximation method for a semi-gradient form of a TD algorithm, with the gradients computed by the backpropagation algorithm. However, instead of using TD(λ) as TD-Gammon did, DQN used the semi-gradient form of Q-learning. TD-Gammon estimated the values of afterstates, which were easily obtained from the rules for making backgammon moves. To use the same algorithm for the Atari games would have required generating the next states for each possible action (which would not have been afterstates in that case). This could have been done by using the game emulator to run single-step simulations

for all the possible actions (which ALE makes possible). Or a model of each game's state-transition function could have been learned and used to predict next states (Oh, Guo, Lee, Lewis, and Singh, 2015). While these methods might have produced results comparable to DQN's, they would have been more complicated to implement and would have significantly increased the time needed for learning. Another motivation for using Q-learning was that DQN used the *experience replay* method, described below, which requires an off-policy algorithm. Being model-free and off-policy made Q-learning a natural choice.

Before describing the details of DQN and how the experiments were conducted, we look at the skill levels DQN was able to achieve. Mnih et al. compared the scores of DQN with the scores of the best performing learning system in the literature at the time, the scores of a professional human games tester, and the scores of an agent that selected actions at random. The best system from the literature used linear function approximation with features designed using some knowledge about Atari 2600 games (Bellemare, Naddaf, Veness, and Bowling, 2013). DQN learned on each game by interacting with the game emulator for 50 million frames, which corresponds to about 38 days of experience with the game. At the start of learning on each game, the weights of DQN's network were reset to random values. To evaluate DQN's skill level after learning, its score was averaged over 30 sessions on each game, each lasting up to 5 minutes and beginning with a random initial game state. The professional human tester played using the same emulator (with the sound turned off to remove any possible advantage over DQN which did not process audio). After 2 hours of practice, the human played about 20 episodes of each game for up to 5 minutes each and was not allowed to take any break during this time. DQN learned to play better than the best previous reinforcement learning systems on all but 6 of the games, and played better than the human player on 22 of the games. By considering any performance that scored at or above 75% of the human score to be comparable to, or better than, human-level play, Mnih et al. concluded that the levels of play DQN learned reached or exceeded human level on 29 of the 46 games. See Mnih et al. (2015) for a more detailed account of these results.

For an artificial learning system to achieve these levels of play would be impressive enough, but what makes these results remarkable—and what many at the time considered to be breakthrough results for artificial intelligence—is that the very same learning system achieved these levels of play on widely varying games without relying on any game-specific modifications.

A human playing any of these 49 Atari games sees 210×160 pixel image frames with 128 colors at 60Hz. In principle, exactly these images could have formed the raw input to DQN, but to reduce memory and processing requirements, Mnih et al. preprocessed each frame to produce an 84×84 array of luminance values. Because the full states of many of the Atari games are not completely observable from the image frames, Mnih et al. "stacked" the four most recent frames so that the inputs to the network had dimension $84 \times 84 \times 4$. This did not eliminate partial observability for all of the games, but it was helpful in making many of them more Markovian.

An essential point here is that these preprocessing steps were exactly the same for all 46 games. No game-specific prior knowledge was involved beyond the general understanding that it should still be possible to learn good policies with this reduced dimension and

that stacking adjacent frames should help with the partial observability of some of the games. Because no game-specific prior knowledge beyond this minimal amount was used in preprocessing the image frames, we can think of the $84 \times 84 \times 4$ input vectors as being "raw" input to DQN.

The basic architecture of DQN is similar to the deep convolutional ANN illustrated in Figure 9.15 (though unlike that network, subsampling in DQN is treated as part of each convolutional layer, with feature maps consisting of units having only a selection of the possible receptive fields). DQN has three hidden convolutional layers, followed by one fully connected hidden layer, followed by the output layer. The three successive hidden convolutional layers of DQN produce 32 20×20 feature maps, 64 9×9 feature maps, and 64 7×7 feature maps. The activation function of the units of each feature map is a rectifier nonlinearity ($\max(0, x)$). The 3,136 ($64 \times 7 \times 7$) units in this third convolutional layer all connect to each of 512 units in the fully connected hidden layer, which then each connect to all 18 units in the output layer, one for each possible action in an Atari game.

The activation levels of DQN's output units were the estimated optimal action values of the corresponding state–action pairs, for the state represented by the network's input. The assignment of output units to a game's actions varied from game to game, and because the number of valid actions varied between 4 and 18 for the games, not all output units had functional roles in all of the games. It helps to think of the network as if it were 18 separate networks, one for estimating the optimal action value of each possible action. In reality, these networks shared their initial layers, but the output units learned to use the features extracted by these layers in different ways.

DQN's reward signal indicated how a games's score changed from one time step to the next: $+1$ whenever it increased, -1 whenever it decreased, and 0 otherwise. This standardized the reward signal across the games and made a single step-size parameter work well for all the games despite their varying ranges of scores. DQN used an ε-greedy policy, with ε decreasing linearly over the first million frames and remaining at a low value for the rest of the learning session. The values of the various other parameters, such as the learning step size, discount rate, and others specific to the implementation, were selected by performing informal searches to see which values worked best for a small selection of the games. These values were then held fixed for all of the games.

After DQN selected an action, the action was executed by the game emulator, which returned a reward and the next video frame. The frame was preprocessed and added to the four-frame stack that became the next input to the network. Skipping for the moment the changes to the basic Q-learning procedure made by Mnih et al., DQN used the following semi-gradient form of Q-learning to update the network's weights:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \Big[R_{t+1} + \gamma \max_a \hat{q}(S_{t+1}, a, \mathbf{w}_t) - \hat{q}(S_t, A_t, \mathbf{w}_t) \Big] \nabla \hat{q}(S_t, A_t, \mathbf{w}_t), \quad (16.3)$$

where $\mathbf{w}_t$ is the vector of the network's weights, A_t is the action selected at time step t, and S_t and S_{t+1} are respectively the preprocessed image stacks input to the network at time steps t and $t+1$.

The gradient in (16.3) was computed by backpropagation. Imagining again that there was a separate network for each action, for the update at time step t, backpropagation was applied only to the network corresponding to A_t. Mnih et al. took advantage of techniques shown to improve the basic backpropagation algorithm when applied to large

networks. They used a *mini-batch method* that updated weights only after accumulating gradient information over a small batch of images (here after 32 images). This yielded smoother sample gradients compared to the usual procedure that updates weights after each action. They also used a gradient-ascent algorithm called RMSProp (Tieleman and Hinton, 2012) that accelerates learning by adjusting the step-size parameter for each weight based on a running average of the magnitudes of recent gradients for that weight.

Mnih et al. modified the basic Q-learning procedure in three ways. First, they used a method called *experience replay* first studied by Lin (1992). This method stores the agent's experience at each time step in a replay memory that is accessed to perform the weight updates. It worked like this in DQN. After the game emulator executed action A_t in a state represented by the image stack S_t, and returned reward R_{t+1} and image stack S_{t+1}, it added the tuple $(S_t, A_t, R_{t+1}, S_{t+1})$ to the replay memory. This memory accumulated experiences over many plays of the same game. At each time step multiple Q-learning updates—a mini-batch—were performed based on experiences sampled uniformly at random from the replay memory. Instead of S_{t+1} becoming the new S_t for the next update as it would in the usual form of Q-learning, a new unconnected experience was drawn from the replay memory to supply data for the next update. Because Q-learning is an off-policy algorithm, it does not need to be applied along connected trajectories.

Q-learning with experience replay provided several advantages over the usual form of Q-learning. The ability to use each stored experience for many updates allowed DQN to learn more efficiently from its experiences. Experience replay reduced the variance of the updates because successive updates were not correlated with one another as they would be with standard Q-learning. And by removing the dependence of successive experiences on the current weights, experience replay eliminated one source of instability.

Mnih et al. modified standard Q-learning in a second way to improve its stability. As in other methods that bootstrap, the target for a Q-learning update depends on the current action-value function estimate. When a parameterized function approximation method is used to represent action values, the target is a function of the same parameters that are being updated. For example, the target in the update given by (16.3) is $\gamma \max_a \hat{q}(S_{t+1}, a, \mathbf{w}_t)$. Its dependence on $\mathbf{w}_t$ complicates the process compared to the simpler supervised-learning situation in which the targets do not depend on the parameters being updated. As discussed in Chapter 11 this can lead to oscillations and/or divergence.

To address this problem Mnih et al. used a technique that brought Q-learning closer to the simpler supervised-learning case while still allowing it to bootstrap. Whenever a certain number, C, of updates had been done to the weights $\mathbf{w}$ of the action-value network, they inserted the network's current weights into another network and held these duplicate weights fixed for the next C updates of $\mathbf{w}$. The outputs of this duplicate network over the next C updates of $\mathbf{w}$ were used as the Q-learning targets. Letting $\tilde{q}$ denote the output of this duplicate network, then instead of (16.3) the update rule was:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \Big[R_{t+1} + \gamma \max_a \tilde{q}(S_{t+1}, a, \mathbf{w}_t) - \hat{q}(S_t, A_t, \mathbf{w}_t) \Big] \nabla \hat{q}(S_t, A_t, \mathbf{w}_t).$$

A final modification of standard Q-learning was also found to improve stability. They clipped the error term $R_{t+1} + \gamma \max_a \tilde{q}(S_{t+1}, a, \mathbf{w}_t) - \hat{q}(S_t, A_t, \mathbf{w}_t)$ so that it remained in the interval $[-1, 1]$.

Mnih et al. conducted a large number of learning runs on 5 of the games to gain insight into the effect that various of DQN's design features had on its performance. They ran DQN with the four combinations of experience replay and the duplicate target network being included or not included. Although the results varied from game to game, each of these features alone significantly improved performance, and very dramatically improved performance when used together. Mnih et al. also studied the role played by the deep convolutional ANN in DQN's learning ability by comparing the deep convolutional version of DQN with a version having a network of just one linear layer, both receiving the same stacked preprocessed video frames. Here, the improvement of the deep convolutional version over the linear version was particularly striking across all 5 of the test games.

Creating artificial agents that excel over a diverse collection of challenging tasks has been an enduring goal of artificial intelligence. The promise of machine learning as a means for achieving this has been frustrated by the need to craft problem-specific representations. DeepMind's DQN stands as a major step forward by demonstrating that a single agent can learn problem-specific features enabling it to acquire human-competitive skills over a range of tasks. This demonstration did not produce one agent that simultaneously excelled at all the tasks (because learning occurred separately for each task), but it showed that deep learning can reduce, and possibly eliminate, the need for problem-specific design and tuning. As Mnih et al. point out, however, DQN is not a complete solution to the problem of task-independent learning. Although the skills needed to excel on the Atari games were markedly diverse, all the games were played by observing video images, which made a deep convolutional ANN a natural choice for this collection of tasks. In addition, DQN's performance on some of the Atari 2600 games fell considerably short of human skill levels on these games. The games most difficult for DQN—especially Montezuma's Revenge on which DQN learned to perform about as well as the random player—require deep planning beyond what DQN was designed to do. Further, learning control skills through extensive practice, like DQN learned how to play the Atari games, is just one of the types of learning humans routinely accomplish. Despite these limitations, DQN advanced the state-of-the-art in machine learning by impressively demonstrating the promise of combining reinforcement learning with modern methods of deep learning.

16.6 Mastering the Game of Go

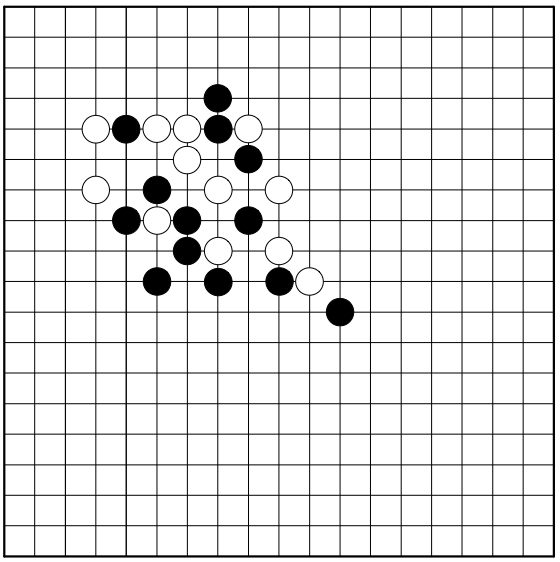
The ancient Chinese game of Go has challenged artificial intelligence researchers for many decades. Methods that achieve human-level skill, or even superhuman-level skill, in other games have not been successful in producing strong Go programs. Thanks to a very active community of Go programmers and international competitions, the level of Go program play has improved significantly over the years. Until recently, however, no Go program had been able to play anywhere near the level of a human Go master.

A team at DeepMind (Silver et al., 2016) developed the program *AlphaGo* that broke this barrier by combining deep ANNs (Section 9.7), supervised learning, Monte Carlo

tree search (MCTS, Section 8.11), and reinforcement learning. By the time of Silver et al.'s 2016 publication, *AlphaGo* had been shown to be decisively stronger than other Go programs, and it had defeated the European Go champion Fan Hui 5 games to 0. These were the first victories of a Go program over a professional human Go player without handicap in full Go games. Shortly thereafter, a similar version of *AlphaGo* won stunning victories over the 18-time world champion Lee Sedol, winning 4 out of a 5 games in a challenge match, making worldwide headline news. Artificial intelligence researchers thought that it would be many more years, perhaps decades, before a program reached this level of play.

Here we describe *AlphaGo* and a successor program called *AlphaGo Zero* (Silver et al. 2017a). Where in addition to reinforcement learning, *AlphaGo* relied on supervised learning from a large database of expert human moves, *AlphaGo Zero* used only reinforcement learning and no human data or guidance beyond the basic rules of the game (hence the *Zero* in its name). We first describe *AlphaGo* in some detail in order to highlight the relative simplicity of *AlphaGo Zero*, which is both higher-performing and more of a pure reinforcement learning program.

In many ways, both *AlphaGo* and *AlphaGo Zero* are descendants of Tesauro's TD-Gammon (Section 16.1), itself a descendant of Samuel's checkers player (Section 16.2). All these programs included reinforcement learning over simulated games of self-play. *AlphaGo* and *AlphaGo Zero* also built upon the progress made by DeepMind on playing Atari games with the program DQN (Section 16.5) that used deep convolutional ANNs to approximate optimal value functions.



A Go board configuration

Go is a game between two players who alternately place black and white 'stones' on unoccupied intersections, or 'points,' on a board with a grid of 19 horizontal and 19 vertical lines to produce positions like that shown to the right. The game's goal is to capture an area of the board larger than that captured by the opponent. Stones are captured according to simple rules. A player's stones are captured if they are completely surrounded by the other player's stones, meaning that there is no horizontally or vertically adjacent point that is unoccupied. For example, the left panel of Figure 16.5 (on the next page) shows three white stones with an unoccupied adjacent point (labeled X). If black were to place a stone on X, then the three white stones would be captured and taken off the board (middle panel). However, if white were to place a stone on point X first, then the possibility of this capture would be blocked (right panel). Other rules are needed to prevent infinite capturing/recapturing loops. The game ends when neither player wishes to place another stone. These rules are simple, but they produce a very complex game that has had wide appeal for thousands of years.

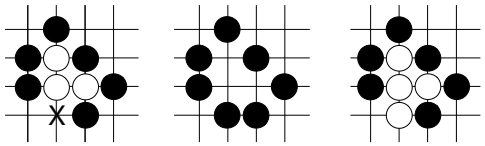


Figure 16.5: Go capturing rule. Left: the three white stones are not surrounded because point X is unoccupied. Middle: if black places a stone on X, the three white stones are captured and removed from the board. Right: if white places a stone on point X first, the capture is blocked.

Methods that produce strong play for other games, such as chess, have not worked as well for Go. The search space for Go is significantly larger than that of chess because Go has a larger number of legal moves per position than chess (≈ 250 versus ≈ 35) and Go games tend to involve more moves than chess games (≈ 150 versus ≈ 80). But the size of the search space is not the major factor that makes Go so difficult. Exhaustive search is infeasible for both chess and Go, and Go on smaller boards (e.g., 9×9) has proven to be exceedingly difficult as well. Experts agree that the major stumbling block to creating stronger-than-amateur Go programs is the difficulty of defining an adequate position evaluation function. A good evaluation function allows search to be truncated at a feasible depth by providing relatively easy-to-compute predictions of what deeper search would likely yield. According to Müller (2002): "No simple yet reasonable evaluation function will ever be found for Go." A major step forward was the introduction of MCTS to Go programs. The strongest programs at the time of *AlphaGo*'s development all included MCTS, but master-level skill remained elusive.

Recall from Section 8.11 that MCTS is a decision-time planning procedure that does not attempt to learn and store a global evaluation function. Like a rollout algorithm (Section 8.10), it runs many Monte Carlo simulations of entire episodes (here, entire Go games) to select each action (here, each Go move: where to place a stone or to resign). Unlike a simple rollout algorithm, however, MCTS is an iterative procedure that incrementally extends a search tree whose root node represents the current environment state. As illustrated in Figure 8.10, each iteration traverses the tree by simulating actions guided by statistics associated with the tree's edges. In its basic version, when a simulation reaches a leaf node of the search tree, MCTS expands the tree by adding some, or all, of the leaf node's children to the tree. From the leaf node, or one of its newly added child nodes, a rollout is executed: a simulation that typically proceeds all the way to a terminal state, with actions selected by a rollout policy. When the rollout completes, the statistics associated with the search tree's edges that were traversed in this iteration are updated by backing up the return produced by the rollout. MCTS continues this process, starting each time at the search tree's root at the current state, for as many iterations as possible given the time constraints. Then, finally, an action from the root node (which still represents the current environment state) is selected according to statistics accumulated in the root node's outgoing edges. This is the action the agent takes. After the environment transitions to its next state, MCTS is executed again with the root node set to represent the new current state. The search tree at the start of this next execution might be just this new root node, or it might include descendants of this node left over from MCTS's previous execution. The remainder of the tree is discarded.

16.6.1 AlphaGo

The main innovation that made *AlphaGo* such a strong player is that it selected moves by a novel version of MCTS that was guided by both a policy and a value function learned by reinforcement learning with function approximation provided by deep convolutional ANNs. Another key feature is that instead of reinforcement learning starting from random network weights, it started from weights that were the result of previous supervised learning from a large collection of human expert moves.

The DeepMind team called *AlphaGo*'s modification of basic MCTS "asynchronous policy and value MCTS," or APV-MCTS. It selected actions via basic MCTS as described above but with some twists in how it extended its search tree and how it evaluated action edges. In contrast to basic MCTS, which expands its current search tree by using stored action values to select an unexplored edge from a leaf node, APV-MCTS, as implemented in *AlphaGo*, expanded its tree by choosing an edge according to probabilities supplied by a 13-layer deep convolutional ANN, called the *SL-policy network*, trained previously by supervised learning to predict moves contained in a database of nearly 30 million human expert moves.

Then, also in contrast to basic MCTS, which evaluates the newly-added state node solely by the return of a rollout initiated from it, APV-MCTS evaluated the node in two ways: by this return of the rollout, but also by a value function, v_θ, learned previously by a reinforcement learning method. If s was the newly-added node, its value became

$$v(s) = (1 - \eta)v_\theta(s) + \eta G, \tag{16.4}$$

where G was the return of the rollout and η controlled the mixing of the values resulting from these two evaluation methods. In *AlphaGo*, these values were supplied by the *value network*, another 13-layer deep convolutional ANN that was trained as we describe below to output estimated values of board positions. APV-MCTS's rollouts in *AlphaGo* were simulated games with both players using a fast *rollout policy* provided by a simple linear network, also trained by supervised learning before play. Throughout its execution, APV-MCTS kept track of how many simulations passed through each edge of the search tree, and when its execution completed, the most-visited edge from the root node was selected as the action to take, here the move *AlphaGo* actually made in a game.

The value network had the same structure as the deep convolutional SL policy network except that it had a single output unit that gave estimated values of game positions instead of the SL policy network's probability distributions over legal actions. Ideally, the value network would output optimal state values, and it might have been possible to approximate the optimal value function along the lines of TD-Gammon described above: self-play with nonlinear TD(λ) coupled to a deep convolutional ANN. But the DeepMind team took a different approach that held more promise for a game as complex as Go. They divided the process of training the value network into two stages. In the first stage, they created the best policy they could by using reinforcement learning to train an *RL policy network*. This was a deep convolutional ANN with the same structure as the SL policy network. It was initialized with the final weights of the SL policy network that were learned via supervised learning, and then policy-gradient reinforcement learning was used to improve upon the SL policy. In the second stage of training the value network,

the team used Monte Carlo policy evaluation on data obtained from a large number of simulated self-play games with moves selected by the RL policy network.

Figure 16.6 illustrates the networks used by *AlphaGo* and the steps taken to train them in what the DeepMind team called the "*AlphaGo* pipeline." All these networks were trained before any live game play took place, and their weights remained fixed throughout live play.

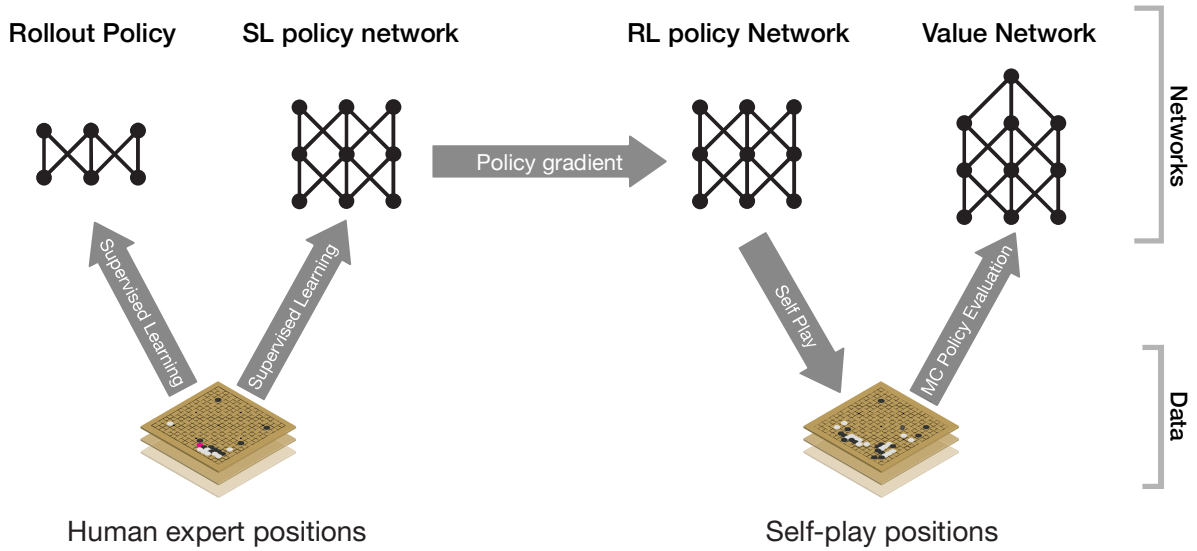


Figure 16.6: *AlphaGo* pipeline. Adapted with permission from Macmillan Publishers Ltd: *Nature*, vol. 529(7587), p. 485, ©2016.

Here is some more detail about *AlphaGo*'s ANNs and their training. The identically-structured SL and RL policy networks were similar to DQN's deep convolutional network described in Section 16.5 for playing Atari games, except that they had 13 convolutional layers with the final layer consisting of a soft-max unit for each point on the 19×19 Go board. The networks' input was a $19 \times 19 \times 48$ image stack in which each point on the Go board was represented by the values of 48 binary or integer-valued features. For example, for each point, one feature indicated if the point was occupied by one of *AlphaGo*'s stones, one of its opponent's stones, or was unoccupied, thus providing the "raw" representation of the board configuration. Other features were based on the rules of Go, such as the number of adjacent points that were empty, the number of opponent stones that would be captured by placing a stone there, the number of turns since a stone was placed there, and other features that the design team considered to be important.

Training the SL policy network took approximately 3 weeks using a distributed implementation of stochastic gradient ascent on 50 processors. The network achieved 57% accuracy, where the best accuracy achieved by other groups at the time of publication was 44.4%. Training the RL policy network was done by policy gradient reinforcement learning over simulated games between the RL policy network's current policy and opponents using policies randomly selected from policies produced by earlier iterations of the learning algorithm. Playing against a randomly selected collection of opponents

prevented overfitting to the current policy. The reward signal was +1 if the current policy won, −1 if it lost, and zero otherwise. These games directly pitted the two policies against one another without involving MCTS. By simulating many games in parallel on 50 processors, the DeepMind team trained the RL policy network on a million games in a single day. In testing the final RL policy, they found that it won more than 80% of games played against the SL policy, and it won 85% of games played against a Go program using MCTS that simulated 100,000 games per move.

The value network, whose structure was similar to that of the SL and RL policy networks except for its single output unit, received the same input as the SL and RL policy networks with the exception that there was an additional binary feature giving the current color to play. Monte Carlo policy evaluation was used to train the network from data obtained from a large number of self-play games played using the RL policy. To avoid overfitting and instability due to the strong correlations between positions encountered in self-play, the DeepMind team constructed a data set of 30 million positions each chosen randomly from a unique self-play game. Then training was done using 50 million mini-batches each of 32 positions drawn from this data set. Training took one week on 50 GPUs.

The rollout policy was learned prior to play by a simple linear network trained by supervised learning from a corpus of 8 million human moves. The rollout policy network had to output actions quickly while still being reasonably accurate. In principle, the SL or RL policy networks could have been used in the rollouts, but the forward propagation through these deep networks took too much time for either of them to be used in rollout simulations, a great many of which had to be carried out for each move decision during live play. For this reason, the rollout policy network was less complex than the other policy networks, and its input features could be computed more quickly than the features used for the policy networks. The rollout policy network allowed approximately 1,000 complete game simulations per second to be run on each of the processing threads that *AlphaGo* used.

One may wonder why the SL policy was used instead of the better RL policy to select actions in the expansion phase of APV-MCTS. These policies took the same amount of time to compute because they used the same network architecture. The team actually found that *AlphaGo* played better against human opponents when APV-MCTS used as the SL policy instead of the RL policy. They conjectured that the reason for this was that the latter was tuned to respond to optimal moves rather than to the broader set of moves characteristic of human play. Interestingly, the situation was reversed for the value function used by APV-MCTS. They found that when APV-MCTS used the value function derived from the RL policy, it performed better than if it used the value function derived from the SL policy.

Several methods worked together to produce *AlphaGo*'s impressive playing skill. The DeepMind team evaluated different versions of *AlphaGo* in order to assess the contributions made by these various components. The parameter η in (16.4) controlled the mixing of game state evaluations produced by the value network and by rollouts. With $\eta = 0$, *AlphaGo* used just the value network without rollouts, and with $\eta = 1$, evaluation relied just on rollouts. They found that *AlphaGo* using just the value network played

better than the rollout-only *AlphaGo*, and in fact played better than the strongest of all other Go programs existing at the time. The best play resulted from setting $\eta = 0.5$, indicating that combining the value network with rollouts was particularly important to *AlphaGo*'s success. These evaluation methods complemented one another: the value network evaluated the high-performance RL policy that was too slow to be used in live play, while rollouts using the weaker but much faster rollout policy were able to add precision to the value network's evaluations for specific states that occurred during games.

Overall, *AlphaGo*'s remarkable success fueled a new round of enthusiasm for the promise of artificial intelligence, specifically for systems combining reinforcement learning with deep ANNs, to address problems in other challenging domains.

16.6.2 AlphaGo Zero

Building upon the experience with *AlphaGo*, a DeepMind team developed *AlphaGo Zero* (Silver et al. 2017a). In contrast to *AlphaGo*, this program used *no human data or guidance beyond the basic rules of the game* (hence the *Zero* in its name). It learned exclusively from self-play reinforcement learning, with input giving just "raw" descriptions of the placements of stones on the Go board. *AlphaGo Zero* implemented a form of policy iteration (Section 4.3), interleaving policy evaluation with policy improvement. Figure 16.7 is an overview of *AlphaGo Zero*'s algorithm. A significant difference between *AlphaGo Zero* and *AlphaGo* is that *AlphaGo Zero* used MCTS to select moves throughout self-play reinforcement learning, whereas *AlphaGo* used MCTS for live play after—but not during—learning. Other differences besides not using any human data or human-crafted features are that *AlphaGo Zero* used only one deep convolutional ANN and used a simpler version of MCTS.

AlphaGo Zero's MCTS was simpler than the version used by *AlphaGo* in that it did not include rollouts of complete games, and therefore did not need a rollout policy. Each iteration of *AlphaGo Zero*'s MCTS ran a simulation that ended at a leaf node of the current search tree instead of at the terminal position of a complete game simulation. But as in *AlphaGo*, each iteration of MCTS in *AlphaGo Zero* was guided by the output of a deep convolutional network, labeled f_θ in Figure 16.7, where θ is the network's weight vector. The input to the network, whose architecture we describe below, consisted of raw representations of board positions, and its output had two parts: a scalar value, v, an estimate of the probability that the current player will win from from the current board position, and a vector, $\mathbf{p}$, of move probabilities, one for each possible stone placement on the current board, plus the pass, or resign, move.

Instead of selecting self-play actions according to the probabilities $\mathbf{p}$, however, *AlphaGo Zero* used these probabilities, together with the network's value output, to direct each execution of MCTS, which returned new move probabilities, shown in Figure 16.7 as the policies π_i. These policies benefitted from the many simulations that MCTS conducted each time it executed. The result was that the policy actually followed by *AlphaGo Zero* was an improvement over the policy given by the network's outputs $\mathbf{p}$. Silver et al. (2017a) wrote that "MCTS may therefore be viewed as a powerful *policy improvement* operator."

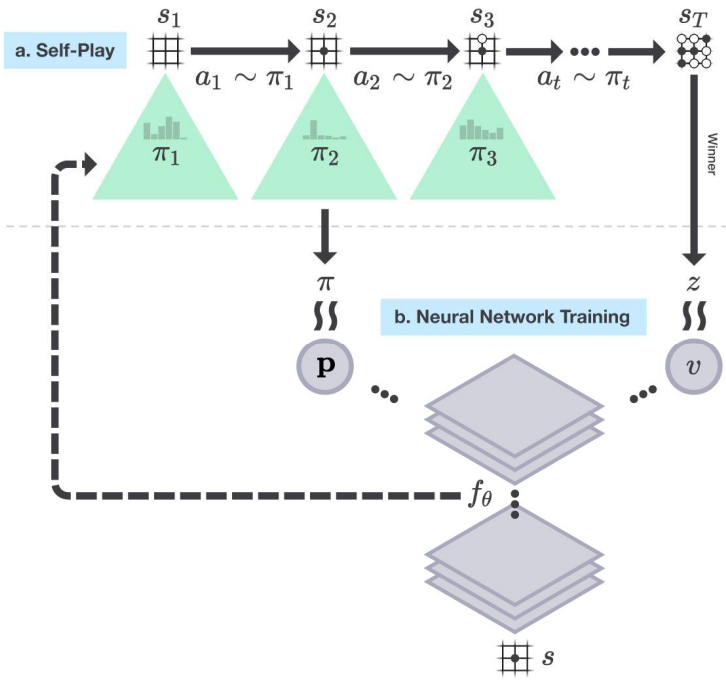


Figure 16.7: *AlphaGo Zero* self-play reinforcement learning. a) The program played many games against itself, one shown here as a sequence of board positions s_i, $i = 1, 2, \ldots, T$, with moves a_i, $i = 1, 2, \ldots, T$, and winner z. Each move a_i was determined by action probabilities π_i returned by MCTS executed from root node s_i and guided by a deep convolutional network, here labeled f_θ, with latest weights θ. Shown here for just one position s but repeated for all s_i, the network's inputs were raw representations of board positions s_i (together with several past positions, though not shown here), and its outputs were vectors $\mathbf{p}$ of move probabilities that guided MCTS's forward searches, and scalar values v that estimated the probability of the current player winning from each position s_i. b) Deep convolutional network training. Training examples were randomly sampled steps from recent self-play games. Weights θ were updated to move the policy vector $\mathbf{p}$ toward the probabilities π returned by MCTS, and to include the winners z in the estimated win probability v. Reprinted from draft of Silver et al. (2017a) with permission of the authors and DeepMind.

Here is more detail about *AlphaGo Zero*'s ANN and how it was trained. The network took as input a $19 \times 19 \times 17$ image stack consisting of 17 binary feature planes. The first 8 feature planes were raw representations of the positions of the current player's stones in the current and seven past board configurations: a feature value was 1 if a player's stone was on the corresponding point, and was 0 otherwise. The next 8 feature planes similarly coded the positions of the opponent's stones. A final input feature plane had a constant value indicating the color of the current play: 1 for black; 0 for white. Because repetition is not allowed in Go and one player is given some number of "compensation points" for not getting the first move, the current board position is not a Markov state of Go. This is why features describing past board positions and the color feature were needed.

The network was "two-headed," meaning that after a number of initial layers, the network split into two separate "heads" of additional layers that separately fed into two sets of output units. In this case, one head fed 362 output units producing $19^2 + 1$ move

probabilities $\mathbf{p}$, one for each possible stone placement plus pass; the other head fed just one output unit producing the scalar v, an estimate of the probability that the current player will win from the current board position. The network before the split consisted of 41 convolutional layers, each followed by batch normalization, and with skip connections added to implement residual learning by pairs of layers (see Section 9.7). Overall, move probabilities and values were computed by 43 and 44 layers respectively.

Starting with random weights, the network was trained by stochastic gradient descent (with momentum, regularization, and step-size parameter decreasing as training continues) using batches of examples sampled uniformly at random from all the steps of the most recent 500,000 games of self-play with the current best policy. Extra noise was added to the network's output $\mathbf{p}$ to encourage exploration of all possible moves. At periodic checkpoints during training, which Silver et al. (2017a) chose to be at every 1,000 training steps, the policy output by the ANN with the latest weights was evaluated by simulating 400 games (using MCTS with 1,600 iterations to select each move) against the current best policy. If the new policy won (by a margin set to reduce noise in the outcome), then it became the best policy to be used in subsequent self-play. The network's weights were updated to make the network's policy output $\mathbf{p}$ more closely match the policy returned by MCTS, and to make its value output, v, more closely match the probability that the current best policy wins from the board position represented by the network's input.

The DeepMind team trained *AlphaGo Zero* over 4.9 million games of self-play, which took about 3 days. Each move of each game was selected by running MCTS for 1,600 iterations, taking approximately 0.4 second per move. Network weights were updated over 700,000 batches each consisting of 2,048 board configurations. They then ran tournaments with the trained *AlphaGo Zero* playing against the version of *AlphaGo* that defeated Fan Hui by 5 games to 0, and against the version that defeated Lee Sedol by 4 games to 1. They used the Elo rating system to evaluate the relative performances of the programs. The difference between two Elo ratings is meant to predict the outcome of games between the players. The Elo ratings of *AlphaGo Zero*, the version of *AlphaGo* that played against Fan Hui, and the version that played against Lee Sedol were respectively 4,308, 3,144, and 3,739. The gaps in these Elo ratings translate into predictions that *AlphaGo Zero* would defeat these other programs with probabilities very close to one. In a match of 100 games between *AlphaGo Zero*, trained as described, and the exact version of *AlphaGo* that defeated Lee Sedol held under the same conditions that were used in that match, *AlphaGo Zero* defeated *AlphaGo* in all 100 games.

The DeepMind team also compared *AlphaGo Zero* with a program using an ANN with the same architecture but trained by supervised learning to predict human moves in a data set containing nearly 30 million positions from 160,000 games. They found that the supervised-learning player initially played better than *AlphaGo Zero*, and was better at predicting human expert moves, but played less well after *AlphaGo Zero* was trained for a day. This suggested that *AlphaGo Zero* had discovered a strategy for playing that was different from how humans play. In fact, *AlphaGo Zero* discovered, and came to prefer, some novel variations of classical move sequences.

Final tests of *AlphaGo Zero*'s algorithm were conducted with a version having a larger ANN and trained over 29 million self-play games, which took about 40 days, again starting

with random weights. This version achieved an Elo rating of 5,185. The team pitted this version of *AlphaGo Zero* against a program called *AlphaGo Master*, the strongest program at the time, that was identical to *AlphaGo Zero* but, like *AlphaGo*, used human data and features. *AlphaGo Master*'s Elo rating was 4,858, and it had defeated the strongest human professional players 60 to 0 in online games. In a 100 game match, *AlphaGo Zero* with the larger network and more extensive learning defeated *AlphaGo Master* 89 games to 11, thus providing a convincing demonstration of the problem-solving power of *AlphaGo Zero*'s algorithm.

AlphaGo Zero soundly demonstrated that superhuman performance can be achieved by pure reinforcement learning, augmented by a simple version of MCTS, and deep ANNs with very minimal knowledge of the domain and no reliance on human data or guidance. We will surely see systems inspired by the DeepMind accomplishments of both *AlphaGo* and *AlphaGo Zero* applied to challenging problems in other domains.

Recently, yet a better program, *AlphaZero*, was described by Silver et al. (2017b) that does not even incorporate knowledge of Go. *AlphaZero* is a general reinforcement learning algorithm that improves over the world's hitherto best programs in the diverse games of Go, chess, and shogi.

16.7 Personalized Web Services

Personalizing web services such as the delivery of news articles or advertisements is one approach to increasing users' satisfaction with a website or to increase the yield of a marketing campaign. A policy can recommend content considered to be the best for each particular user based on a profile of that user's interests and preferences inferred from their history of online activity. This is a natural domain for machine learning, and in particular, for reinforcement learning. A reinforcement learning system can improve a recommendation policy by making adjustments in response to user feedback. One way to obtain user feedback is by means of website satisfaction surveys, but for acquiring feedback in real time it is common to monitor user clicks as indicators of interest in a link.

A method long used in marketing called *A/B testing* is a simple type of reinforcement learning used to decide which of two versions, A or B, of a website users prefer. Because it is non-associative, like a two-armed bandit problem, this approach does not personalize content delivery. Adding context consisting of features describing individual users and the content to be delivered allows personalizing service. This has been formalized as a contextual bandit problem (or an associative reinforcement learning problem, Section 2.9) with the objective of maximizing the total number of user clicks. Li, Chu, Langford, and Schapire (2010) applied a contextual bandit algorithm to the problem of personalizing the Yahoo! Front Page Today webpage (one of the most visited pages on the internet at the time of their research) by selecting the news story to feature. Their objective was to maximize the *click-through rate* (CTR), which is the ratio of the total number of clicks all users make on a webpage to the total number of visits to the page. Their contextual bandit algorithm improved over a standard non-associative bandit algorithm by 12.5%.

Theocharous, Thomas, and Ghavamzadeh (2015) argued that better results are possible

by formulating personalized recommendation as a Markov decision problem (MDP) with the objective of maximizing the total number of clicks users make over repeated visits to a website. Policies derived from the contextual bandit formulation are greedy in the sense that they do not take long-term effects of actions into account. These policies effectively treat each visit to a website as if it were made by a new visitor uniformly sampled from the population of the website's visitors. By not using the fact that many users repeatedly visit the same websites, greedy policies do not take advantage of possibilities provided by long-term interactions with individual users.

As an example of how a marketing strategy might take advantage of long-term user interaction, Theocharous et al. contrasted a greedy policy with a longer-term policy for displaying ads for buying a product, say a car. The ad displayed by the greedy policy might offer a discount if the user buys the car immediately. A user either takes the offer or leaves the website, and if they ever return to the site, they would likely see the same offer. A longer-term policy, on the other hand, can transition the user "down a sales funnel" before presenting the final deal. It might start by describing the availability of favorable financing terms, then praise an excellent service department, and then, on the next visit, offer the final discount. This type of policy can result in more clicks by a user over repeated visits to the site, and if the policy is suitably designed, more eventual sales.

Working at Adobe Systems Incorporated, Theocharous et al. conducted experiments to see if policies designed to maximize clicks over the long term could in fact improve over short-term greedy policies. The Adobe Marketing Cloud, a set of tools that many companies use to run digital marketing campaigns, provides infrastructure for automating user-targeted advertising and fund-raising campaigns. Actually deploying novel policies using these tools entails significant risk because a new policy may end up performing poorly. For this reason, the research team needed to assess what a policy's performance would be if it were to be actually deployed, but to do so on the basis of data collected under the execution of other policies. A critical aspect of this research, then, was off-policy evaluation. Further, the team wanted to do this with high confidence to reduce the risk of deploying a new policy. Although high confidence off-policy evaluation was a central component of this research (see also Thomas, 2015; Thomas, Theocharous, and Ghavamzadeh, 2015), here we focus only on the algorithms and their results.

Theocharous et al. compared the results of two algorithms for learning ad recommendation policies. The first algorithm, which they called *greedy optimization*, had the goal of maximizing only the probability of immediate clicks. As in the standard contextual bandit formulation, this algorithm did not take the long-term effects of recommendations into account. The other algorithm, a reinforcement learning algorithm based on an MDP formulation, aimed at improving the number of clicks users made over multiple visits to a website. They called this latter algorithm *life-time value* (LTV) optimization. Both algorithms faced challenging problems because the reward signal in this domain is very sparse because users usually do not click on ads, and user clicking is very random so that returns have high variance.

Data sets from the banking industry were used for training and testing these algorithms. The data sets consisted of many complete trajectories of customer interaction with a bank's website that showed each customer one out of a collection of possible offers. If

a customer clicked, the reward was 1, and otherwise it was 0. One data set contained approximately 200,000 interactions from a month of a bank's campaign that randomly offered one of 7 offers. The other data set from another bank's campaign contained 4,000,000 interactions involving 12 possible offers. All interactions included customer features such as the time since the customer's last visit to the website, the number of their visits so far, the last time the customer clicked, geographic location, one of a collection of interests, and features giving demographic information.

Greedy optimization was based on a mapping estimating the probability of a click as a function of user features. The mapping was learned via supervised learning from one of the data sets by means of a random forest (RF) algorithm (Breiman, 2001). RF algorithms have been widely used for large-scale applications in industry because they are effective predictive tools that tend not to overfit and are relatively insensitive to outliers and noise. Theocharous et al. then used the mapping to define an ε-greedy policy that selected with probability 1-ε the offer predicted by the RF algorithm to have the highest probability of producing a click, and otherwise selected from the other offers uniformly at random.

LTV optimization used a batch-mode reinforcement learning algorithm called *fitted Q iteration* (FQI). It is a variant of *fitted value iteration* (Gordon, 1999) adapted to Q-learning. Batch mode means that the entire data set for learning is available from the start, as opposed to the online mode of the algorithms we focus on in this book in which data are acquired sequentially while the learning algorithm executes. Batch-mode reinforcement learning algorithms are sometimes necessary when online learning is not practical, and they can use any batch-mode supervised learning regression algorithm, including algorithms known to scale well to high-dimensional spaces. The convergence of FQI depends on properties of the function approximation algorithm (Gordon, 1999). For their application to LTV optimization, Theocharous et al. used the same RF algorithm they used for the greedy optimization approach. Because in this case FQI convergence is not monotonic, Theocharous et al. kept track of the best FQI policy by off-policy evaluation using a validation training set. The final policy for testing the LTV approach was the ε-greedy policy based on the best policy produced by FQI with the initial action-value function set to the mapping produced by the RF for the greedy optimization approach.

To measure the performance of the policies produced by the greedy and LTV approaches, Theocharous et al. used the CTR metric and a metric they called the LTV metric. These metrics are similar, except that the LTV metric critically distinguishes between individual website visitors:

$$\text{CTR} = \frac{\text{Total \# of Clicks}}{\text{Total \# of Visits}},$$

$$\text{LTV} = \frac{\text{Total \# of Clicks}}{\text{Total \# of Visitors}}.$$

Figure 16.8 illustrates how these metrics differ. Each circle represents a user visit to the site; black circles are visits at which the user clicks. Each row represents visits by a particular user. By not distinguishing between visitors, the CTR for these sequences is 0.35, whereas the LTV is 1.5. Because LTV is larger than CTR to the extent that

individual users revisit the site, it is an indicator of how successful a policy is in encouraging users to engage in extended interactions with the site.

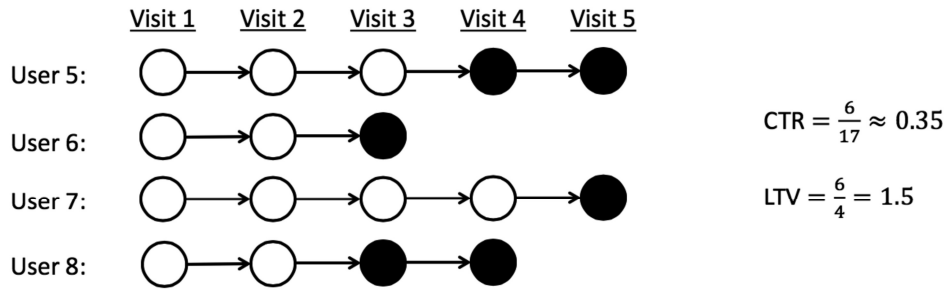


Figure 16.8: Click through rate (CTR) versus life-time value (LTV). Each circle represents a user visit; black circles are visits at which the user clicks. Adapted from Theocharous et al. (2015).

Testing the policies produced by the greedy and LTV approaches was done using a high confidence off-policy evaluation method on a test data set consisting of real-world interactions with a bank website served by a random policy. As expected, results showed that greedy optimization performed best as measured by the CTR metric, while LTV optimization performed best as measured by the LTV metric. Furthermore—although we have omitted its details—the high confidence off-policy evaluation method provided probabilistic guarantees that the LTV optimization method would, with high probability, produce policies that improve upon policies currently deployed. Assured by these probabilistic guarantees, Adobe announced in 2016 that the new LTV algorithm would be a standard component of the Adobe Marketing Cloud so that a retailer could issue a sequence of offers following a policy likely to yield higher return than a policy that is insensitive to long-term results.

16.8 Thermal Soaring

Birds and gliders take advantage of upward air currents—thermals—to gain altitude in order to maintain flight while expending little, or no, energy. Thermal soaring, as this behavior is called, is a complex skill requiring responding to subtle environmental cues to increase altitude by exploiting a rising column of air for as long as possible. Reddy, Celani, Sejnowski, and Vergassola (2016) used reinforcement learning to investigate thermal soaring policies that are effective in the strong atmospheric turbulence usually accompanying rising air currents. Their primary goal was to provide insight into the cues birds sense and how they use them to achieve their impressive thermal soaring performance, but the results also contribute to technology relevant to autonomous gliders. Reinforcement learning had previously been applied to the problem of navigating efficiently to the vicinity of a thermal updraft (Woodbury, Dunn, and Valasek, 2014) but not to the more challenging problem of soaring within the turbulence of the updraft itself.

Reddy et al. modeled the soaring problem as a continuing MDP with discounting. The agent interacted with a detailed model of a glider flying in turbulent air. They

devoted significant effort toward making the model generate realistic thermal soaring conditions, including investigating several different approaches to atmospheric modeling. For the learning experiments, air flow in a three-dimensional box with one kilometer sides, one of which was at ground level, was modeled by a sophisticated physics-based set of partial differential equations involving air velocity, temperature, and pressure. Introducing small random perturbations into the numerical simulation caused the model to produce analogs of thermal updrafts and accompanying turbulence (Figure 16.9 Left) Glider flight was modeled by aerodynamic equations involving velocity, lift, drag, and other factors governing powerless flight of a fixed-wing aircraft. Maneuvering the glider involved changing its angle of attack (the angle between the glider's wing and the direction of air flow) and its bank angle (Figure 16.9 Right).

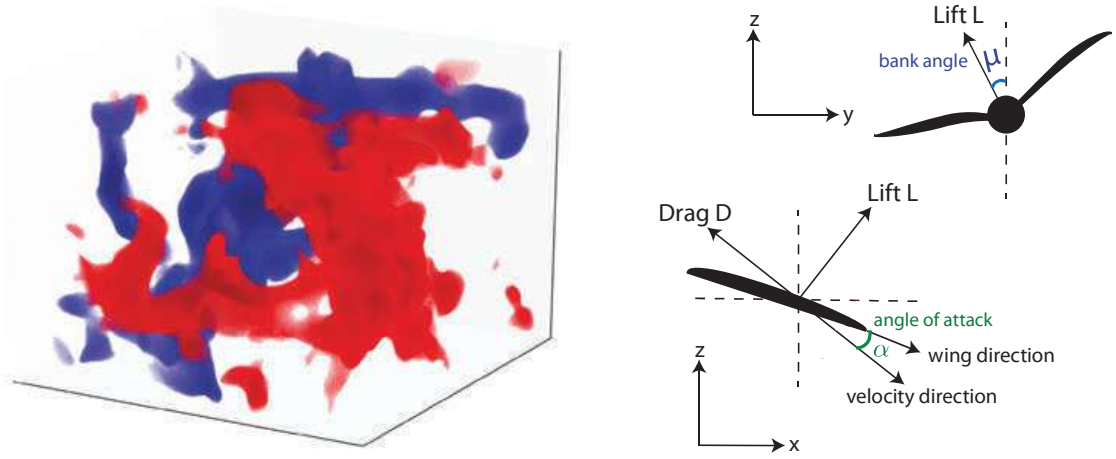


Figure 16.9: Thermal soaring model: Left: snapshot of the vertical velocity field of the simulated cube of air: in red (blue) is a region of large upward (downward) flow. Right: diagram of powerless flight showing bank angle μ and angle of attack α. Adapted with permission From PNAS vol. 113(22), p. E4879, 2016, Reddy, Celani, Sejnowski, and Vergassola, Learning to Soar in Turbulent Environments.

The interface between the agent and the environment required defining the agent's actions, the state information the agent receives from the environment, and the reward signal. By experimenting with various possibilities, Reddy et al. decided that three actions each for the angle of attack and the bank angle were enough for their purposes: increment or decrement the current bank angle and angle of attack by 5° and 2.5°, respectively, or leave them unchanged. This resulted in 3^2 possible actions. The bank angle was bounded to remain between -15° and $+15^\circ$.

Because a goal of their study was to try to determine what minimal set of sensory cues are necessary for effective soaring, both to shed light on the cues birds might use for soaring and to minimize the sensing complexity required for automated glider soaring, the authors tried various sets of signals as input to the reinforcement learning agent. They started by using state aggregation (Section 9.3) of a four-dimensional state space with dimensions giving local vertical wind speed, local vertical wind acceleration, torque depending on the difference between the vertical wind velocities at the left and right wing tips, and the local temperature. Each dimension was discretized into three bins: positive

high, negative high, and small. Results, described below, showed that only two of these dimensions were critical for effective soaring behavior.

The overall objective of thermal soaring is to gain as much altitude as possible from each rising column of air. Reddy et al. tried a straightforward reward signal that rewarded the agent at the end of each episode based on the altitude gained over the episode, a large negative reward signal if the glider touched the ground, and zero otherwise. They found that learning was not successful with this reward signal for episodes of realistic duration and that eligibility traces did not help. By experimenting with various reward signals, they found that learning was best with a reward signal that at each time step linearly combined the vertical wind velocity and vertical wind acceleration observed on the previous time step.

Learning was by one-step Sarsa, with actions selected according to a soft-max distribution based on normalized action values. Specifically, the action probabilities were computed according to (13.2) with action preferences:

$$h(s,a,\boldsymbol{\theta}) = \frac{\hat{q}(s,a,\boldsymbol{\theta}) - \min_b \hat{q}(s,b,\boldsymbol{\theta})}{\tau\big(\max_b \hat{q}(s,b,\boldsymbol{\theta}) - \min_b \hat{q}(s,b,\boldsymbol{\theta})\big)},$$

where $\boldsymbol{\theta}$ is a parameter vector with one component for each action and aggregated group of states, and $\hat{q}(s,a,\boldsymbol{\theta})$ merely returned the component corresponding to s,a in the usual way for state aggregation methods. The above equation forms the action preferences by normalizing the approximate action values to the interval $[0,1]$ then dividing by τ, a positive "temperature parameter."[3] As τ increases, the probability of selecting an action becomes less dependent on its preference; as τ decreases toward zero, the probability of selecting the most highly-preferred action approaches one, making the policy approach the greedy policy. The temperature parameter τ was initialized to 2.0 and incrementally decreased to 0.2 during learning. Action preferences were computed from the current estimates of the action values: the action with the maximum estimated action value was given preference $1/\tau$, the action with the minimum estimated action value was given preference 0, and the preferences of the other actions were scaled between these extremes. The step-size and discount-rate parameters were fixed at 0.1 and 0.98 respectively.

Each learning episode took place with the agent controlling simulated flight in an independently generated period of simulated turbulent air currents. Each episode lasted 2.5 minutes simulated with a 1 second time step. Learning effectively converged after a few hundred episodes. The left panel of Figure 16.10 shows a sample trajectory before learning where the agent selects actions randomly. Starting at the top of the volume shown, the glider's trajectory is in the direction indicated by the arrow and quickly loses altitude. Figure 16.10's right panel is a trajectory after learning. The glider starts at the same place (here appearing at the bottom of the volume) and gains altitude by spiraling within the rising column of air. Although Reddy et al. found that performance varied widely over different simulated periods of air flow, the number of times the glider touched the ground consistently decreased to nearly zero as learning progressed.

After experimenting with different sets of features available to the learning agent, it turned out that the combination of just vertical wind acceleration and torques worked

[3] Reddy et al. described this slightly differently, but our version is equivalent to theirs.

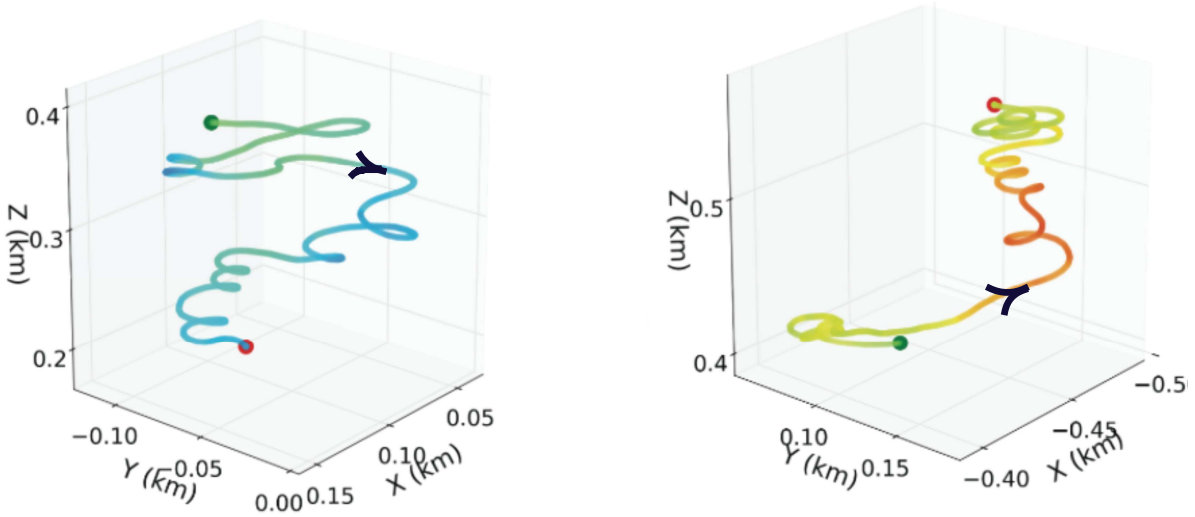


Figure 16.10: Sample thermal soaring trajectories, with arrows showing the direction of flight from the same starting point (note that the altitude scales are shifted). Left: before learning: the agent selects actions randomly and the glider descends. Right: after learning: the glider gains altitude by following a spiral trajectory. Adapted with permission from PNAS vol. 113(22), p. E4879, 2016, Reddy, Celani, Sejnowski, and Vergassola, Learning to Soar in Turbulent Environments.

best. The authors conjectured that because these features give information about the gradient of vertical wind velocity in two different directions, they allow the controller to select between turning by changing the bank angle or continuing along the same course by leaving the bank angle alone. This allows the glider to stay within a rising column of air. Vertical wind velocity is indicative of the strength of the thermal but does not help in staying within the flow. They found that sensitivity to temperature was of little help. They also found that controlling the angle of attack is not helpful in staying within a particular thermal, being useful instead for traveling between thermals when covering large distances, as in cross-country gliding and bird migration.

Due to the fact that soaring in different levels of turbulence requires different policies, training was done in conditions ranging from weak to strong turbulence. In strong turbulence the rapidly changing wind and glider velocities allowed less time for the controller to react. This reduced the amount of control possible compared to what was possible for maneuvering when fluctuations were weak. Reddy et al. examined the policies Sarsa learned under these different conditions. Common to policies learned in all regimes were these features: when sensing negative wind acceleration, bank sharply in the direction of the wing with the higher lift; when sensing large positive wind acceleration and no torque, do nothing. However, different levels of turbulence led to policy differences. Policies learned in strong turbulence were more conservative in that they preferred small bank angles, whereas in weak turbulence, the best action was to turn as much as possible by banking sharply. Systematic study of the bank angles preferred by the policies learned under the different conditions led the authors to suggest that by detecting when vertical

wind acceleration crosses a certain threshold the controller can adjust its policy to cope with different turbulence regimes.

Reddy et al. also conducted experiments to investigate the effect of the discount-rate parameter γ on the performance of the learned policies. They found that the altitude gained in an episode increased as γ increased, reaching a maximum for $\gamma = .99$, suggesting that effective thermal soaring requires taking into account long-term effects of control decisions.

This computational study of thermal soaring illustrates how reinforcement learning can further progress toward different kinds of objectives. Learning policies having access to different sets of environmental cues and control actions contributes to both the engineering objective of designing autonomous gliders and the scientific objective of improving understanding of the soaring skills of birds. In both cases, hypotheses resulting from the learning experiments can be tested in the field by instrumenting real gliders[4] and by comparing predictions with observed bird soaring behavior.

[4] This work has recently been applied to real gliders. See Reddy, Wong-Ng, Celani, Sejnowski, and Vergassola, "Glider soaring via reinforcement learning in the field." *Nature 562*:236–239, 2018.

Chapter 17

Frontiers

In this final chapter we touch on some topics that are beyond the scope of this book but that we see as particularly important for the future of reinforcement learning. Many of these topics bring us beyond what is reliably known, and some bring us beyond the MDP framework.

17.1 General Value Functions and Auxiliary Tasks

Over the course of this book, our notion of value function has become quite general. With off-policy learning we allowed a value function to be conditional on an arbitrary target policy. Then in Section 12.8 we generalized discounting to a *termination function* $\gamma : \mathcal{S} \mapsto [0,1]$, so that a different discount rate could be applied at each time step in determining the return (12.17). This allowed us to express predictions about how much reward we will get over an arbitrary, state-dependent horizon. The next, and perhaps final, step is to generalize beyond rewards to permit predictions about arbitrary signals. Rather than predicting the sum of future rewards, we might predict the sum of the future values of a sound or color sensation, or of an internal, highly processed signal such as another prediction. Whatever signal is added up in this way in a value-function-like prediction, we call it the *cumulant* of that prediction. We formalize it in a *cumulant signal* $C_t \in \mathbb{R}$. Using this, a *general value function*, or GVF, is written

$$v_{\pi,\gamma,C}(s) \doteq \mathbb{E}\left[\sum_{k=t}^{\infty}\left(\prod_{i=t+1}^{k}\gamma(S_i)\right)C_{k+1} \;\middle|\; S_t{=}s, A_{t:\infty}{\sim}\pi\right]. \tag{17.1}$$

As with conventional value functions (such as v_π or q_*) this is an ideal function that we seek to approximate with a parameterized form, which we might continue to denote $\hat{v}(s,\mathbf{w})$, although of course there would have to be a different $\mathbf{w}$ for each prediction, that is, for each choice of π, γ, and C. Because a GVF has no necessary connection to reward, it is perhaps a misnomer to call it a *value* function. You might simply call it a prediction or, to make it more distinctive, a *forecast* (Ring, in preparation). Whatever it is called, it

is in the form of a value function and thus can be learned in the usual ways using the methods developed in this book for learning approximate value functions. Along with the learned predictions, we might also learn policies to maximize the predictions in the usual ways by Generalized Policy Iteration (Section 4.6) or by actor–critic methods. In this way an agent could learn to predict and control great numbers of signals, not just long-term reward.

Why might it be useful to predict and control signals other than long-term reward? These are *auxiliary* tasks in that they are extra (in addition to) the main task of maximizing reward. One answer is that the ability to predict and control a diverse multitude of signals can constitute a powerful kind of environmental model. As we saw in Chapter 8, a good model can enable the agent to get reward more efficiently. It takes a couple of further concepts to develop this answer clearly, so we postpone it to the next section. First let's consider two simpler ways in which a multitude of diverse predictions can be helpful to a reinforcement learning agent.

One simple way in which auxiliary tasks can help on the main task is that they may require some of the same representations as are needed on the main task. Some of the auxiliary tasks may be easier, with less delay and a clearer connection between actions and outcomes. If good features can be found early on easy auxiliary tasks, then those features may significantly speed learning on the main task. There is no necessary reason why this has to be true, but in many cases it seems plausible. For example, if you learn to predict and control your sensors over short time scales, say seconds, then you might plausibly come up with part of the idea of physical objects, which would then greatly help with the prediction and control of long-term reward.

We might imagine an artificial neural network (ANN) in which the last layer is split into multiple parts, or *heads*, each working on a different task. One head might produce the approximate value function for the main task (with reward as its cumulant) whereas the others would produce solutions to various auxiliary tasks. All heads could propagate errors by stochastic gradient descent into the same body—the shared preceding part of the network—which would then try to form representations, in its next-to-last layer, to support all the heads. Researchers have experimented with auxiliary tasks such as predicting change in pixels, predicting the next time step's reward, and predicting the distribution of the return. In many cases this approach has been shown to greatly accelerate learning on the main task (Jaderberg et al., 2017). Multiple predictions have similarly been repeatedly proposed as a way of directing the construction of state estimates (see Section 17.3).

Another simple way in which the learning of auxiliary tasks can improve performance is best explained by analogy to the psychological phenomena of classical conditioning (Section 14.2). One way of understanding classical conditioning is that evolution has built in a reflexive (non-learned) association to a particular action from the prediction of a particular signal. For example, humans and many other animals appear to have a built-in reflex to blink whenever their prediction of being poked in the eye exceeds some threshold. The prediction is learned, but the association from prediction to eye closure is built in, and thus the animal is saved many unprotected pokes in its eye. Similarly, the association from fear to increased heart rate, or to freezing, may be built in. Agent

designers can do something similar, connecting by design (without learning) predictions of specific events to predetermined actions. For example, a self-driving car that learns to predict whether going forward will produce a collision could be given a built-in reflex to stop, or to turn away, whenever the prediction is above some threshold. Or consider a vacuum-cleaning robot that learned to predict whether it might run out of battery power before returning to the charger and that reflexively headed back to the charger whenever the prediction became non-zero. The correct prediction would depend on the size of the house, the room the robot was in, and the age of the battery, all of which would be hard for the robot designer to know. It would be difficult for the designer to build in a reliable algorithm for deciding whether to head back to the charger in sensory terms, but it might be easy to do this in terms of the learned prediction. We foresee many possible ways like this in which learned predictions might combine usefully with built-in algorithms for controlling behavior.

Finally, perhaps the most important role for auxiliary tasks is in moving beyond the assumption we have made throughout this book that the state representation is fixed and given to the agent. To explain this role, we first have to take a few steps back to appreciate the magnitude of this assumption and the implications of removing it. We do that in Section 17.3.

17.2 Temporal Abstraction via Options

An appealing aspect of the MDP formalism is that it can be applied usefully to tasks at many different time scales. It can be used to formalize the task of deciding which muscles to twitch to grasp an object, which airplane flight to take to arrive conveniently at a distant city, and which job to take to lead a satisfying life. These tasks differ greatly in their time scales, yet each can be usefully formulated as an MDP that can be solved by planning or learning processes as described in this book. All involve interaction with the world, sequential decision making, and a goal usefully conceived of as accumulating rewards over time, and so all can be formulated as MDPs.

Although all these tasks can be formulated as MDPs, you might think that they cannot be formulated as a *single* MDP. They involve such different time scales, such different notions of choice and action! It would be no good, for example, to plan a flight across a continent at the level of muscle twitches. Yet for other tasks—such as grasping objects, throwing darts, or hitting a baseball—low-level muscle twitches may be just the right level. People do all these things seamlessly without appearing to switch between levels. Can the MDP framework be stretched to cover all the levels simultaneously?

Perhaps it can. One popular idea is to formalize an MDP at a detailed level, with a small time step, yet enable planning at higher levels using extended courses of action that correspond to many base-level time steps. To do this we need a notion of course of action that extends over many time steps and includes a notion of termination. A general way to formulate these two ideas is as a policy, π, and a state-dependent termination function, γ, as in GVFs. We define a pair of these as a generalized notion of action termed an *option*. To execute an option $\omega = \langle \pi_\omega, \gamma_\omega \rangle$ at time t is to obtain the action to take, A_t, from $\pi_\omega(\cdot|S_t)$, then terminate at time $t+1$ with probability $1 - \gamma_\omega(S_{t+1})$. If the option does

not terminate at $t+1$, then A_{t+1} is selected from $\pi_\omega(\cdot|S_{t+1})$, and the option terminates at $t+2$ with probability $1-\gamma_\omega(S_{t+2})$, and so on until eventual termination. It is convenient to consider low-level actions to be special cases of options—each action a corresponds to an option $\langle \pi_\omega, \gamma_\omega \rangle$ whose policy picks the action ($\pi_\omega(s)=a$ for all $s \in \mathcal{S}$) and whose termination function is zero ($\gamma_\omega(s)=0$ for all $s \in \mathcal{S}^+$). Options effectively extend the action space. The agent can either select a low-level action/option, terminating after one time step, or select an extended option that might execute for many time steps before terminating.

Options are designed so that they are interchangable with low-level actions. For example, the notion of an action-value function q_π naturally generalizes to an *option-value function* that takes a state and option as input and returns the expected return starting from that state, executing that option to termination, and thereafter following the policy, π. We can also generalize the notion of policy to a *hierarchical policy* that selects from options rather than actions, where options, when selected, execute until termination. With these ideas, many of the algorithms in this book can be generalized to learn approximate option-value functions and hierarchical policies. In the simplest case, the learning process 'jumps' from option initiation to option termination, with an update only occurring when an option terminates. More subtly, updates can be made on each time step, using "intra-option" learning algorithms, which in general require off-policy learning.

Perhaps the most important generalization made possible by option ideas is that of the environmental model as developed in Chapters 3, 4, and 8. The conventional model of an action is the state-transition probabilities and the expected immediate reward for taking the action in each state. How do conventional action models generalize to *option models*? For options, the appropriate model is again of two parts, one corresponding to the state transition resulting from executing the option and one corresponding to the expected cumulative reward along the way. The reward part of an option model, analogous to the expected reward for state–action pairs (3.5), is

$$r(s,\omega) \doteq \mathbb{E}\big[R_1 + \gamma R_2 + \gamma^2 R_3 + \cdots + \gamma^{\tau-1} R_\tau \;\big|\; S_0=s, A_{0:\tau-1} \sim \pi_\omega, \tau \sim \gamma_\omega\big], \quad (17.2)$$

for all options ω and all states $s \in \mathcal{S}$, where τ is the random time step at which the option terminates according to γ_ω. Note the role of the overall discounting parameter γ in this equation—discounting is according to γ, but termination of the option is according to γ_ω. The state-transition part of an option model is a little more subtle. This part of the model characterizes the probability of each possible resulting state (as in (3.4)), but now this state may result after various numbers of time steps, each of which must be discounted differently. The model for option ω specifies, for each state s that ω might start executing in, and for each state s' that ω might terminate in,

$$p(s'|s,\omega) \doteq \sum_{k=1}^{\infty} \gamma^k \Pr\{S_k=s', \tau=k \mid S_0=s, A_{0:k-1} \sim \pi_\omega, \tau \sim \gamma_\omega\}. \quad (17.3)$$

Note that, because of the factor of γ^k, this $p(s'|s,\omega)$ is no longer a transition probability and no longer sums to one over all values of s'. (Nevertheless, we continue to use the '|' notation in p.)

The above definition of the transition part of an option model allows us to formulate Bellman equations and dynamic programming algorithms that apply to all options, including low-level actions as a special case. For example, the general Bellman equation for the state values of a hierarchical policy π is

$$v_\pi(s) = \sum_{\omega \in \Omega(s)} \pi(\omega|s) \left[r(s,\omega) + \sum_{s'} p(s' \,|\, s,\omega) v_\pi(s') \right], \tag{17.4}$$

where $\Omega(s)$ denotes the set of options available in state s. If $\Omega(s)$ includes only the low-level actions, then this equation reduces to a version of the usual Bellman equation (3.14), except of course γ is included in the new p (17.3) and thus does not appear. Similarly, the corresponding planning algorithms also have no γ. For example, the value iteration algorithm with options, analogous to (4.10), is

$$v_{k+1}(s) \doteq \max_{\omega \in \Omega(s)} \left[r(s,\omega) + \sum_{s'} p(s' \,|\, s,\omega) v_k(s') \right], \text{ for all } s \in \mathcal{S}.$$

If $\Omega(s)$ includes all the low-level actions available in each state s, then this algorithm converges to the conventional v_*, from which the optimal policy can be computed. However, it is particularly useful to plan with options when only a subset of the possible options are considered (in $\Omega(s)$) in each state. Value iteration will then converge to the best hierarchical policy limited to the restricted set of options. Although this policy may be sub-optimal, convergence can be much faster because fewer options are considered and because each option can jump over many time steps.

To plan with options, the agent must either be given the option models, or learn them. One natural way to learn an option model is to formulate it as a collection of GVFs (as defined in the preceding section) and then learn the GVFs using the methods presented in this book. It is not difficult to see how this could be done for the reward part of the option model. You merely choose one GVF's cumulant to be the reward ($C_t = R_t$), its policy to be the option's policy ($\pi = \pi_\omega$), and its termination function to be the discount rate times the option's termination function ($\gamma(s) = \gamma \cdot \gamma_\omega(s)$). The true GVF then equals the reward part of the option model, $v_{\pi,\gamma,C}(s) = r(s,\omega)$, and the learning methods described in this book can be used to approximate it. The state-transition part of the option model is a little more complicated. You need to allocate one GVF for each state that the option might terminate in. We don't want these GVFs to accumulate anything except when the option terminates, and then only when termination is in the appropriate state. This can be achieved by choosing the cumulant of the GVF that predicts transition to state s' to be $C_t = (1 - \gamma_\omega(S_t))\mathbb{1}_{S_t = s'}$. The GVF's policy and termination functions are chosen the same as for the reward part of the option model. The true GVF then equals the s' portion of the option's state-transition model, $v_{\pi,\gamma,C}(s) = p(s' \,|\, s,\omega)$, and again this book's methods could be employed to learn it. Although each of these steps is seemingly natural, putting them all together (including function approximation and other essential components) is quite challenging and beyond the current state of the art.

Exercise 17.1 This section has presented options for the discounted case, but discounting is arguably inappropriate for control when using function approximation (Section 10.4). What is the natural Bellman equation for a hierarchical policy, analogous to (17.4), but for the average reward setting (Section 10.3)? What are the two parts of the option model, analogous to (17.2) and (17.3), for the average reward setting? □

17.3 Observations and State

Throughout this book we have written the learned approximate value functions (and the policies in Chapter 13) as functions of the environment's state. This is a significant limitation of the methods presented in Part I, in which the learned value function was implemented as a table such that any value function could be exactly approximated; that case is tantamount to assuming that the state of the environment is completely observed by the agent. But in many cases of interest, and certainly in the lives of all natural intelligences, the sensory input gives only partial information about the state of the world. Some objects may be occluded by others, or behind the agent, or miles away. In these cases, potentially important aspects of the environment's state are not directly observable, and it is a strong, unrealistic, and limiting assumption to assume that the learned value function is implemented as a table over the environment's state space.

The framework of parametric function approximation that we developed in Part II is far less restrictive and, arguably, no limitation at all. In Part II we retained the assumption that the learned value functions (and policies) are functions of the environment's state, but allowed these functions to be arbitrarily restricted by the parameterization. It is somewhat surprising and not widely recognized that function approximation includes important aspects of partial observability. For example, if there is a state variable that is not observable, then the parameterization can be chosen such that the approximate value does not depend on that state variable. The effect is just as if the state variable were not observable. Because of this, all the results obtained for the parameterized case apply to partial observability without change. In this sense, the case of parameterized function approximation includes the case of partial observability.

Nevertheless, there are many issues that cannot be investigated without a more explicit treatment of partial observability. Although we cannot give them a full treatment here, we can outline the changes that would be needed to do so. There are four steps.

First, we would change the problem. The environment would emit not its states, but only *observations*—signals that depend on its state but, like a robot's sensors, provide only partial information about it. For convenience, without loss of generality, we assume that the reward is a direct, known function of the observation (perhaps the observation is a vector, and the reward is one of its components). The environmental interaction would then have no explicit states or rewards, but could simply be an alternating sequence of actions $A_t \in \mathcal{A}$ and observations $O_t \in \mathcal{O}$:

$$A_0, O_1, A_1, O_2, A_2, O_3, A_3, O_4, \ldots,$$

going on forever (cf. Equation 3.1) or forming episodes each ending with a special terminal observation.

Second, we can recover the idea of state as used in this book from the sequence of observations and actions. Let us use the word *history*, and the notation H_t, for an initial portion of the trajectory up to an observation: $H_t \doteq A_0, O_1, \ldots, A_{t-1}, O_t$. The history represents the most that we can know about the past without looking outside of the data stream (because the history is the whole past data stream). Of course, the history grows with t and can become large and unwieldy. The idea of state is that of a compact summary of the history that is useful for predicting future sequences. To be a summary of the history, a state must be a function of history, $S_t = f(H_t)$. The summary would be informationally perfect if it retained all information about the history (and thus could be used to predict futures as accurately as could be done from the full history). In this case, the state S_t and the function f are said to have the *Markov property*, and S_t is a state as we have used the term in this book. Let us henceforth call it a *Markov state* to distinguish it from states that are summaries of the history but are not sufficient to predict all futures. In practice, the states of real agents will not be Markov but may approach it as an ideal.

To be more explicit about the Markov property it is useful to formalize possible futures. Let a *test* be any specific sequence of alternating actions and observations that might occur in the future. For example, a three-step test might be denoted $\tau = a_1 o_1 a_2 o_2 a_3 o_3$. The probability of this test given a specific history h is defined as

$$p(\tau|h) \doteq \Pr\{O_{t+1}=o_1, O_{t+2}=o_2, O_{t+3}=o_3 \mid H_t=h, A_t=a_1, A_{t+1}=a_2, A_{t+2}=a_3\}. \tag{17.5}$$

Formally, f is Markov if and only if, for any test τ, and for any histories h and h' that map to the same state under f, the test's probabilities given the two histories are equal:

$$f(h) = f(h') \quad \Rightarrow \quad p(\tau|h) = p(\tau|h'), \qquad \text{for all } h, h', \tau \in \{\mathcal{A} \times \mathcal{O}\}^*. \tag{17.6}$$

A Markov state summarizes all the information in the history necessary for determining any test's probability. In fact, it summarizes all that is necessary for making *any prediction*, including any GVF. It also summarizes all that is necessary for optimal behavior: if f is Markov, then there is always a deterministic function π such that choosing $A_t \doteq \pi(f(H_t))$ is an optimal policy.

The third step in extending reinforcement learning to partial observability is to deal with certain computational considerations. As mentioned earlier, we want the state to be *compact*—relatively small compared to the history. (The identity function, for example, is not a good f even though it is Markov, because the corresponding state $S_t = H_t$ would grow unboundedly with time.) In addition, we don't really want a function f that takes whole histories. Instead, we want an f that can be compactly implemented with an incremental, recursive update that computes S_{t+1} from S_t, incorporating only the next increment of data, A_t and O_{t+1}:

$$S_{t+1} \doteq u(S_t, A_t, O_{t+1}), \text{ for all } t \geq 0, \tag{17.7}$$

with the first state S_0 given. The function u is called the *state-update* function. For example, if f were the identity ($S_t = H_t$), then u would merely extend S_t by appending A_t and O_{t+1} to it. Given f, it is always possible to construct a corresponding u, but it may

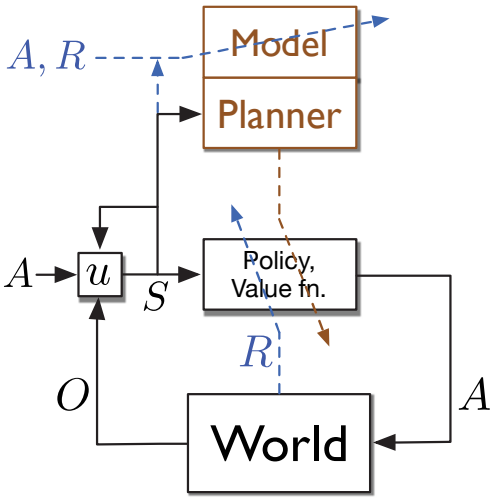


Figure 17.1: A conceptual agent architecture including a model, a planner, and a state-update function. The world in this case receives actions A and emits observations O. The observations and a copy of the action are used by the state-update function u to produce the new state. The new state is input to the policy and value function, producing the next action, and is also input to the planner (and to u). The information flows most responsible for learning are shown by dashed lines that pass diagonally across the boxes that they change. The reward R directly changes the policy and value function. The action, reward, and state change the model, which works closely with the planner to also change the policy and value function. Note that the operation of the planner can be decoupled from the agent–environment interaction, whereas the other processes should operate in lock step with this interaction to keep up with the arrival of new data. Also note that the model and planner do not deal with observations directly, but only with the states produced by u, which can act as targets for model learning.

not be computationally convenient and, as in the identity example, it may not produce a compact state. The state-update function is a central part of any agent architecture that handles partial observability. It must be efficiently computable, as no actions or predictions can be made until the state is available. An overall diagram of such an agent architecture is given in Figure 17.1.

A common strategy for finding a Markov state is to look for something compact that is recursively updatable and enables accurate short-term predictions. In fact, it is only necessary to make accurate *one-step* predictions. An important fact is that, if an f is incrementally updatable, then it is Markov if and only if all one-step tests can be accurately predicted, that is, if and only if

$$f(h) = f(h') \quad \Rightarrow \quad \Pr\{O_{t+1}{=}o|H_t{=}h, A_t{=}a\} = \Pr\{O_{t+1}{=}o|H_t{=}h', A_t{=}a\}, \quad (17.8)$$

for all $h, h' \in \{\mathcal{A} \times \mathcal{O}\}^*$, $o \in \mathcal{O}$ and $a \in \mathcal{A}$. Accurate one-step predictions are informationally sufficient, together with the state-update function, to accurately predict the probability of any test of any length. This can be done by iteratively and alternately making one-step predictions and applying the state-update function. From the whole tree of possibilities the exact probability of any test or the expectation of any GVF can be determined. These observations have led many researchers to focus on one-step predictions rather than directly on multi-step predictions such as GVFs. However, note

that determining long-term predictions from single-step predictions is exponentially complex in the length of the predictions. Moreover, one-step predictions can be iterated to give accurate long-term predictions only if they are exact. If there is any error or approximation in the one-step predictions, then it can compound to make the long-term predictions wildly inaccurate. In practice this is often what happens.

An example of obtaining Markov states through a state-update function is provided by the popular Bayesian approach known as *Partially Observable MDPs*, or *POMDPs*. In this approach the environment is assumed to have a well defined *latent state* X_t that underlies and produces the environment's observations, but is never available to the agent (and is not to be confused with the state S_t used by the agent to make predictions and decisions). The natural Markov state, S_t, for a POMDP is the *distribution* over the latent states given the history, called the *belief state*. For concreteness, assume the usual case in which there are a finite number of hidden states, $X_t \in \{1, 2, \ldots, d\}$. Then the belief state is the vector $S_t \doteq \mathbf{s}_t \in [0,1]^d$ with components

$$\mathbf{s}_t[i] \doteq \Pr\{X_t = i \mid H_t\}, \text{ for all possible latent states } i \in \{1, 2, \ldots, d\}. \tag{17.9}$$

The belief state remains the same size (same number of components) even as t grows. It can also be incrementally updated by Bayes' rule, assuming complete knowledge of the internal workings of the environment. Specifically, the ith component of the belief-state update function is

$$u(\mathbf{s}, a, o)[i] \doteq \frac{\sum_{x=1}^{d} \mathbf{s}[x] p(i, o|x, a)}{\sum_{x=1}^{d} \sum_{x'=1}^{d} \mathbf{s}[x] p(x', o|x, a)}, \quad \text{for all } a \in \mathcal{A}, o \in \mathcal{O}, \tag{17.10}$$

and for all belief states $\mathbf{s}$ with components $\mathbf{s}[x]$, where the four-argument p function here is not the usual one for MDPs (as in Chapter 3), but the analogous one for POMDPs, in terms of the *latent* state: $p(x', o|x, a) \doteq \Pr\{X_t = x', O_t = o \mid X_{t-1} = x, A_{t-1} = a\}$. This approach is popular in theoretical work and has many significant applications, but its assumptions and computational complexity scale poorly, and we do not recommend it as an approach to artificial intelligence.

Another example of Markov states is provided by *Predictive State Representations*, or *PSRs*. PSRs address the weakness of the POMDP approach that the semantics of its agent state S_t are grounded in the environment state, X_t, which is never observed and thus is difficult to learn about. In PSRs and related approaches, the semantics of the agent state is instead grounded in predictions about future observations and actions, which are readily observable. In PSRs, a Markov state is defined as a d-vector of the probabilities of d specially chosen "core" tests as defined above (17.5). The vector is then updated by a state-update function u that is analogous to Bayes rule, but with a semantics grounded in observable data, which arguably makes it easier to learn. This approach has been extended in many ways, including end-tests, compositional tests, powerful "spectral" methods, and closed-loop and temporally abstract tests learned by TD methods. Some of the best theoretical developments are for systems known as *Observable Operator Models* (OOMs) and Sequential Systems (Thon, 2017).

The fourth and final step in our brief outline of how to handle partial observability in reinforcement learning is to re-introduce approximation. As discussed in the introduction

to Part II, to approach artificial intelligence ambitiously we must embrace approximation. This is just as true for states as it is for value functions. We must accept and work with an approximate notion of state. The approximate state will play the same role in our algorithms as before, so we continue to use the notation S_t for the state used by the agent, even though it may not be Markov.

Perhaps the simplest example of an approximate state is just the latest observation, $S_t \doteq O_t$. Of course this approach cannot handle any hidden state information. It would be better to use the last k observations and actions, $S_t \doteq O_t, A_{t-1}, O_{t-1}, \ldots, A_{t-k}$, for some $k \geq 1$, which can be achieved by a state-update function that just shifts the new data in and the oldest data out. This *kth-order history* approach is still very simple, but can greatly increase the agent's capabilities compared to trying to use the single immediate observation directly as the state.

What happens when the Markov property (17.8) is only approximately satisfied? Unfortunately, long-term prediction performance can degrade dramatically when one-step predictions become even slightly inaccurate. Longer-term tests, GVFs, and state-update functions may or may not approximate better. The short-term and long-term approximation objectives are just different, and there are no useful theoretical guarantees at present.

Nevertheless, there are still reasons to think that the general idea outlined in this section applies to the approximate case. The general idea is that a state that is good for some predictions is also good for others—in particular, that a Markov state, sufficient for one-step predictions, is also sufficient for all others. If we step back from that specific result for the Markov case, the general idea is similar to what we discussed in Section 17.1 with multi-headed learning and auxiliary tasks. We discussed how representations that were good for the auxiliary tasks were often also good for the main task. Taken together, these suggest an approach to both partial observability and representation learning in which multiple predictions are pursued and used to direct the construction of state features. The guarantee provided by the perfect-but-impractical Markov property is replaced by the heuristic that what's good for some predictions may be good for others. This approach scales well with computational resources. With a powerful computer we could experiment with large numbers of predictions, perhaps favoring those that are most similar to the ones of ultimate interest, that are easiest to learn reliably, or that satisfy other criteria. It is important here to move beyond selecting the predictions manually. The agent should do it. This would require a general language for predictions, so that the agent can systematically explore a large space of possible predictions, sifting through them for the ones that are most useful.

In particular, both POMDP and PSR approaches can be applied with approximate states. The semantics of the state is often useful in forming the state-update function, as it is in these two approaches and in the kth-order history approach. However, there is not a strong need for the state to be accurate with respect to its semantics in order to retain useful information. Some approaches to state augmentation, such as Echo state networks (Jaeger, 2002), keep almost arbitrary information about the history and can nevertheless perform well. There are many possibilities, and we expect more work and ideas in this area. Learning the state-update function for an approximate state is a major part of the representation learning problem as it arises in reinforcement learning.

17.4 Designing Reward Signals

A major advantage of reinforcement learning over supervised learning is that reinforcement learning does not rely on detailed instructional information: generating a reward signal does not depend on knowledge of what the agent's correct actions should be. But the success of a reinforcement learning application strongly depends on how well the reward signal frames the goal of the application's designer and how well the signal assesses progress in reaching that goal. For these reasons, designing a reward signal is a critical part of any application of reinforcement learning.

By designing a reward signal we mean designing the part of an agent's environment that is responsible for computing each scalar reward R_t and sending it to the agent at each time t. In our discussion of terminology at the end of Chapter 14, we said that R_t is more like a signal generated inside an animal's brain than it is like an object or event in the animal's external environment. The parts of our brains that generate these signals for us evolved over millions of years to be well suited to the challenges our ancestors had to face in their struggles to propagate their genes to future generations. We should therefore not think that designing a good reward signal is always an easy thing to do!

One challenge is to design a reward signal so that as an agent learns, its behavior approaches, and ideally eventually achieves, what the application's designer actually desires. This can be easy if the designer's goal is simple and easy to identify, such as finding the solution to a well-defined problem or earning a high score in a well-defined game. In cases like these, it is usual to reward the agent according to its success in solving the problem or its success in improving its score. But some problems involve goals that are difficult to translate into reward signals. This is especially true when the problem requires the agent to skillfully perform a complex task or set of tasks, such as would be required of a useful household robotic assistant. Further, reinforcement learning agents can discover unexpected ways to make their environments deliver reward, some of which might be undesirable, or even dangerous. This is a longstanding and critical challenge for any method, like reinforcement learning, that is based on optimization. We discuss this issue more in Section 17.6, the final section of this book.

Even when there is a simple and easily identifiable goal, the problem of *sparse reward* often arises. Delivering non-zero reward frequently enough to allow the agent to achieve the goal once, let alone to learn to achieve it efficiently from multiple initial conditions, can be a daunting challenge. State–action pairs that clearly deserve to trigger reward may be few and far between, and rewards that mark progress toward a goal can be infrequent because progress is difficult or even impossible to detect. The agent may wander aimlessly for long periods of time (what Minsky, 1961, called the "plateau problem").

In practice, designing a reward signal is often left to an informal trial-and-error search for a signal that produces acceptable results. If the agent fails to learn, learns too slowly, or learns the wrong thing, then the designer tweaks the reward signal and tries again. To do this, the designer judges the agent's performance by criteria that he or she is attempting to translate into a reward signal so that the agent's goal matches his or her own. And if learning is too slow, the designer may try to design a non-sparse reward signal that effectively guides learning throughout the agent's interaction with its environment.

It is tempting to address the sparse reward problem by rewarding the agent for achieving subgoals that the designer thinks are important way stations to the overall goal. But augmenting the reward signal with well-intentioned supplemental rewards may lead the agent to behave differently from what is intended; the agent may end up not achieving the overall goal. A better way to provide such guidance is to leave the reward signal alone and instead augment the value-function approximation with an initial guess of what it should ultimately be, or augment it with initial guesses as to what certain parts of it should be. For example, suppose we wants to offer $v_0 : \mathcal{S} \to \mathbb{R}$ as an initial guess at the true optimal value function v_*, and that we are using linear function approximation with features $\mathbf{x} : \mathcal{S} \to \mathbb{R}^d$. Then we would define the initial value function approximation as

$$\hat{v}(s,\mathbf{w}) \doteq \mathbf{w}^\top \mathbf{x}(s) + v_0(s), \tag{17.11}$$

and update the weights $\mathbf{w}$ as usual. If the initial weight vector is $\mathbf{0}$, then the initial value function will be v_0, but the asymptotic solution quality will be determined by the feature vectors as usual. This initialization can also be done for arbitrary nonlinear approximators and arbitrary forms of v_0, though it is not guaranteed to always accelerate learning.

A particularly effective approach to the sparse reward problem is the *shaping* technique introduced by the psychologist B. F. Skinner and described in Section 14.3. The effectiveness of this technique relies on the fact that sparse reward problems are not just problems with the reward signal; they are also problems with an agent's policy in that it prevents the agent from frequently encountering rewarding states. Shaping involves changing the reward signal as learning proceeds, starting from a reward signal that is not sparse given the agent's initial behavior, and gradually modifying it toward a reward signal suited to the problem of original interest. Shaping might also involve modifying the dynamics of the task as learning proceeds. Each modification is made so that the agent is frequently rewarded given its current behavior. The agent faces a sequence of increasingly-difficult reinforcement learning problems, where what is learned at each stage makes the next-harder problem relatively easy because the agent now encounters reward more frequently than it would if it did not have prior experience with easier problems. This kind of shaping is an essential technique in training animals, and it is effective in computational reinforcement learning as well.

What if you have no idea what the rewards should be, but there is another agent, perhaps a person, who is already expert at the task and whose behavior can be observed? In this case you could use methods known variously as "imitation learning," "learning from demonstration," and "apprenticeship learning." The idea here is to benefit from the expert agent but leave open the possibility of eventually performing better. Learning from an expert's behavior can be done either by learning directly by supervised learning or by extracting a reward signal using what is known as "inverse reinforcement learning" and then using a reinforcement learning algorithm with that reward signal to learn a policy. The task of inverse reinforcement learning as explored by Ng and Russell (2000) is to try to recover the expert's reward signal from the expert's behavior alone. This cannot be done exactly because a policy can be optimal with respect to many different reward signals (for example, all policies are optimal with respect to a constant reward signal), but it is possible to find plausible reward signal candidates. Unfortunately, strong

assumptions are required, including knowledge of the environment's dynamics and of the feature vectors in which the reward signal is linear. The method also requires completely solving the problem (e.g., by dynamic programming methods) multiple times. These difficulties notwithstanding, Abbeel and Ng (2004) argue that the inverse reinforcement learning approach can sometimes be more effective than supervised learning for benefiting from the behavior of an expert.

Another approach to finding a good reward signal is to automate the trial-and-error search for a good signal that we mentioned above. From an application perspective, the reward signal is a parameter of the learning algorithm. As is true for other algorithm parameters, the search for a good reward signal can be automated by defining a space of feasible candidates and applying an optimization algorithm. The optimization algorithm evaluates each candidate reward signal by running the reinforcement learning system with that signal for some number of steps, and then scoring the overall result by a "high-level" objective function intended to encode the designer's true goal, ignoring the limitations of the agent. Reward signals can even be improved via online gradient ascent, where the gradient is that of the high-level objective function (Sorg, Lewis, and Singh, 2010). Relating this approach to the natural world, the algorithm for optimizing the high-level objective function is analogous to evolution, where the high-level objective function is an animal's evolutionary fitness determined by the number of its offspring that survive to reproductive age.

Computational experiments with this bilevel optimization approach—one level analogous to evolution, and the other due to reinforcement learning by individual agents—have confirmed that intuition alone is not always adequate to devise a good reward signal (Singh, Lewis, and Barto, 2009). The performance of a reinforcement learning agent as evaluated by the high-level objective function can be very sensitive to details of the agent's reward signal in subtle ways determined by the agent's limitations and the environment in which it acts and learns. These experiments have also demonstrated that an agent's goal should not always be the same as the goal of the agent's designer.

At first this seems counterintuitive, but it may be impossible for the agent to achieve the designer's goal no matter what its reward signal is. The agent has to learn under various kinds of constraints, such as limited computational power, limited access to information about its environment, or limited time to learn. When there are constraints like these, learning to achieve a goal that is different from the designer's goal can sometimes end up getting closer to the designer's goal than if that goal were pursued directly (Sorg, Singh, and Lewis, 2010; Sorg, 2011). Examples of this in the natural world are easy to find. Because we cannot directly assess the nutritional value of most foods, evolution—the designer of our reward signal—gave us a reward signal that makes us seek certain tastes. Though certainly not infallible (indeed, possibly detrimental in environments that differ in certain ways from ancestral environments), this compensates for many of our limitations: our limited sensory abilities, the limited time over which we can learn, and the risks involved in finding a healthy diet through personal experimentation. Similarly, because an animal cannot always observe its own evolutionary fitness, that objective function does not work as a reward signal for learning. Evolution instead provides reward signals that are sensitive to observable predictors of evolutionary fitness.

Finally, remember that a reinforcement learning agent is not necessarily like a complete

organism or robot; it can be a component of a larger behaving system. This means that reward signals may be influenced by things inside the larger behaving agent, such as motivational states, memories, ideas, or even hallucinations. Reward signals may also depend on properties of the learning process itself, such as measures of how much progress learning is making. Making reward signals sensitive to information about internal factors such as these makes it possible for an agent to learn how to control the "cognitive architecture" of which it is a part, as well as to acquire knowledge and skills that would be difficult to learn from a reward signal that depended only on external events. Possibilities like these led to the idea of "intrinsically-motivated reinforcement learning" that we briefly discuss further at the end of the following section.

17.5 Remaining Issues

In this book we have presented the foundations of a reinforcement learning approach to artificial intelligence. Roughly speaking, that approach is based on model-free and model-based methods working together, as in the Dyna architecture of Chapter 8, combined with function approximation as developed in Part II. The focus has been on online and incremental algorithms, which we see as fundamental even to model-based methods, and on how these can be applied in off-policy training situations. The full rationale for the latter has been presented only in this last chapter. That is, we have all along presented off-policy learning as an appealing way to deal with the explore/exploit dilemma, but only in this chapter have we discussed learning about many diverse auxiliary tasks simultaneously with GVFs and learning about the world hierarchically in terms of temporally-abstract option models, both of which involve off-policy learning. Much remains to be worked out, as we have indicated throughout the book and as evidenced by the directions for additional research discussed in this chapter. But suppose we are generous and grant the broad outlines of everything that we have done in the book *and* everything that has been outlined so far in this chapter. What would remain after that? Of course we can't know for sure what will be required, but we can make some guesses. In this section we highlight six further issues which it seems to us will still need to be addressed by future research.

First, we still need powerful parametric function approximation methods that work well in fully incremental and online settings. Methods based on deep learning and ANNs are a major step in this direction but, still, only work well with batch training on large data sets, with training from extensive off-line self play, or with learning from the interleaved experience of multiple agents on the same task. These and other settings are ways of working around a basic limitation of today's deep learning methods, which struggle to learn rapidly in the incremental, online settings that are most natural for the reinforcement learning algorithms emphasized in this book. The problem is sometimes described as one of "catastrophic interference" or "correlated data." When something new is learned it tends to replace what has previously been learned rather than adding to it, with the result that the benefit of the older learning is lost. Techniques such as "replay buffers" are often used to retain and replay old data so that its benefits are not permanently lost. An honest assessment has to be that current deep learning methods are not well suited to online learning. We see no reason that this limitation is insurmountable, but algorithms

that address it, while at the same time retaining the advantages of deep learning, have not yet been devised. Most current deep learning research is directed toward working around this limitation rather than removing it.

Second (and perhaps closely related), we still need methods for learning features such that subsequent learning generalizes well. This issue is an instance of a general problem variously called "representation learning," "constructive induction," and "meta-learning"—how can we use experience not just to learn a given desired function, but to learn inductive biases such that future learning generalizes better and is thus faster? This is an old problem, dating back to the origins of artificial intelligence and pattern recognition in the 1950s and 1960s.[1] Such age should give one pause. Perhaps there is no solution. But it is equally likely that the time for finding a solution and demonstrating its effectiveness has not yet arrived. Today machine learning is conducted at a far larger scale than it has been in the past, and the potential benefits of a good representation learning method have become much more apparent. We note that a new annual conference—the International Conference on Learning Representations—has been exploring this and related topics every year since 2013. It is also less common to explore representation learning within a reinforcement learning context. Reinforcement learning brings some new possibilities to this old issue, such as the auxiliary tasks discussed in Section 17.1. In reinforcement learning, the problem of representation learning can be identified with the problem of learning the state-update function discussed in Section 17.3.

Third, we still need scalable methods for planning with learned environment models. Planning methods have proven extremely effective in applications such as AlphaGo Zero and computer chess in which the model of the environment is known from the rules of the game or can otherwise be supplied by human designers. But cases of full model-based reinforcement learning, in which the environment model is learned from data and then used for planning, are rare. The Dyna system described in Chapter 8 is one example, but as described there and in most subsequent work it uses a tabular model without function approximation, which greatly limits its applicability. Only a few studies have included learned linear models, and even fewer have also explored the inclusion of temporally-abstract models using options as discussed in Section 17.2.

More work is needed before planning with learned models can be effective. For example, the learning of the model needs to be selective because the scope of a model strongly affects planning efficiency. If a model focuses on the key consequences of the most important options, then planning can be efficient and rapid, but if a model includes details of unimportant consequences of options that are unlikely to be selected, then planning may be almost useless. Environment models should be constructed judiciously with regard to both their states and dynamics with the goal of optimizing the planning process. The various parts of the model should be continually monitored as to the degree to which they contribute to, or detract from, planning efficiency. The field has not yet addressed this complex of issues or designed model-learning methods that take into account their implications.

[1] Some would claim that deep learning solves this problem, for example, that DQN as described in Section 16.5 illustrates a solution, but we are unconvinced. There is as yet little evidence that deep learning alone solves the representation learning problem in a general and efficient way.

A fourth issue that needs to be addressed in future research is that of automating the choice of tasks on which an agent works and uses to structure its developing competence. It is usual in machine learning for human designers to set the tasks that the learning agent is expected to master. Because these tasks are known in advance and remain fixed, they can be built into the learning algorithm code. However, looking ahead, we will want the agent to make its own choices about what tasks it should try to master. These might be subtasks of a specific overall task that is already known, or they might be intended to create building blocks that permit more efficient learning of many different tasks that the agent is likely to face in the future but which are currently unknown.

These tasks may be like the auxiliary tasks or the GVFs discussed in Section 17.1, or tasks solved by options as discussed in Section 17.2. In forming a GVF, for example, what should the cumulant, the policy, and the termination function be? The current state of the art is to select these manually, but far greater power and generality would come from making these task choices automatically, particularly when they derive from what the agent has previously constructed as a result of representation learning or experience with previous subproblems. If GVF design is automated, then the design choices themselves will have to be explicitly represented. Rather than the task choices being in the mind of the designer and built into the code, they will have to be in the machine itself in such a way that they can be set and changed, monitored, filtered, and searched among automatically. Tasks could then be built hierarchically upon others much like features are in an ANN. The tasks are the questions, and the contents of the ANN are the answers to those questions. We expect there will need to be a full hierarchy of questions to match the hierarchy of answers provided by modern deep learning methods.

The fifth issue that we would like to highlight for future research is that of the interaction between behavior and learning via some computational analog of *curiosity*. In this chapter we have been imagining a setting in which many tasks are being learned simultaneously, using off-policy methods, from the same stream of experience. The actions taken will of course influence this stream of experience, which in turn will determine how much learning occurs and which tasks are learned. When reward is not available, or not strongly influenced by behavior, the agent is free to choose actions that maximize in some sense the learning on the tasks, that is, to use some measure of learning progress as an internal or "intrinsic" reward, implementing a computational form of curiosity. In addition to measuring learning progress, intrinsic reward can, among other possibilities, signal the receipt of unexpected, novel, or otherwise interesting input, or can assess the agent's ability to cause changes in its environment. Intrinsic reward signals generated in these ways can be used by an agent to pose tasks for itself by defining auxiliary tasks, GVFs, or options, as discussed above, so that skills learned in this way can contribute to the agent's ability to master future tasks. The result is a computational analog of something like *play*. Many preliminary studies of such uses of intrinsic reward signals have been conducted, and exciting topics for future research remain in this general area.

A final issue that demands attention in future research is that of developing methods to make it acceptably safe to embed reinforcement learning agents into physical environments. This is one of the most pressing areas for future research, and we discuss it further in the following section.

17.6 Reinforcement Learning and the Future of Artificial Intelligence

When we were writing the first edition of this book in the mid-1990s, artificial intelligence was making significant progress and was having an impact on society, though it was mostly still the *promise* of artificial intelligence that was inspiring developments. Machine learning was part of that outlook, but it had not yet become indispensable to artificial intelligence. By today that promise has transitioned to applications that are changing the lives of millions of people, and machine learning has come into its own as a key technology. As we write this second edition, some of the most remarkable developments in artificial intelligence have involved reinforcement learning, most notably "deep reinforcement learning"—reinforcement learning with function approximation by deep artificial neural networks. We are at the beginning of a wave of real-world applications of artificial intelligence, many of which will include reinforcement learning, deep and otherwise, that will impact our lives in ways that are hard to predict.

But an abundance of successful real-world applications does not mean that true artificial intelligence has arrived. Despite great progress in many areas, the gulf between artificial intelligence and the intelligence of humans, and other animals, remains great. Superhuman performance can be achieved in some domains, even formidable domains like Go, but it remains a significant challenge to develop systems that are like us in being complete, interactive agents having general adaptability and problem-solving skills, emotional sophistication, creativity, and the ability to learn quickly from experience. With its focus on learning by interacting with dynamic environments, reinforcement learning, as it develops over the future, will be a critical component of agents with these abilities.

Reinforcement learning's connections to psychology and neuroscience (Chapters 14 and 15) underscore its relevance to another longstanding goal of artificial intelligence: shedding light on fundamental questions about the mind and how it emerges from the brain. Reinforcement learning theory is already contributing to our understanding of the brain's reward, motivation, and decision-making processes, and there is good reason to believe that through its links to computational psychiatry, reinforcement learning theory will contribute to methods for treating mental disorders, including drug abuse and addiction.

Another contribution that reinforcement learning can make over the future is as an aid to human decision making. Policies derived by reinforcement learning in simulated environments can advise human decision makers in such areas as education, healthcare, transportation, energy, and public-sector resource allocation. Particularly relevant is the key feature of reinforcement learning that it takes long-term consequences of decisions into account. This is very clear in games like backgammon and Go, where some of the most impressive results of reinforcement learning have been demonstrated, but it is also a property of many high-stakes decisions that affect our lives and our planet. Reinforcement learning follows related methods for advising human decision making that have been developed in the past by decision analysts in many disciplines. With advanced function approximation methods and massive computational power, reinforcement learning

methods have the potential to overcome some of the difficulties of scaling up traditional decision-support methods to larger and more complex problems.

The rapid pace of advances in artificial intelligence has led to warnings that artificial intelligence poses serious threats to our societies, even to humanity itself. The renowned scientist and artificial intelligence pioneer Herbert Simon anticipated the warnings we are hearing today in a presentation at the Earthware Symposium at CMU in 2000 (Simon, 2000). He spoke of the eternal conflict between the promise and perils of any new knowledge, reminding us of the Greek myths of Prometheus, the idealized hero of modern science, who stole fire from the gods for the benefit of mankind, and of Pandora, whose mythical box could be opened by a small and innocent action to release untold perils on the world. While accepting that this conflict is inevitable, Simon urged us to recognize that as designers of our future and not mere spectators, the decisions *we* make can tilt the scale in Prometheus' favor. This is certainly true for reinforcement learning, which can benefit society but can also produce undesirable outcomes if it is carelessly deployed. Thus, the *safety* of artificial intelligence applications involving reinforcement learning is a topic that deserves careful attention.

A reinforcement learning agent can learn by interacting with either the real world or with a simulation of some piece of the real world, or by a mixture of these two sources of experience. Simulators provide safe environments in which an agent can explore and learn without risking real damage to itself or to its environment. In most current applications, policies are learned from simulated experience instead of direct interaction with the real world. In addition to avoiding undesirable real-world consequences, learning from simulated experience can make virtually unlimited data available for learning, generally at less cost than needed to obtain real experience, and because simulations typically run much faster than real time, learning can often occur more quickly than if it relied on real experience.

Nevertheless, the full potential of reinforcement learning requires reinforcement learning agents to be embedded into the flow of real-world experience, where they act, explore, and learn in *our* world, and not just in *their* worlds. After all, reinforcement learning algorithms—at least those upon which we focus in this book—are designed to learn online, and they emulate many aspects of how animals are able to survive in nonstationary and hostile environments. Embedding reinforcement learning agents in the real world can be transformative in realizing the promises of artificial intelligence to amplify and extend human abilities.

A major reason for wanting a reinforcement learning agent to act and learn in the real world is that it is often difficult, sometimes impossible, to simulate real-world experience with enough fidelity to make the resulting policies, whether derived by reinforcement learning or by other methods, work well—and safely—when directing real actions. This is especially true for environments whose dynamics depend on the behavior of humans, such as in education, healthcare, transportation, and public policy—domains that can surely benefit from improved decision making. However, it is for real-world embedded agents that warnings about potential dangers of artificial intelligence need to be heeded.

Some of these warnings are particularly relevant to reinforcement learning. Because reinforcement learning is based on optimization, it inherits the plusses and minuses of all optimization methods. On the minus side is the problem of devising objective functions,

or reward signals in the case of reinforcement learning, so that optimization produces the desired results while avoiding undesirable results. We said in Section 17.4 that reinforcement learning agents can discover unexpected ways to make their environments deliver reward, some of which might be undesirable, or even dangerous. When we specify what we want a system to learn only indirectly, as we do in designing a reinforcement learning system's reward signal, we will not know how closely the agent will fulfill our desire until its learning is complete. This is hardly a new problem with reinforcement learning; recognition of it has a long history in both literature and engineering. For example, in Goethe's poem "The Sorcerer's Apprentice" (Goethe, 1878), the apprentice uses magic to enchant a broom to do his job of fetching water, but the result is an unintended flood due to the apprentice's inadequate knowledge of magic. In the engineering context, Norbert Wiener, the founder of cybernetics, warned of this problem more than half a century ago by relating the supernatural story of "The Monkey's Paw" (Wiener, 1964): "... it grants what you ask for, not what you should have asked for or what you intend" (p. 59). The problem has also been discussed at length in a modern context by Nick Bostrom (2014). Anyone having experience with reinforcement learning has likely seen their systems discover unexpected ways to obtain a lot of reward. Sometimes the unexpected behavior is good: it solves a problem in a nice new way. In other instances, what the agent learns violates considerations that the system designer may never have thought about. Careful design of reward signals is essential if an agent is to act in the real world with no opportunity for human vetting of its actions or means to easily interrupt its behavior.

Despite the possibility of unintended negative consequences, optimization has been used for hundreds of years by engineers, architects, and others whose designs have positively impacted the world. We owe much that is good in our environment to the application of optimization methods. Many approaches have been developed to mitigate the risk of optimization, such as adding hard and soft constraints, restricting optimization to robust and risk-sensitive policies, and optimizing with multiple objective functions. Some of these approaches have been adapted to reinforcement learning, and more research is needed to address these concerns. The problem of ensuring that a reinforcement learning agent's goal is attuned to our own remains a challenge.

Another challenge if reinforcement learning agents are to act and learn in the real world is not just about what they might learn *eventually*, but about how they will behave while they are learning. How do you make sure that an agent gets enough experience to learn a high-performing policy, all the while not harming its environment, other agents, or itself (or more realistically, while keeping the probability of harm acceptably low)? This problem is also not novel or unique to reinforcement learning. Risk management and mitigation for embedded reinforcement learning is similar to what control engineers have had to confront from the beginning of using automatic control in situations where a controller's behavior can have unacceptable, possibly catastrophic, consequences, as in the control of an aircraft or a delicate chemical process. Control applications rely on careful system modeling, model validation, and extensive testing, and there is a highly-developed body of theory aimed at ensuring convergence and stability of adaptive controllers designed for use when the dynamics of the system to be controlled are not fully known. Theoretical guarantees are never iron-clad because they depend on the validity of the assumptions underlying the mathematics, but without this theory, combined with risk-management

and mitigation practices, automatic control—adaptive and otherwise—would not be as beneficial as it is today in improving the quality, efficiency, and cost-effectiveness of processes on which we have come to rely. One of the most pressing areas for future reinforcement learning research is to adapt and extend methods developed in control engineering with the goal of making it acceptably safe to fully embed reinforcement learning agents into physical environments.

In closing, we return to Simon's call for us to recognize that we are designers of our future and not simply spectators. By decisions we make as individuals, and by the influence we can exert on how our societies are governed, we can work toward ensuring that the benefits made possible by a new technology outweigh the harm it can cause. There is ample opportunity to do this in the case of reinforcement learning, which can help improve the quality, fairness, and sustainability of life on our planet, but which can also release new perils. A threat already here is the displacement of jobs caused by applications of artificial intelligence. Still there are good reasons to believe that the benefits of artificial intelligence can outweigh the disruption it causes. As to safety, hazards possible with reinforcement learning are not completely different from those that have been managed successfully for related applications of optimization and control methods. As reinforcement learning moves out into the real world in future applications, developers have an obligation to follow best practices that have evolved for similar technologies, while at the same time extending them to make sure that Prometheus keeps the upper hand.

Bibliographical and Historical Remarks

17.1 General value functions were first explicitly identified by Sutton and colleagues (Sutton, 1995a; Sutton et al., 2011; Modayil, White, and Sutton, 2013). Ring (in preparation) developed an extensive thought experiment with GVFs ("forecasts") that has been influential despite not yet having been published.

The first demonstrations of multi-headed learning in reinforcement learning were by Jaderberg et al. (2017). Bellemare, Dabney, and Munos (2017) showed that predicting more things about the distribution of reward could significantly accelerate learning to optimize its expectation, an instance of auxiliary tasks. Many others have since taken up this line of research.

The general theory of classical conditioning as learned predictions together with built-in, reflexive reactions to the predictions has not to our knowledge been clearly articulated in the psychological literature. Modayil and Sutton (2014) describe it as an approach to the engineering of robots and other agents, calling it "Pavlovian control" to allude to its roots in classical conditioning.

17.2 The formalization of temporally abstract courses of action as options was introduced by Sutton, Precup, and Singh (1999), building on prior work by Parr (1998) and Sutton (1995a), and on classical work on Semi-MDPs (e.g., see Puterman, 1994). Precup's (2000) PhD thesis developed option ideas fully. An important limitation of these early works is that they did not treat the off-policy case

with function approximation. Intra-option learning in general requires off-policy learning, which could not be done reliably with function approximation at that time. Although now we have a variety of stable off-policy learning methods using function approximation, their combination with option ideas had not been significantly explored at the time of publication of this book. Barto and Mahadevan (2003) and Hengst (2012) review the options formalism and other approaches to temporal abstraction.

Using GVFs to implement option models has not previously been described. Our presentation uses the trick introduced by Modayil, White, and Sutton (2014) for predicting signals at the termination of policies.

Among the few works that have learned option models with function approximation are those by Sorg and Singh (2010), and by Bacon, Harb, and Precup (2017).

The extension of options and option models to the average-reward setting has not yet been developed in the literature.

17.3 A good presentation of the POMDP approach is given by Monahan (1982). PSRs and tests were introduced by Littman, Sutton, and Singh (2002). OOMs were introduced by Jaeger (1997, 1998, 2000). Sequential Systems, which unify PSRs, OOMs, and many other works, were introduced in the PhD thesis of Michael Thon (2017; Thon and Jaeger, 2015). Extensions to networks of temporal relationships were developed by Tanner (2006; Sutton and Tanner, 2005) and then extended to options (Sutton, Rafols, and Koop, 2006).

The theory of reinforcement learning with a non-Markov state representation was developed explicitly by Singh, Jaakkola, and Jordan (1994; Jaakkola, Singh, and Jordan, 1995). Early reinforcement learning approaches to partial observability were developed by Chrisman (1992), McCallum (1993, 1995), Parr and Russell (1995), Littman, Cassandra, and Kaelbling (1995), and by Lin and Mitchell (1992).

17.4 Early efforts to include advice and teaching in reinforcement learning include those by Lin (1992), Maclin and Shavlik (1994), Clouse (1996), and Clouse and Utgoff (1992).

Skinner's shaping should not be confused with the "potential-based shaping" technique introduced by Ng, Harada, and Russell (1999). Their technique has been shown by Wiewiora (2003) to be equivalent to the simpler idea of providing an initial approximation to the value function, as in (17.11).

17.5 We recommend the book by Goodfellow, Bengio, and Courville (2016) for discussion of today's deep learning techniques. The problem of catastrophic interference in ANNs was developed by McCloskey and Cohen (1989), Ratcliff (1990), and French (1999). The idea of a replay buffer was introduced by Lin (1992) and used prominently in deep learning in the Atari game playing system (Section 16.5, Mnih et al., 2013, 2015).

Minsky (1961) was one of the first to identify the problem of representation learning.

Among the few works to consider planning with learned, approximate models are those by Kuvayev and Sutton (1996), Sutton, Szepesvari, Geramifard, and Bowling (2008), Nouri and Littman (2009), and Hester and Stone (2012).

The need to be selective in model construction to avoid slowing planning is well known in artificial intelligence. Some of the classic work is by Minton (1990) and Tambe, Newell, and Rosenbloom (1990). Hauskrecht, Meuleau, Kaelbling, Dean, and Boutilier (1998) showed this effect in MDPs with deterministic options.

Schmidhuber (1991a, b) proposed how something like curiosity would result if reward signals were a function of how quickly an agent's environment model is improving. The empowerment function proposed by Klyubin, Polani, and Nehaniv (2005) is an information-theoretic measure of an agent's ability to control its environment that can function as an intrinsic reward signal. Baldassarre and Mirolli (2013) is a collection of contributions by researchers studying intrinsic reward and motivation from both biological and computational perspectives, including a perspective on "intrinsically-motivated reinforcement learning," to use the term introduced by Singh, Barto, and Chentenez (2004). See also Oudeyer and Kaplan (2007), Oudeyer, Kaplan, and Hafner (2007), and Barto (2013).

References

Abbeel, P., Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the 21st International Conference on Machine Learning*. ACM, New York.

Abramson, B. (1990). Expected-outcome: A general model of static evaluation. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 12*(2):182–193.

Adams, C. D. (1982). Variations in the sensitivity of instrumental responding to reinforcer devaluation. *The Quarterly Journal of Experimental Psychology, 34*(2):77–98.

Adams, C. D., Dickinson, A. (1981). Instrumental responding following reinforcer devaluation. *The Quarterly Journal of Experimental Psychology, 33*(2):109–121.

Adams, R. A., Huys, Q. J. M., Roiser, J. P. (2015). Computational Psychiatry: towards a mathematically informed understanding of mental illness. *Journal of Neurology, Neurosurgery & Psychiatry*. doi:10.1136/jnnp-2015-310737

Agrawal, R. (1995). Sample mean based index policies with $O(logn)$ regret for the multi-armed bandit problem. *Advances in Applied Probability, 27*(4):1054–1078.

Agre, P. E. (1988). *The Dynamic Structure of Everyday Life*. PhD thesis, Massachusetts Institute of Technology, Cambridge MA. AI-TR 1085, MIT Artificial Intelligence Laboratory.

Agre, P. E., Chapman, D. (1990). What are plans for? *Robotics and Autonomous Systems, 6*(1-2):17–34.

Aizerman, M. A., Braverman, E. Í., Rozonoer, L. I. (1964). Probability problem of pattern recognition learning and potential functions method. *Avtomat. i Telemekh, 25*(9):1307–1323.

Albus, J. S. (1971). A theory of cerebellar function. *Mathematical Biosciences, 10*(1-2):25–61.

Albus, J. S. (1981). *Brain, Behavior, and Robotics*. Byte Books, Peterborough, NH.

Aleksandrov, V. M., Sysoev, V. I., Shemeneva, V. V. (1968). Stochastic optimization of systems. *Izv. Akad. Nauk SSSR, Tekh. Kibernetika*:14–19.

Amari, S. I. (1998). Natural gradient works efficiently in learning. *Neural Computation, 10*(2):251–276.

An, P. C. E. (1991). *An Improved Multi-dimensional CMAC Neural network: Receptive Field Function and Placement*. PhD thesis, University of New Hampshire, Durham.

An, P. C. E., Miller, W. T., Parks, P. C. (1991). Design improvements in associative memories for cerebellar model articulation controllers (CMAC). *Artificial Neural Networks*, pp. 1207–1210, Elsevier North-Holland. http://www.incompleteideas.net/papers/AnMillerParks1991.pdf

Anderson, C. W. (1986). *Learning and Problem Solving with Multilayer Connectionist Systems*. PhD thesis, University of Massachusetts, Amherst.

Anderson, C. W. (1987). Strategy learning with multilayer connectionist representations. In *Proceedings of the 4th International Workshop on Machine Learning*, pp. 103–114. Morgan Kaufmann.

Anderson, C. W. (1989). Learning to control an inverted pendulum using neural networks. *IEEE Control Systems Magazine, 9*(3):31–37.

Anderson, J. A., Silverstein, J. W., Ritz, S. A., Jones, R. S. (1977). Distinctive features, categorical perception, and probability learning: Some applications of a neural model. *Psychological Review, 84*(5):413–451.

Andreae, J. H. (1963). STELLA, A scheme for a learning machine. In *Proceedings of the 2nd IFAC Congress, Basle*, pp. 497–502. Butterworths, London.

Andreae, J. H. (1969). Learning machines—a unified view. In A. R. Meetham and R. A. Hudson (Eds.), *Encyclopedia of Information, Linguistics, and Control*, pp. 261–270. Pergamon, Oxford.

Andreae, J. H. (1977). *Thinking with the Teachable Machine.* Academic Press, London.

Andreae, J. H. (2017a). A model of how the brain learns: A short introduction to multiple context associative learning (MCAL) and the PP system. Unpublished report.

Andreae, J. H. (2017b). Working memory for the associative learning of language. Unpublished report.

Andreae, J. H., Cashin, P. M. (1969). A learning machine with monologue. *International Journal of Man–Machine Studies, 1*(1):1–20.

Arthur, W. B. (1991). Designing economic agents that act like human agents: A behavioral approach to bounded rationality. *The American Economic Review, 81*(2):353–359.

Asadi, K., Allen, C., Roderick, M., Mohamed, A. R., Konidaris, G., Littman, M. (2017). Mean actor critic. ArXiv:1709.00503.

Atkeson, C. G. (1992). Memory-based approaches to approximating continuous functions. In *Sante Fe Institute Studies in the Sciences of Complexity*, Proceedings Vol. 12, pp. 521–521. Addison-Wesley.

Atkeson, C. G., Moore, A. W., Schaal, S. (1997). Locally weighted learning. *Artificial Intelligence Review, 11*:11–73.

Auer, P., Cesa-Bianchi, N., Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine learning, 47*(2-3):235–256.

Bacon, P. L., Harb, J., Precup, D. (2017). The option-critic architecture. In *Proceedings of the Association for the Advancement of Artificial Intelligence*, pp. 1726–1734.

Baird, L. C. (1995). Residual algorithms: Reinforcement learning with function approximation. In *Proceedings of the 12th International Conference on Machine Learning*, pp. 30–37. Morgan Kaufmann.

Baird, L. C. (1999). *Reinforcement Learning through Gradient Descent.* PhD thesis, Carnegie Mellon University, Pittsburgh PA.

Baird, L. C., Klopf, A. H. (1993). Reinforcement learning with high-dimensional, continuous actions. Wright Laboratory, Wright-Patterson Air Force Base, Tech. Rep. WL-TR-93-1147.

Baird, L., Moore, A. W. (1999). Gradient descent for general reinforcement learning. In *Advances in Neural Information Processing Systems 11*, pp. 968–974. MIT Press, Cambridge MA.

Baldassarre, G., Mirolli, M. (Eds.) (2013). *Intrinsically Motivated Learning in Natural and Artificial Systems.* Springer-Verlag, Berlin Heidelberg.

Balke, A., Pearl, J. (1994). Counterfactual probabilities: Computational methods, bounds and applications. In *Proceedings of the Tenth International Conference on Uncertainty in Artificial Intelligence*, pp. 46–54. Morgan Kaufmann.

Baras, D., Meir, R. (2007). Reinforcement learning, spike-time-dependent plasticity, and the BCM rule. *Neural Computation, 19*(8):2245–2279.

Barnard, E. (1993). Temporal-difference methods and Markov models. *IEEE Transactions on Systems, Man, and Cybernetics, 23*(2):357–365.

Barreto, A. S., Precup, D., Pineau, J. (2011). Reinforcement learning using kernel-based stochastic factorization. In *Advances in Neural Information Processing Systems 24*, pp. 720–728. Curran Associates, Inc.

Bartlett, P. L., Baxter, J. (1999). Hebbian synaptic modifications in spiking neurons that learn. Technical report, Research School of Information Sciences and Engineering, Australian National University.

Bartlett, P. L., Baxter, J. (2000). A biologically plausible and locally optimal learning algorithm for spiking neurons. Rapport technique, Australian National University.

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

Barto, A. G. (1986). Game-theoretic cooperativity in networks of self-interested units. In J. S. Denker (Ed.), *Neural Networks for Computing*, pp. 41–46. American Institute of Physics, New York.

Barto, A. G. (1989). From chemotaxis to cooperativity: Abstract exercises in neuronal learning strategies. In R. Durbin, R. Maill and G. Mitchison (Eds.), *The Computing Neuron*, pp. 73–98. Addison-Wesley, Reading, MA.

Barto, A. G. (1990). Connectionist learning for control: An overview. In T. Miller, R. S. Sutton, and P. J. Werbos (Eds.), *Neural Networks for Control*, pp. 5–58. MIT Press, Cambridge, MA.

Barto, A. G. (1991). Some learning tasks from a control perspective. In L. Nadel and D. L. Stein (Eds.), *1990 Lectures in Complex Systems*, pp. 195–223. Addison-Wesley, Redwood City, CA.

Barto, A. G. (1992). Reinforcement learning and adaptive critic methods. In D. A. White and D. A. Sofge (Eds.), *Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches*, pp. 469–491. Van Nostrand Reinhold, New York.

Barto, A. G. (1995a). Adaptive critics and the basal ganglia. In J. C. Houk, J. L. Davis, and D. G. Beiser (Eds.), *Models of Information Processing in the Basal Ganglia*, pp. 215–232. MIT Press, Cambridge, MA.

Barto, A. G. (1995b). Reinforcement learning. In M. A. Arbib (Ed.), *Handbook of Brain Theory and Neural Networks*, pp. 804–809. MIT Press, Cambridge, MA.

Barto, A. G. (2011). Adaptive real-time dynamic programming. In C. Sammut and G. I Webb (Eds.), *Encyclopedia of Machine Learning*, pp. 19–22. Springer Science and Business Media.

Barto, A. G. (2013). Intrinsic motivation and reinforcement learning. In G. Baldassarre and M. Mirolli (Eds.), *Intrinsically Motivated Learning in Natural and Artificial Systems*, pp. 17–47. Springer-Verlag, Berlin Heidelberg.

Barto, A. G., Anandan, P. (1985). Pattern recognizing stochastic learning automata. *IEEE Transactions on Systems, Man, and Cybernetics, 15*(3):360–375.

Barto, A. G., Anderson, C. W. (1985). Structural learning in connectionist systems. In *Proceedings of the Seventh Annual Conference of the Cognitive Science Society*, pp. 43–54.

Barto, A. G., Anderson, C. W., Sutton, R. S. (1982). Synthesis of nonlinear control surfaces by a layered associative search network. *Biological Cybernetics, 43*(3):175–185.

Barto, A. G., Bradtke, S. J., Singh, S. P. (1991). Real-time learning and control using asynchronous dynamic programming. Technical Report 91-57. Department of Computer and Information Science, University of Massachusetts, Amherst.

Barto, A. G., Bradtke, S. J., Singh, S. P. (1995). Learning to act using real-time dynamic programming. *Artificial Intelligence, 72*(1-2):81–138.

Barto, A. G., Duff, M. (1994). Monte Carlo matrix inversion and reinforcement learning. In *Advances in Neural Information Processing Systems 6*, pp. 687–694. Morgan Kaufmann, San Francisco.

Barto, A. G., Jordan, M. I. (1987). Gradient following without back-propagation in layered networks. In M. Caudill and C. Butler (Eds.), *Proceedings of the IEEE First Annual Conference on Neural Networks*, pp. II629–II636. SOS Printing, San Diego.

Barto, A. G., Mahadevan, S. (2003). Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems, 13*(4):341–379.

Barto, A. G., Singh, S. P. (1990). On the computational economics of reinforcement learning. In *Connectionist Models: Proceedings of the 1990 Summer School*. Morgan Kaufmann.

Barto, A. G., Sutton, R. S. (1981a). Goal seeking components for adaptive intelligence: An initial assessment. Technical Report AFWAL-TR-81-1070. Air Force Wright Aeronautical Laboratories/Avionics Laboratory, Wright-Patterson AFB, OH.

Barto, A. G., Sutton, R. S. (1981b). Landmark learning: An illustration of associative search. *Biological Cybernetics, 42*(1):1–8.

Barto, A. G., Sutton, R. S. (1982). Simulation of anticipatory responses in classical conditioning by a neuron-like adaptive element. *Behavioural Brain Research, 4*(3):221–235.

Barto, A. G., Sutton, R. S., Anderson, C. W. (1983). Neuronlike elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics, 13*(5):835–846. Reprinted in J. A. Anderson and E. Rosenfeld (Eds.), *Neurocomputing: Foundations of Research*, pp. 535–549. MIT Press, Cambridge, MA, 1988.

Barto, A. G., Sutton, R. S., Brouwer, P. S. (1981). Associative search network: A reinforcement learning associative memory. *Biological Cybernetics, 40*(3):201–211.

Barto, A. G., Sutton, R. S., Watkins, C. J. C. H. (1990). Learning and sequential decision making. In M. Gabriel and J. Moore (Eds.), *Learning and Computational Neuroscience: Foundations of Adaptive Networks*, pp. 539–602. MIT Press, Cambridge, MA.

Baxter, J., Bartlett, P. L. (2001). Infinite-horizon policy-gradient estimation. *Journal of Artificial Intelligence Research, 15*:319–350.

Baxter, J., Bartlett, P. L., Weaver, L. (2001). Experiments with infinite-horizon, policy-gradient estimation. *Journal of Artificial Intelligence Research, 15*:351–381.

Bellemare, M. G., Dabney, W., Munos, R. (2017). A distributional perspective on reinforcement learning. ArXiv:1707.06887.

Bellemare, M. G., Naddaf, Y., Veness, J., Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research, 47*:253–279.

Bellemare, M. G., Veness, J., Bowling, M. (2012). Investigating contingency awareness using Atari 2600 games. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, pp. 864–871. AAAI Press, Menlo Park, CA.

Bellman, R. E. (1956). A problem in the sequential design of experiments. *Sankhya, 16*:221–229.

Bellman, R. E. (1957a). *Dynamic Programming*. Princeton University Press, Princeton.

Bellman, R. E. (1957b). A Markov decision process. *Journal of Mathematics and Mechanics, 6*(5):679–684.

Bellman, R. E., Dreyfus, S. E. (1959). Functional approximations and dynamic programming. *Mathematical Tables and Other Aids to Computation, 13*:247–251.

Bellman, R. E., Kalaba, R., Kotkin, B. (1963). Polynomial approximation—A new computational technique in dynamic programming: Allocation processes. *Mathematical Computation, 17*:155–161.

Bengio, Y. (2009). Learning deep architectures for AI. *Foundations and Trends in Machine Learning, 2*(1):1–27.

Bengio, Y., Courville, A. C., Vincent, P. (2012). Unsupervised feature learning and deep learning: A review and new perspectives. *CoRR 1*, ArXiv:1206.5538.

Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. *Communications of the ACM, 18*(9):509–517.

Berg, H. C. (1975). Chemotaxis in bacteria. *Annual review of biophysics and bioengineering, 4*(1):119–136.

Berns, G. S., McClure, S. M., Pagnoni, G., Montague, P. R. (2001). Predictability modulates human brain response to reward. *The journal of neuroscience, 21*(8):2793–2798.

Berridge, K. C., Kringelbach, M. L. (2008). Affective neuroscience of pleasure: reward in humans and animals. *Psychopharmacology, 199*(3):457–480.

Berridge, K. C., Robinson, T. E. (1998). What is the role of dopamine in reward: hedonic impact, reward learning, or incentive salience? *Brain Research Reviews, 28*(3):309–369.

Berry, D. A., Fristedt, B. (1985). *Bandit Problems.* Chapman and Hall, London.

Bertsekas, D. P. (1982). Distributed dynamic programming. *IEEE Transactions on Automatic Control, 27*(3):610–616.

Bertsekas, D. P. (1983). Distributed asynchronous computation of fixed points. *Mathematical Programming, 27*(1):107–120.

Bertsekas, D. P. (1987). *Dynamic Programming: Deterministic and Stochastic Models.* Prentice-Hall, Englewood Cliffs, NJ.

Bertsekas, D. P. (2005). *Dynamic Programming and Optimal Control, Volume 1,* third edition. Athena Scientific, Belmont, MA.

Bertsekas, D. P. (2012). *Dynamic Programming and Optimal Control, Volume 2: Approximate Dynamic Programming*, fourth edition. Athena Scientific, Belmont, MA.

Bertsekas, D. P. (2013). Rollout algorithms for discrete optimization: A survey. In *Handbook of Combinatorial Optimization*, pp. 2989–3013. Springer, New York.

Bertsekas, D. P., Tsitsiklis, J. N. (1989). *Parallel and Distributed Computation: Numerical Methods.* Prentice-Hall, Englewood Cliffs, NJ.

Bertsekas, D. P., Tsitsiklis, J. N. (1996). *Neuro-Dynamic Programming.* Athena Scientific, Belmont, MA.

Bertsekas, D. P., Tsitsiklis, J. N., Wu, C. (1997). Rollout algorithms for combinatorial optimization. *Journal of Heuristics, 3*(3):245–262.

Bertsekas, D. P., Yu, H. (2009). Projected equation methods for approximate solution of large linear systems. *Journal of Computational and Applied Mathematics, 227*(1):27–50.

Bhat, N., Farias, V., Moallemi, C. C. (2012). Non-parametric approximate dynamic programming via the kernel method. In *Advances in Neural Information Processing Systems 25*, pp. 386–394. Curran Associates, Inc.

Bhatnagar, S., Sutton, R., Ghavamzadeh, M., Lee, M. (2009). Natural actor–critic algorithms. *Automatica, 45*(11).

Biermann, A. W., Fairfield, J. R. C., Beres, T. R. (1982). Signature table systems and learning. *IEEE Transactions on Systems, Man, and Cybernetics, 12*(5):635–648.

Bishop, C. M. (1995). *Neural Networks for Pattern Recognition.* Clarendon, Oxford.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning.* Springer Science + Business Media New York LLC.

Blodgett, H. C. (1929). The effect of the introduction of reward upon the maze performance of rats. *University of California Publications in Psychology, 4*:113–134.

Boakes, R. A., Costa, D. S. J. (2014). Temporal contiguity in associative learning: Iinterference and decay from an historical perspective. *Journal of Experimental Psychology: Animal Learning and Cognition, 40*(4):381–400.

Booker, L. B. (1982). *Intelligent Behavior as an Adaptation to the Task Environment*. PhD thesis, University of Michigan, Ann Arbor.

Bostrom, N. (2014). *Superintelligence: Paths, Dangers, Strategies*. Oxford University Press.

Bottou, L., Vapnik, V. (1992). Local learning algorithms. *Neural Computation, 4*(6):888–900.

Boyan, J. A. (1999). Least-squares temporal difference learning. In *Proceedings of the 16th International Conference on Machine Learning*, pp. 49–56.

Boyan, J. A. (2002). Technical update: Least-squares temporal difference learning. *Machine Learning, 49*(2):233–246.

Boyan, J. A., Moore, A. W. (1995). Generalization in reinforcement learning: Safely approximating the value function. In *Advances in Neural Information Processing Systems 7*, pp. 369–376. MIT Press, Cambridge, MA.

Bradtke, S. J. (1993). Reinforcement learning applied to linear quadratic regulation. In *Advances in Neural Information Processing Systems 5*, pp. 295–302. Morgan Kaufmann.

Bradtke, S. J. (1994). *Incremental Dynamic Programming for On-Line Adaptive Optimal Control*. PhD thesis, University of Massachusetts, Amherst. Appeared as CMPSCI Technical Report 94-62.

Bradtke, S. J., Barto, A. G. (1996). Linear least–squares algorithms for temporal difference learning. *Machine Learning, 22*:33–57.

Bradtke, S. J., Ydstie, B. E., Barto, A. G. (1994). Adaptive linear quadratic control using policy iteration. In *Proceedings of the American Control Conference*, pp. 3475–3479. American Automatic Control Council, Evanston, IL.

Brafman, R. I., Tennenholtz, M. (2003). R-max – a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research, 3*:213–231.

Breiman, L. (2001). Random forests. *Machine Learning, 45*(1):5–32.

Breiter, H. C., Aharon, I., Kahneman, D., Dale, A., Shizgal, P. (2001). Functional imaging of neural responses to expectancy and experience of monetary gains and losses. *Neuron, 30*(2):619–639.

Breland, K., Breland, M. (1961). The misbehavior of organisms. *American Psychologist, 16*(11):681–684.

Bridle, J. S. (1990). Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimates of parameters. In *Advances in Neural Information Processing Systems 2*, pp. 211–217. Morgan Kaufmann, San Mateo, CA.

Broomhead, D. S., Lowe, D. (1988). Multivariable functional interpolation and adaptive networks. *Complex Systems, 2*:321–355.

Bromberg-Martin, E. S., Matsumoto, M., Hong, S., Hikosaka, O. (2010). A pallidus-habenula-dopamine pathway signals inferred stimulus values. *Journal of Neurophysiology, 104*(2):1068–1076.

Browne, C.B., Powley, E., Whitehouse, D., Lucas, S.M., Cowling, P.I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., Colton, S. (2012). A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games, 4*(1):1–43.

Brown, J., Bullock, D., Grossberg, S. (1999). How the basal ganglia use parallel excitatory and inhibitory learning pathways to selectively respond to unexpected rewarding cues. *The Journal of Neuroscience, 19*(23):10502–10511.

Bryson, A. E., Jr. (1996). Optimal control—1950 to 1985. *IEEE Control Systems, 13*(3):26–33.

Buchanan, B. G., Mitchell, T., Smith, R. G., Johnson, C. R., Jr. (1978). Models of learning systems. *Encyclopedia of Computer Science and technology, 11*.

Buhusi, C. V., Schmajuk, N. A. (1999). Timing in simple conditioning and occasion setting: A neural network approach. *Behavioural Processes, 45*(1):33–57.

Buşoniu, L., Lazaric, A., Ghavamzadeh, M., Munos, R., Babuška, R., De Schutter, B. (2012). Least-squares methods for policy iteration. In M. Wiering and M. van Otterlo (Eds.), *Reinforcement Learning: State-of-the-Art*, pp. 75–109. Springer-Verlag Berlin Heidelberg.

Bush, R. R., Mosteller, F. (1955). *Stochastic Models for Learning*. Wiley, New York.

Byrne, J. H., Gingrich, K. J., Baxter, D. A. (1990). Computational capabilities of single neurons: Relationship to simple forms of associative and nonassociative learning in *aplysia*. In R. D. Hawkins and G. H. Bower (Eds.), *Computational Models of Learning*, pp. 31–63. Academic Press, New York.

Calabresi, P., Picconi, B., Tozzi, A., Filippo, M. D. (2007). Dopamine-mediated regulation of corticostriatal synaptic plasticity. *Trends in Neuroscience, 30*(5):211–219.

Camerer, C. (2011). *Behavioral Game Theory: Experiments in Strategic Interaction*. Princeton University Press.

Campbell, D. T. (1960). Blind variation and selective survival as a general strategy in knowledge-processes. In M. C. Yovits and S. Cameron (Eds.), *Self-Organizing Systems*, pp. 205–231. Pergamon, New York.

Cao, X. R. (2009). Stochastic learning and optimization—A sensitivity-based approach. *Annual Reviews in Control, 33*(1):11–24.

Cao, X. R., Chen, H. F. (1997). Perturbation realization, potentials, and sensitivity analysis of Markov processes. *IEEE Transactions on Automatic Control, 42*(10):1382–1393.

Carlström, J., Nordström, E. (1997). Control of self-similar ATM call traffic by reinforcement learning. In *Proceedings of the International Workshop on Applications of Neural Networks to Telecommunications 3*, pp. 54–62. Erlbaum, Hillsdale, NJ.

Chapman, D., Kaelbling, L. P. (1991). Input generalization in delayed reinforcement learning: An algorithm and performance comparisons. In *Proceedings of the Twelfth International Conference on Artificial Intelligence*, pp. 726–731. Morgan Kaufmann, San Mateo, CA.

Chaslot, G., Bakkes, S., Szita, I., Spronck, P. (2008). Monte-Carlo tree search: A new framework for game AI. In *Proceedings of the Fourth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIDE-08)*, pp. 216–217. AAAI Press, Menlo Park, CA.

Chow, C.-S., Tsitsiklis, J. N. (1991). An optimal one-way multigrid algorithm for discrete-time stochastic control. *IEEE Transactions on Automatic Control, 36*(8):898–914.

Chrisman, L. (1992). Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pp. 183–188. AAAI/MIT Press, Menlo Park, CA.

Christensen, J., Korf, R. E. (1986). A unified theory of heuristic evaluation functions and its application to learning. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pp. 148–152. Morgan Kaufmann.

Cichosz, P. (1995). Truncating temporal differences: On the efficient implementation of TD(λ) for reinforcement learning. *Journal of Artificial Intelligence Research, 2*:287–318.

Ciosek, K., Whiteson, S. (2017). Expected policy gradients. ArXiv:1706.05374v1. A revised version appeared in *Proceedings of the Annual Conference of the Association for the Advancement of Artificial Intelligence*, pp. 2868–2875.

Ciosek, K., Whiteson, S. (2018). Expected policy gradients for reinforcement learning. ArXiv: 1801.03326.

Claridge-Chang, A., Roorda, R. D., Vrontou, E., Sjulson, L., Li, H., Hirsh, J., Miesenböck, G. (2009). Writing memories with light-addressable reinforcement circuitry. *Cell, 139*(2):405–415.

Clark, R. E., Squire, L. R. (1998). Classical conditioning and brain systems: the role of awareness. *Science, 280*(5360):77–81.

Clark, W. A., Farley, B. G. (1955). Generalization of pattern recognition in a self-organizing system. In *Proceedings of the 1955 Western Joint Computer Conference*, pp. 86–91.

Clouse, J. (1996). *On Integrating Apprentice Learning and Reinforcement Learning TITLE2.* PhD thesis, University of Massachusetts, Amherst. Appeared as CMPSCI Technical Report 96-026.

Clouse, J., Utgoff, P. (1992). A teaching method for reinforcement learning systems. In *Proceedings of the 9th International Workshop on Machine Learning*, pp. 92–101. Morgan Kaufmann.

Cobo, L. C., Zang, P., Isbell, C. L., Thomaz, A. L. (2011). Automatic state abstraction from demonstration. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*, pp. 1243-1248. AAAI Press.

Connell, J. (1989). A colony architecture for an artificial creature. Technical Report AI-TR-1151. MIT Artificial Intelligence Laboratory, Cambridge, MA.

Connell, M. E., Utgoff, P. E. (1987). Learning to control a dynamic physical system. *Computational intelligence, 3*(1):330–337.

Contreras-Vidal, J. L., Schultz, W. (1999). A predictive reinforcement model of dopamine neurons for learning approach behavior. *Journal of Computational Neuroscience, 6*(3):191–214.

Coulom, R. (2006). Efficient selectivity and backup operators in Monte-Carlo tree search. In *Proceedings of the 5th International Conference on Computers and Games (CG'06)*, pp. 72–83. Springer-Verlag Berlin, Heidelberg.

Courville, A. C., Daw, N. D., Touretzky, D. S. (2006). Bayesian theories of conditioning in a changing world. *Trends in Cognitive Science, 10*(7):294–300.

Craik, K. J. W. (1943). *The Nature of Explanation.* Cambridge University Press, Cambridge.

Cross, J. G. (1973). A stochastic learning model of economic behavior. *The Quarterly Journal of Economics, 87*(2):239–266.

Crow, T. J. (1968). Cortical synapses and reinforcement: a hypothesis. *Nature, 219*(5155):736–737.

Curtiss, J. H. (1954). A theoretical comparison of the efficiencies of two classical methods and a Monte Carlo method for computing one component of the solution of a set of linear algebraic equations. In H. A. Meyer (Ed.), *Symposium on Monte Carlo Methods*, pp. 191–233. Wiley, New York.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems, 2*(4):303–314.

Cziko, G. (1995). *Without Miracles: Universal Selection Theory and the Second Darvinian Revolution.* MIT Press, Cambridge, MA.

Dabney, W. (2014). *Adaptive step-sizes for reinforcement learning.* PhD thesis, University of Massachusetts, Amherst.

Dabney, W., Barto, A. G. (2012). Adaptive step-size for online temporal difference learning. In *Proceedings of the Annual Conference of the Association for the Advancement of Artificial Intelligence.*

Daniel, J. W. (1976). Splines and efficiency in dynamic programming. *Journal of Mathematical Analysis and Applications, 54*:402–407.

Dann, C., Neumann, G., Peters, J. (2014). Policy evaluation with temporal differences: A survey and comparison. *Journal of Machine Learning Research, 15*:809–883.

Daw, N. D., Courville, A. C., Touretzky, D. S. (2003). Timing and partial observability in the dopamine system. In *Advances in Neural Information Processing Systems 15*, pp. 99–106. MIT Press, Cambridge, MA.

Daw, N. D., Courville, A. C., Touretzky, D. S. (2006). Representation and timing in theories of the dopamine system. *Neural Computation, 18*(7):1637–1677.

Daw, N. D., Niv, Y., Dayan, P. (2005). Uncertainty based competition between prefrontal and dorsolateral striatal systems for behavioral control. *Nature Neuroscience, 8*(12):1704–1711.

Daw, N. D., Shohamy, D. (2008). The cognitive neuroscience of motivation and learning. *Social Cognition, 26*(5):593–620.

Dayan, P. (1991). Reinforcement comparison. In D. S. Touretzky, J. L. Elman, T. J. Sejnowski, and G. E. Hinton (Eds.), *Connectionist Models: Proceedings of the 1990 Summer School*, pp. 45–51. Morgan Kaufmann.

Dayan, P. (1992). The convergence of TD(λ) for general λ. *Machine Learning, 8*(3):341–362.

Dayan, P. (2002). Matters temporal. *Trends in Cognitive Sciences, 6*(3):105–106.

Dayan, P., Abbott, L. F. (2001). *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. MIT Press, Cambridge, MA.

Dayan, P., Berridge, K. C. (2014). Model-based and model-free Pavlovian reward learning: Revaluation, revision, and revaluation. *Cognitive, Affective, & Behavioral Neuroscience, 14*(2):473–492.

Dayan, P., Niv, Y. (2008). Reinforcement learning: the good, the bad and the ugly. *Current Opinion in Neurobiology, 18*(2):185–196.

Dayan, P., Niv, Y., Seymour, B., Daw, N. D. (2006). The misbehavior of value and the discipline of the will. *Neural Networks, 19*(8):1153–1160.

Dayan, P., Sejnowski, T. (1994). TD(λ) converges with probability 1. *Machine Learning, 14*(3):295–301.

De Asis, K., Hernandez-Garcia, J. F., Holland, G. Z., Sutton, R. S. (2017). Multi-step Reinforcement Learning: A Unifying Algorithm. ArXiv:1703.01327.

de Farias, D. P. (2002). The Linear Programming Approach to Approximate Dynamic Programming: Theory and Application. Stanford University PhD thesis.

de Farias, D. P., Van Roy, B. (2003). The linear programming approach to approximate dynamic programming. *Operations Research 51*(6):850–865.

Dean, T., Lin, S.-H. (1995). Decomposition techniques for planning in stochastic domains. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pp. 1121–1127. Morgan Kaufmann. See also Technical Report CS-95-10, Brown University, Department of Computer Science, 1995.

Degris, T., Pilarski, P. M., Sutton, R. S. (2012). Model-free reinforcement learning with continuous action in practice. In *2012 American Control Conference*, pp. 2177–2182. IEEE.

Degris, T., White, M., Sutton, R. S. (2012). Off-policy actor–critic. In *Proceedings of the 29th International Conference on Machine Learning*. ArXiv:1205.4839, 2012.

Denardo, E. V. (1967). Contraction mappings in the theory underlying dynamic programming. *SIAM Review, 9*(2):165–177.

Dennett, D. C. (1978). Why the Law of Effect Will Not Go Away. *Brainstorms*, pp. 71–89. Bradford/MIT Press, Cambridge, MA.

Derthick, M. (1984). Variations on the Boltzmann machine learning algorithm. Carnegie-Mellon University Department of Computer Science Technical Report No. CMU-CS-84-120.

Deutsch, J. A. (1953). A new type of behaviour theory. *British Journal of Psychology. General Section, 44*(4):304–317.

Deutsch, J. A. (1954). A machine with insight. *Quarterly Journal of Experimental Psychology, 6*(1):6–11.

Dick, T. (2015). *Policy Gradient Reinforcement Learning Without Regret.* M.Sc. thesis, University of Alberta.

Dickinson, A. (1980). *Contemporary Animal Learning Theory.* Cambridge University Press.

Dickinson, A. (1985). Actions and habits: the development of behavioral autonomy. *Phil. Trans. R. Soc. Lond. B, 308*(1135):67–78.

Dickinson, A., Balleine, B. W. (2002). The role of learning in motivation. In C. R. Gallistel (Ed.), *Stevens' Handbook of Experimental Psychology*, volume 3, pp. 497–533. Wiley, NY.

Dietterich, T. G., Buchanan, B. G. (1984). The role of the critic in learning systems. In O. G. Selfridge, E. L. Rissland, and M. A. Arbib (Eds.), *Adaptive Control of Ill-Defined Systems*, pp. 127–147. Plenum Press, NY.

Dietterich, T. G., Flann, N. S. (1995). Explanation-based learning and reinforcement learning: A unified view. In A. Prieditis and S. Russell (Eds.), *Proceedings of the 12th International Conference on Machine Learning*, pp. 176–184. Morgan Kaufmann.

Dietterich, T. G., Wang, X. (2002). Batch value function approximation via support vectors. In *Advances in Neural Information Processing Systems 14*, pp. 1491–1498. MIT Press, Cambridge, MA.

Diuk, C., Cohen, A., Littman, M. L. (2008). An object-oriented representation for efficient reinforcement learning. In *Proceedings of the 25th International Conference on Machine Learning*, pp. 240–247. ACM, New York.

Dolan, R. J., Dayan, P. (2013). Goals and habits in the brain. *Neuron, 80*(2):312–325.

Doll, B. B., Simon, D. A., Daw, N. D. (2012). The ubiquity of model-based reinforcement learning. *Current Opinion in Neurobiology, 22*(6):1–7.

Donahoe, J. W., Burgos, J. E. (2000). Behavior analysis and revaluation. *Journal of the Experimental Analysis of Behavior, 74*(3):331–346.

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

Doya, K. (1996). Temporal difference learning in continuous time and space. In *Advances in Neural Information Processing Systems 8*, pp. 1073–1079. MIT Press, Cambridge, MA.

Doya, K., Sejnowski, T. J. (1995). A novel reinforcement model of birdsong vocalization learning. In *Advances in Neural Information Processing Systems 7*, pp. 101–108. MIT Press, Cambridge, MA.

Doya, K., Sejnowski, T. J. (1998). A computational model of birdsong learning by auditory experience and auditory feedback. In P. W. F. Poon and J. F. Brugge (Eds.), *Central Auditory Processing and Neural Modeling*, pp. 77–88. Springer, Boston, MA.

Doyle, P. G., Snell, J. L. (1984). *Random Walks and Electric Networks.* The Mathematical Association of America. Carus Mathematical Monograph 22.

Dreyfus, S. E., Law, A. M. (1977). *The Art and Theory of Dynamic Programming.* Academic Press, New York.

Du, S. S., Chen, J., Li, L., Xiao, L., Zhou, D. (2017). Stochastic variance reduction methods for policy evaluation. *Proceedings of the 34th International Conference on Machine Learning*, pp. 1049–1058. ArXiv:1702.07944.

Duda, R. O., Hart, P. E. (1973). *Pattern Classification and Scene Analysis.* Wiley, New York.

Duff, M. O. (1995). Q-learning for bandit problems. In *Proceedings of the 12th International Conference on Machine Learning*, pp. 209–217. Morgan Kaufmann.

Egger, D. M., Miller, N. E. (1962). Secondary reinforcement in rats as a function of information value and reliability of the stimulus. *Journal of Experimental Psychology, 64*:97–104.

Eshel, N., Tian, J., Bukwich, M., Uchida, N. (2016). Dopamine neurons share common response function for reward prediction error. *Nature Neuroscience, 19*(3):479–486.

Estes, W. K. (1943). Discriminative conditioning. I. A discriminative property of conditioned anticipation. *Journal of Experimental Psychology, 32*(2):150–155.

Estes, W. K. (1948). Discriminative conditioning. II. Effects of a Pavlovian conditioned stimulus upon a subsequently established operant response. *Journal of Experimental Psychology, 38*(2):173–177.

Estes, W. K. (1950). Toward a statistical theory of learning. *Psycholollgical Review, 57*(2):94–107.

Farley, B. G., Clark, W. A. (1954). Simulation of self-organizing systems by digital computer. *IRE Transactions on Information Theory, 4*(4):76–84.

Farries, M. A., Fairhall, A. L. (2007). Reinforcement learning with modulated spike timing-dependent synaptic plasticity. *Journal of Neurophysiology, 98*(6):3648–3665.

Feldbaum, A. A. (1965). *Optimal Control Systems.* Academic Press, New York.

Finch, G., Culler, E. (1934). Higher order conditioning with constant motivation. *The American Journal of Psychology*:596–602.

Finnsson, H., Björnsson, Y. (2008). Simulation-based approach to general game playing. In *Proceedings of the Association for the Advancement of Artificial Intelligence*, pp. 259–264.

Fiorillo, C. D., Yun, S. R., Song, M. R. (2013). Diversity and homogeneity in responses of midbrain dopamine neurons. *The Journal of Neuroscience, 33*(11):4693–4709.

Florian, R. V. (2007). Reinforcement learning through modulation of spike-timing-dependent synaptic plasticity. *Neural Computation, 19*(6):1468–1502.

Fogel, L. J., Owens, A. J., Walsh, M. J. (1966). *Artificial Intelligence through Simulated Evolution.* John Wiley and Sons.

French, R. M. (1999). Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences, 3*(4):128–135.

Frey, U., Morris, R. G. M. (1997). Synaptic tagging and long-term potentiation. *Nature, 385*(6616):533–536.

Frémaux, N., Sprekeler, H., Gerstner, W. (2010). Functional requirements for reward-modulated spike-timing-dependent plasticity. *The Journal of Neuroscience, 30*(40): 13326–13337

Friedman, J. H., Bentley, J. L., Finkel, R. A. (1977). An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software, 3*(3):209–226.

Friston, K. J., Tononi, G., Reeke, G. N., Sporns, O., Edelman, G. M. (1994). Value-dependent selection in the brain: Simulation in a synthetic neural model. *Neuroscience, 59*(2):229–243.

Fu, K. S. (1970). Learning control systems—Review and outlook. *IEEE Transactions on Automatic Control, 15*(2):210–221.

Galanter, E., Gerstenhaber, M. (1956). On thought: The extrinsic theory. *Psychological Review, 63*(4):218–227.

Gallistel, C. R. (2005). Deconstructing the law of effect. *Games and Economic Behavior, 52*(2):410–423.

Gardner, M. (1973). Mathematical games. *Scientific American, 228*(1):108–115.

Geist, M., Scherrer, B. (2014). Off-policy learning with eligibility traces: A survey. *Journal of Machine Learning Research, 15*(1):289–333.

Gelly, S., Silver, D. (2007). Combining online and offline knowledge in UCT. *Proceedings of the 24th International Conference on Machine Learning*, pp. 273–280.

Gelperin, A., Hopfield, J. J., Tank, D. W. (1985). The logic of *limax* learning. In A. Selverston (Ed.), *Model Neural Networks and Behavior*, pp. 247–261. Plenum Press, New York.

Genesereth, M., Thielscher, M. (2014). General game playing. *Synthesis Lectures on Artificial Intelligence and Machine Learning, 8*(2):1–229.

Gershman, S. J., Moustafa, A. A., Ludvig, E. A. (2014). Time representation in reinforcement learning models of the basal ganglia. *Frontiers in Computational Neuroscience, 7*:194.

Gershman, S. J., Pesaran, B., Daw, N. D. (2009). Human reinforcement learning subdivides structured action spaces by learning effector-specific values. *The Journal of Neuroscience, 29*(43):13524–13531.

Ghiassian, S., Rafiee, B., Sutton, R. S. (2016). A first empirical study of emphatic temporal difference learning. Workshop on Continual Learning and Deep Learning at the Conference on Neural Information Processing Systems. ArXiv:1705.04185.

Ghiassian, S., Patterson, A., White, M., Sutton, R. S., White, A. (2018). Online off-policy prediction. ArXiv:1811.02597.

Gibbs, C. M., Cool, V., Land, T., Kehoe, E. J., Gormezano, I. (1991). Second-order conditioning of the rabbit's nictitating membrane response. *Integrative Physiological and Behavioral Science, 26*(4):282–295.

Gittins, J. C., Jones, D. M. (1974). A dynamic allocation index for the sequential design of experiments. In J. Gani, K. Sarkadi, and I. Vincze (Eds.), *Progress in Statistics*, pp. 241–266. North-Holland, Amsterdam–London.

Glimcher, P. W. (2011). Understanding dopamine and reinforcement learning: The dopamine reward prediction error hypothesis. *Proceedings of the National Academy of Sciences, 108*(Supplement 3):15647–15654.

Glimcher, P. W. (2003). *Decisions, Uncertainty, and the Brain: The science of Neuroeconomics.* MIT Press, Cambridge, MA.

Glimcher, P. W., Fehr, E. (Eds.) (2013). *Neuroeconomics: Decision Making and the Brain, Second Edition.* Academic Press.

Goethe, J. W. V. (1878). The Sorcerer's Apprentice. In *The Permanent Goethe*, p. 349. The Dial Press, Inc., New York.

Goldstein, H. (1957). *Classical Mechanics.* Addison-Wesley, Reading, MA.

Goodfellow, I., Bengio, Y., Courville, A. (2016). *Deep Learning.* MIT Press, Cambridge, MA.

Goodwin, G. C., Sin, K. S. (1984). *Adaptive Filtering Prediction and Control.* Prentice-Hall, Englewood Cliffs, NJ.

Gopnik, A., Glymour, C., Sobel, D., Schulz, L. E., Kushnir, T., Danks, D. (2004). A theory of causal learning in children: Causal maps and Bayes nets. *Psychological Review, 111*(1):3–32.

Gordon, G. J. (1995). Stable function approximation in dynamic programming. In A. Prieditis and S. Russell (Eds.), *Proceedings of the 12th International Conference on Machine Learning*, pp. 261–268. Morgan Kaufmann. An expanded version was published as Technical Report CMU-CS-95-103. Carnegie Mellon University, Pittsburgh, PA, 1995.

Gordon, G. J. (1996a). Chattering in SARSA(λ). CMU learning lab internal report.

Gordon, G. J. (1996b). Stable fitted reinforcement learning. In *Advances in Neural Information Processing Systems 8*, pp. 1052–1058. MIT Press, Cambridge, MA.

Gordon, G. J. (1999). *Approximate Solutions to Markov Decision Processes.* PhD thesis, Carnegie Mellon University, Pittsburgh PA. Pittsburgh, PA.

Gordon, G. J. (2001). Reinforcement learning with function approximation converges to a

region. In *Advances in Neural Information Processing Systems 13*, pp. 1040–1046. MIT Press, Cambridge, MA.

Graybiel, A. M. (2000). The basal ganglia. *Current Biology, 10*(14):R509–R511.

Greensmith, E., Bartlett, P. L., Baxter, J. (2002). Variance reduction techniques for gradient estimates in reinforcement learning. In *Advances in Neural Information Processing Systems 14*, pp. 1507–1514. MIT Press, Cambridge, MA.

Greensmith, E., Bartlett, P. L., Baxter, J. (2004). Variance reduction techniques for gradient estimates in reinforcement learning. *Journal of Machine Learning Research, 5*(Nov):1471–1530.

Griffith, A. K. (1966). A new machine learning technique applied to the game of checkers. Technical Report Project MAC, Artificial Intelligence Memo 94. Massachusetts Institute of Technology, Cambridge, MA.

Griffith, A. K. (1974). A comparison and evaluation of three machine learning procedures as applied to the game of checkers. *Artificial Intelligence, 5*(2):137–148.

Grondman, I., Busoniu, L., Lopes, G. A., Babuska, R. (2012). A survey of actor–critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 42*(6):1291–1307.

Grossberg, S. (1975). A neural model of attention, reinforcement, and discrimination learning. *International Review of Neurobiology, 18*:263–327.

Grossberg, S., Schmajuk, N. A. (1989). Neural dynamics of adaptive timing and temporal discrimination during associative learning. *Neural Networks, 2*(2):79–102.

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

Gullapalli, V., Barto, A. G. (1992). Shaping as a method for accelerating reinforcement learning. In *Proceedings of the 1992 IEEE International Symposium on Intelligent Control*, pp. 554–559. IEEE.

Gurvits, L., Lin, L.-J., Hanson, S. J. (1994). Incremental learning of evaluation functions for absorbing Markov chains: New methods and theorems. Siemans Corporate Research, Princeton, NJ.

Hackman, L. (2012). *Faster Gradient-TD Algorithms*. M.Sc. thesis, University of Alberta, Edmonton.

Hallak, A., Tamar, A., Mannor, S. (2015). Emphatic TD Bellman operator is a contraction. ArXiv:1508.03411.

Hallak, A., Tamar, A., Munos, R., Mannor, S. (2016). Generalized emphatic temporal difference learning: Bias-variance analysis. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pp. 1631–1637. AAAI Press, Menlo Park, CA.

Hammer, M. (1997). The neural basis of associative reward learning in honeybees. *Trends in Neuroscience, 20*(6):245–252.

Hammer, M., Menzel, R. (1995). Learning and memory in the honeybee. *The Journal of Neuroscience, 15*(3):1617–1630.

Hampson, S. E. (1983). *A Neural Model of Adaptive Behavior*. PhD thesis, University of California, Irvine.

Hampson, S. E. (1989). *Connectionist Problem Solving: Computational Aspects of Biological Learning*. Birkhauser, Boston.

Hare, T. A., O'Doherty, J., Camerer, C. F., Schultz, W., Rangel, A. (2008). Dissociating the role of the orbitofrontal cortex and the striatum in the computation of goal values and prediction errors. *The Journal of Neuroscience, 28*(22):5623–5630.

Harth, E., Tzanakou, E. (1974). Alopex: A stochastic method for determining visual receptive fields. *Vision Research, 14*(12):1475–1482.

Hassabis, D., Maguire, E. A. (2007). Deconstructing episodic memory with construction. *Trends in Cognitive Sciences, 11*(7):299–306.

Hauskrecht, M., Meuleau, N., Kaelbling, L. P., Dean, T., Boutilier, C. (1998). Hierarchical solution of Markov decision processes using macro-actions. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, pp. 220–229. Morgan Kaufmann.

Hawkins, R. D., Kandel, E. R. (1984). Is there a cell-biological alphabet for simple forms of learning? *Psychological Review, 91*(3):375–391.

Haykin, S. (1994). *Neural networks: A Comprehensive Foundation*, Macmillan, New York.

He, K., Huertas, M., Hong, S. Z., Tie, X., Hell, J. W., Shouval, H., Kirkwood, A. (2015). Distinct eligibility traces for LTP and LTD in cortical synapses. *Neuron, 88*(3):528–538.

He, K., Zhang, X., Ren, S., Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the 1992 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778.

Hebb, D. O. (1949). *The Organization of Behavior: A Neuropsychological Theory.* John Wiley and Sons Inc., New York. Reissued by Lawrence Erlbaum Associates Inc., Mahwah NJ, 2002.

Hengst, B. (2012). Hierarchical approaches. In M. Wiering and M. van Otterlo (Eds.), *Reinforcement Learning: State-of-the-Art*, pp. 293–323. Springer-Verlag Berlin Heidelberg.

Herrnstein, R. J. (1970). On the Law of Effect. *Journal of the Experimental Analysis of Behavior, 13*(2):243–266.

Hersh, R., Griego, R. J. (1969). Brownian motion and potential theory. *Scientific American, 220*(3):66–74.

Hester, T., Stone, P. (2012). Learning and using models. In M. Wiering and M. van Otterlo (Eds.), *Reinforcement Learning: State-of-the-Art*, pp. 111–141. Springer-Verlag Berlin Heidelberg.

Hesterberg, T. C. (1988), *Advances in Importance Sampling*, PhD thesis, Statistics Department, Stanford University.

Hilgard, E. R. (1956). *Theories of Learning, Second Edition.* Appleton-Century-Cofts, Inc., New York.

Hilgard, E. R., Bower, G. H. (1975). *Theories of Learning.* Prentice-Hall, Englewood Cliffs, NJ.

Hinton, G. E. (1984). Distributed representations. Technical Report CMU-CS-84-157. Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA.

Hinton, G. E., Osindero, S., Teh, Y. (2006). A fast learning algorithm for deep belief nets. *Neural Computation, 18*(7):1527–1554.

Hochreiter, S., Schmidhuber, J. (1997). LTSM can solve hard time lag problems. In *Advances in Neural Information Processing Systems 9*, pp. 473–479. MIT Press, Cambridge, MA.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor.

Holland, J. H. (1976). Adaptation. In R. Rosen and F. M. Snell (Eds.), *Progress in Theoretical Biology*, vol. 4, pp. 263–293. Academic Press, New York.

Holland, J. H. (1986). Escaping brittleness: The possibility of general-purpose learning algorithms applied to rule-based systems. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach*, vol. 2, pp. 593–623. Morgan Kaufmann.

Hollerman, J. R., Schultz, W. (1998). Dopmine neurons report an error in the temporal prediction of reward during learning. *Nature Neuroscience, 1*(4):304–309.

Houk, J. C., Adams, J. L., Barto, A. G. (1995). A model of how the basal ganglia generates and uses neural signals that predict reinforcement. In J. C. Houk, J. L. Davis, and D. G. Beiser (Eds.), *Models of Information Processing in the Basal Ganglia*, pp. 249–270. MIT Press, Cambridge, MA.

Howard, R. (1960). *Dynamic Programming and Markov Processes.* MIT Press, Cambridge, MA.

Hull, C. L. (1932). The goal-gradient hypothesis and maze learning. *Psychological Review, 39*(1):25–43.

Hull, C. L. (1943). *Principles of Behavior.* Appleton-Century, New York.

Hull, C. L. (1952). *A Behavior System.* Wiley, New York.

Ioffe, S., Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. ArXiv:1502.03167.

İpek, E., Mutlu, O., Martínez, J. F., Caruana, R. (2008). Self-optimizing memory controllers: A reinforcement learning approach. In *ISCA'08:Proceedings of the 35th Annual International Symposium on Computer Architecture*, pp. 39–50. IEEE Computer Society Washington, DC.

Izhikevich, E. M. (2007). Solving the distal reward problem through linkage of STDP and dopamine signaling. *Cerebral Cortex, 17*(10):2443–2452.

Jaakkola, T., Jordan, M. I., Singh, S. P. (1994). On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation, 6*:1185–1201.

Jaakkola, T., Singh, S. P., Jordan, M. I. (1995). Reinforcement learning algorithm for partially observable Markov decision problems. In *Advances in Neural Information Processing Systems 7*, pp. 345–352. MIT Press, Cambridge, MA.

Jacobs, R. A. (1988). Increased rates of convergence through learning rate adaptation. *Neural Networks, 1*(4):295–307.

Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., Kavukcuoglu, K. (2016). Reinforcement learning with unsupervised auxiliary tasks. ArXiv:1611.05397.

Jaeger, H. (1997). Observable operator models and conditioned continuation representations. Arbeitspapiere der GMD 1043, GMD Forschungszentrum Informationstechnik, Sankt Augustin, Germany.

Jaeger, H. (1998). *Discrete Time, Discrete Valued Observable Operator Models: A Tutorial.* GMD-Forschungszentrum Informationstechnik.

Jaeger, H. (2000). Observable operator models for discrete stochastic time series. *Neural Computation, 12*(6):1371–1398.

Jaeger, H. (2002). Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the 'echo state network' approach. German National Research Center for Information Technology, Technical Report GMD report 159, 2002.

Joel, D., Niv, Y., Ruppin, E. (2002). Actor–critic models of the basal ganglia: New anatomical and computational perspectives. *Neural Networks, 15*(4):535–547.

Johnson, A., Redish, A. D. (2007). Neural ensembles in CA3 transiently encode paths forward of the animal at a decision point. *The Journal of Neuroscience, 27*(45):12176–12189.

Kaelbling, L. P. (1993a). Hierarchical learning in stochastic domains: Preliminary results. In *Proceedings of the 10th International Conference on Machine Learning*, pp. 167–173. Morgan Kaufmann.

Kaelbling, L. P. (1993b). *Learning in Embedded Systems.* MIT Press, Cambridge, MA.

Kaelbling, L. P. (Ed.) (1996). Special triple issue on reinforcement learning, *Machine Learning, 22*(1/2/3).

Kaelbling, L. P., Littman, M. L., Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research, 4*:237–285.

Kakade, S. M. (2002). A natural policy gradient. In *Advances in Neural Information Processing Systems 14*, pp. 1531–1538. MIT Press, Cambridge, MA.

Kakade, S. M. (2003). *On the Sample Complexity of Reinforcement Learning.* PhD thesis, University of London.

Kakutani, S. (1945). Markov processes and the Dirichlet problem. *Proceedings of the Japan Academy, 21*(3-10):227–233.

Kalos, M. H., Whitlock, P. A. (1986). *Monte Carlo Methods.* Wiley, New York.

Kamin, L. J. (1968). "Attention-like" processes in classical conditioning. In M. R. Jones (Ed.), *Miami Symposium on the Prediction of Behavior, 1967: Aversive Stimulation*, pp. 9–31. University of Miami Press, Coral Gables, Florida.

Kamin, L. J. (1969). Predictability, surprise, attention, and conditioning. In B. A. Campbell and R. M. Church (Eds.), *Punishment and Aversive Behavior*, pp. 279–296. Appleton-Century-Crofts, New York.

Kandel, E. R., Schwartz, J. H., Jessell, T. M., Siegelbaum, S. A., Hudspeth, A. J. (Eds.) (2013). *Principles of Neural Science, Fifth Edition.* McGraw-Hill Companies, Inc.

Karampatziakis, N., Langford, J. (2010). Online importance weight aware updates. ArXiv:1011.1576.

Kashyap, R. L., Blaydon, C. C., Fu, K. S. (1970). Stochastic approximation. In J. M. Mendel and K. S. Fu (Eds.), *Adaptive, Learning, and Pattern Recognition Systems: Theory and Applications*, pp. 329–355. Academic Press, New York.

Kearney, A., Veeriah, V, Travnik, J, Sutton, R. S., Pilarski, P. M. (in preparation). TIDBD: Adapting Temporal-difference Step-sizes Through Stochastic Meta-descent.

Kearns, M., Singh, S. (2002). Near-optimal reinforcement learning in polynomial time. *Machine Learning, 49*(2-3):209–232.

Keerthi, S. S., Ravindran, B. (1997). Reinforcement learning. In E. Fieslerm and R. Beale (Eds.), *Handbook of Neural Computation*, C3. Oxford University Press, New York.

Kehoe, E. J. (1982). Conditioning with serial compound stimuli: Theoretical and empirical issues. *Experimental Animal Behavior, 1*:30–65.

Kehoe, E. J., Schreurs, B. G., Graham, P. (1987). Temporal primacy overrides prior training in serial compound conditioning of the rabbit's nictitating membrane response. *Animal Learning & Behavior, 15*(4):455–464.

Keiflin, R., Janak, P. H. (2015). Dopamine prediction errors in reward learning and addiction: From theory to neural circuitry. *Neuron, 88*(2):247– 263.

Kimble, G. A. (1961). *Hilgard and Marquis' Conditioning and Learning.* Appleton-Century-Crofts, New York.

Kimble, G. A. (1967). *Foundations of Conditioning and Learning.* Appleton-Century-Crofts, New York.

Kingma, D., Ba, J. (2014). Adam: A method for stochastic optimization. ArXiv:1412.6980.

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

Klopf, A. H. (1975). A comparison of natural and artificial intelligence. *SIGART Newsletter, 53*:11–13.

Klopf, A. H. (1982). *The Hedonistic Neuron: A Theory of Memory, Learning, and Intelligence.* Hemisphere, Washington, DC.

Klopf, A. H. (1988). A neuronal model of classical conditioning. *Psychobiology, 16*(2):85–125.

Klyubin, A. S., Polani, D., Nehaniv, C. L. (2005). Empowerment: A universal agent-centric measure of control. In *Proceedings of the 2005 IEEE Congress on Evolutionary Computation* (Vol. 1, pp. 128–135). IEEE.

Kober, J., Peters, J. (2012). Reinforcement learning in robotics: A survey. In M. Wiering, M. van Otterlo (Eds.), *Reinforcement Learning: State-of-the-Art*, pp. 579–610. Springer-Verlag.

Kocsis, L., Szepesvári, Cs. (2006). Bandit based Monte-Carlo planning. In *Proceedings of the European Conference on Machine Learning*, pp. 282–293. Springer-Verlag Berlin Heidelberg.

Kohonen, T. (1977). *Associative Memory: A System Theoretic Approach.* Springer-Verlag, Berlin.

Koller, D., Friedman, N. (2009). *Probabilistic Graphical Models: Principles and Techniques.* MIT Press.

Kolodziejski, C., Porr, B., Wörgötter, F. (2009). On the asymptotic equivalence between differential Hebbian and temporal difference learning. *Neural Computation, 21*(4):1173–1202.

Kolter, J. Z. (2011). The fixed points of off-policy TD. In *Advances in Neural Information Processing Systems 24*, pp. 2169–2177. Curran Associates, Inc.

Konda, V. R., Tsitsiklis, J. N. (2000). Actor-critic algorithms. In *Advances in Neural Information Processing Systems 12*, pp. 1008–1014. MIT Press, Cambridge, MA.

Konda, V. R., Tsitsiklis, J. N. (2003). On actor-critic algorithms. *SIAM Journal on Control and Optimization, 42*(4):1143–1166.

Konidaris, G. D., Osentoski, S., Thomas, P. S. (2011). Value function approximation in reinforcement learning using the Fourier basis . In *Proceedings of the Twenty-Fifth Conference of the Association for the Advancement of Artificial Intelligence*, pp. 380–385.

Korf, R. E. (1988). Optimal path finding algorithms. In L. N. Kanal and V. Kumar (Eds.), *Search in Artificial Intelligence*, pp. 223–267. Springer-Verlag, Berlin.

Korf, R. E. (1990). Real-time heuristic search. *Artificial Intelligence, 42*(2–3), 189–211.

Koshland, D. E. (1980). *Bacterial Chemotaxis as a Model Behavioral System.* Raven Press, New York.

Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection* (Vol. 1). MIT Press., Cambridge, MA.

Kraft, L. G., Campagna, D. P. (1990). A summary comparison of CMAC neural network and traditional adaptive control systems. In T. Miller, R. S. Sutton, and P. J. Werbos (Eds.), *Neural Networks for Control*, pp. 143–169. MIT Press, Cambridge, MA.

Kraft, L. G., Miller, W. T., Dietz, D. (1992). Development and application of CMAC neural network-based control. In D. A. White and D. A. Sofge (Eds.), *Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches*, pp. 215–232. Van Nostrand Reinhold, New York.

Kumar, P. R., Varaiya, P. (1986). *Stochastic Systems: Estimation, Identification, and Adaptive Control.* Prentice-Hall, Englewood Cliffs, NJ.

Kumar, P. R. (1985). A survey of some results in stochastic adaptive control. *SIAM Journal of Control and Optimization, 23*(3):329–380.

Kumar, V., Kanal, L. N. (1988). The CDP, A unifying formulation for heuristic search, dynamic programming, and branch-and-bound. In L. N. Kanal and V. Kumar (Eds.), *Search in Artificial Intelligence*, pp. 1–37. Springer-Verlag, Berlin.

Kushner, H. J., Dupuis, P. (1992). *Numerical Methods for Stochastic Control Problems in Continuous Time.* Springer-Verlag, New York.

Kuvayev, L., Sutton, R.S. (1996). Model-based reinforcement learning with an approximate, learned model. *Proceedings of the Ninth Yale Workshop on Adaptive and Learning Systems*, pp. 101–105, Yale University, New Haven, CT.

Lagoudakis, M., Parr, R. (2003). Least squares policy iteration. *Journal of Machine Learning Research, 4*(Dec):1107–1149.

Lai, T. L., Robbins, H. (1985). Asymptotically efficient adaptive allocation rules. *Advances in Applied Mathematics, 6*(1):4–22.

Lakshmivarahan, S., Narendra, K. S. (1982). Learning algorithms for two-person zero-sum stochastic games with incomplete information: A unified approach. *SIAM Journal of Control and Optimization, 20*(4):541–552.

Lammel, S., Lim, B. K., Malenka, R. C. (2014). Reward and aversion in a heterogeneous midbrain dopamine system. *Neuropharmacology, 76*:353–359.

Lane, S. H., Handelman, D. A., Gelfand, J. J. (1992). Theory and development of higher-order CMAC neural networks. *IEEE Control Systems, 12*(2):23–30.

LeCun, Y. (1985). Une procédure d'apprentissage pour réseau a seuil asymmetrique (a learning scheme for asymmetric threshold networks). In *Proceedings of Cognitiva 85*, Paris, France.

LeCun, Y., Bottou, L., Bengio, Y., Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE, 86*(11):2278–2324.

Legenstein, R. W., Maass, D. P. (2008). A learning theory for reward-modulated spike-timing-dependent plasticity with application to biofeedback. *PLoS Computational Biology, 4*(10).

Levy, W. B., Steward, D. (1983). Temporal contiguity requirements for long-term associative potentiation/depression in the hippocampus. *Neuroscience, 8*(4):791–797.

Lewis, F. L., Liu, D. (Eds.) (2012). *Reinforcement Learning and Approximate Dynamic Programming for Feedback Control*. John Wiley and Sons.

Lewis, R. L., Howes, A., Singh, S. (2014). Computational rationality: Linking mechanism and behavior through utility maximization. *Topics in Cognitive Science, 6*(2):279–311.

Li, L. (2012). Sample complexity bounds of exploration. In M. Wiering and M. van Otterlo (Eds.), *Reinforcement Learning: State-of-the-Art*, pp. 175–204. Springer-Verlag Berlin Heidelberg.

Li, L., Chu, W., Langford, J., Schapire, R. E. (2010). A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th International Conference on World Wide Web*, pp. 661–670. ACM, New York.

Lin, C.-S., Kim, H. (1991). CMAC-based adaptive critic self-learning control. *IEEE Transactions on Neural Networks, 2*(5):530–533.

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

Lin, L.-J., Mitchell, T. (1992). Reinforcement learning with hidden states. In *Proceedings of the Second International Conference on Simulation of Adaptive Behavior: From Animals to Animats*, pp. 271–280. MIT Press, Cambridge, MA.

Littman, M. L., Cassandra, A. R., Kaelbling, L. P. (1995). Learning policies for partially observable environments: Scaling up. In *Proceedings of the 12th International Conference on Machine Learning*, pp. 362–370. Morgan Kaufmann.

Littman, M. L., Dean, T. L., Kaelbling, L. P. (1995). On the complexity of solving Markov decision problems. In *Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence*, pp. 394–402.

Littman, M. L., Sutton, R. S., Singh, S. (2002). Predictive representations of state. In *Advances in Neural Information Processing Systems 14*, pp. 1555–1561. MIT Press, Cambridge, MA.

Liu, J. S. (2001). *Monte Carlo Strategies in Scientific Computing*. Springer-Verlag, Berlin.

Ljung, L. (1998). System identification. In A. Procházka, J. Uhlíř, P. W. J. Rayner, and N. G. Kingsbury (Eds.), *Signal Analysis and Prediction*, pp. 163–173. Springer Science + Business Media New York, LLC.

Ljung, L., Söderstrom, T. (1983). *Theory and Practice of Recursive Identification.* MIT Press, Cambridge, MA.

Ljungberg, T., Apicella, P., Schultz, W. (1992). Responses of monkey dopamine neurons during learning of behavioral reactions. *Journal of Neurophysiology, 67*(1):145–163.

Lovejoy, W. S. (1991). A survey of algorithmic methods for partially observed Markov decision processes. *Annals of Operations Research, 28*(1):47–66.

Luce, D. (1959). *Individual Choice Behavior.* Wiley, New York.

Ludvig, E. A., Bellemare, M. G., Pearson, K. G. (2011). A primer on reinforcement learning in the brain: Psychological, computational, and neural perspectives. In E. Alonso and E. Mondragón (Eds.), *Computational Neuroscience for Advancing Artificial Intelligence: Models, Methods and Applications*, pp. 111–44. Medical Information Science Reference, Hershey PA.

Ludvig, E. A., Sutton, R. S., Kehoe, E. J. (2008). Stimulus representation and the timing of reward-prediction errors in models of the dopamine system. *Neural Computation, 20*(12):3034–3054.

Ludvig, E. A., Sutton, R. S., Kehoe, E. J. (2012). Evaluating the TD model of classical conditioning. *Learning & behavior, 40*(3):305–319.

Machado, A. (1997). Learning the temporal dynamics of behavior. *Psychological Review, 104*(2):241–265.

Mackintosh, N. J. (1975). A theory of attention: Variations in the associability of stimuli with reinforcement. *Psychological Review, 82*(4):276–298.

Mackintosh, N. J. (1983). *Conditioning and Associative Learning.* Clarendon Press, Oxford.

Maclin, R., Shavlik, J. W. (1994). Incorporating advice into agents that learn from reinforcements. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pp. 694–699. AAAI Press, Menlo Park, CA.

Maei, H. R. (2011). *Gradient Temporal-Difference Learning Algorithms.* PhD thesis, University of Alberta, Edmonton.

Maei, H. R. (2018). Convergent actor-critic algorithms under off-policy training and function approximation. ArXiv:1802.07842.

Maei, H. R., Sutton, R. S. (2010). GQ(λ): A general gradient algorithm for temporal-difference prediction learning with eligibility traces. In *Proceedings of the Third Conference on Artificial General Intelligence,* pp. 91–96.

Maei, H. R., Szepesvári, Cs., Bhatnagar, S., Precup, D., Silver, D., Sutton, R. S. (2009). Convergent temporal-difference learning with arbitrary smooth function approximation. In *Advances in Neural Information Processing Systems 22*, pp. 1204–1212. Curran Associates, Inc.

Maei, H. R., Szepesvári, Cs., Bhatnagar, S., Sutton, R. S. (2010). Toward off-policy learning control with function approximation. In *Proceedings of the 27th International Conference on Machine Learning*, pp. 719–726).

Mahadevan, S. (1996). Average reward reinforcement learning: Foundations, algorithms, and empirical results. *Machine Learning, 22*(1):159–196.

Mahadevan, S., Liu, B., Thomas, P., Dabney, W., Giguere, S., Jacek, N., Gemp, I., Liu, J. (2014). Proximal reinforcement learning: A new theory of sequential decision making in primal-dual spaces. ArXiv:1405.6757.

Mahadevan, S., Connell, J. (1992). Automatic programming of behavior-based robots using

reinforcement learning. *Artificial Intelligence, 55*(2-3):311–365.

Mahmood, A. R. (2017). *Incremental Off-Policy Reinforcement Learning Algorithms.* PhD thesis, University of Alberta, Edmonton.

Mahmood, A. R., Sutton, R. S. (2015). Off-policy learning based on weighted importance sampling with linear computational complexity. In *Proceedings of the 31st Conference on Uncertainty in Artificial Intelligence*, pp. 552–561. AUAI Press Corvallis, Oregon.

Mahmood, A. R., Sutton, R. S., Degris, T., Pilarski, P. M. (2012). Tuning-free step-size adaptation. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing, Proceedings*, pp. 2121–2124. IEEE.

Mahmood, A. R., Yu, H, Sutton, R. S. (2017). Multi-step off-policy learning without importance sampling ratios. ArXiv:1702.03006.

Mahmood, A. R., van Hasselt, H., Sutton, R. S. (2014). Weighted importance sampling for off-policy learning with linear function approximation. *Advances in Neural Information Processing Systems 27*, pp. 3014–3022. Curran Associates, Inc.

Marbach, P., Tsitsiklis, J. N. (1998). Simulation-based optimization of Markov reward processes. MIT Technical Report LIDS-P-2411.

Marbach, P., Tsitsiklis, J. N. (2001). Simulation-based optimization of Markov reward processes. *IEEE Transactions on Automatic Control, 46*(2):191–209.

Markram, H., Lübke, J., Frotscher, M., Sakmann, B. (1997). Regulation of synaptic efficacy by coincidence of postsynaptic APs and EPSPs. *Science, 275*(5297):213–215.

Martínez, J. F., İpek, E. (2009). Dynamic multicore resource management: A machine learning approach. *Micro, IEEE, 29*(5):8–17.

Mataric, M. J. (1994). Reward functions for accelerated learning. In *Proceedings of the 11th International Conference on Machine Learning*, pp. 181–189. Morgan Kaufmann.

Matsuda, W., Furuta, T., Nakamura, K. C., Hioki, H., Fujiyama, F., Arai, R., Kaneko, T. (2009). Single nigrostriatal dopaminergic neurons form widely spread and highly dense axonal arborizations in the neostriatum. *The Journal of Neuroscience, 29*(2):444–453.

Mazur, J. E. (1994). *Learning and Behavior*, 3rd ed. Prentice-Hall, Englewood Cliffs, NJ.

McCallum, A. K. (1993). Overcoming incomplete perception with utile distinction memory. In *Proceedings of the 10th International Conference on Machine Learning*, pp. 190–196. Morgan Kaufmann.

McCallum, A. K. (1995). *Reinforcement Learning with Selective Perception and Hidden State.* PhD thesis, University of Rochester, Rochester NY.

McCloskey, M., Cohen, N. J. (1989). Catastrophic interference in connectionist networks: The sequential learning problem. *Psychology of Learning and Motivation, 24*:109–165.

McClure, S. M., Daw, N. D., Montague, P. R. (2003). A computational substrate for incentive salience. *Trends in Neurosciences, 26*(8):423–428.

McCulloch, W. S., Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics, 5*(4):115–133.

McMahan, H. B., Gordon, G. J. (2005). Fast Exact Planning in Markov Decision Processes. In *Proceedings of the International Conference on Automated Planning and Scheduling*, pp. 151-160.

Melo, F. S., Meyn, S. P., Ribeiro, M. I. (2008). An analysis of reinforcement learning with function approximation. In *Proceedings of the 25th International Conference on Machine Learning*, pp. 664–671.

Mendel, J. M. (1966). A survey of learning control systems. *ISA Transactions, 5*:297–303.

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

Michie, D. (1961). Trial and error. In S. A. Barnett and A. McLaren (Eds.), *Science Survey, Part 2*, pp. 129–145. Penguin, Harmondsworth.

Michie, D. (1963). Experiments on the mechanisation of game learning. 1. characterization of the model and its parameters. *The Computer Journal, 6*(3):232–263.

Michie, D. (1974). *On Machine Intelligence*. Edinburgh University Press, Edinburgh.

Michie, D., Chambers, R. A. (1968). BOXES, An experiment in adaptive control. In E. Dale and D. Michie (Eds.), *Machine Intelligence 2*, pp. 137–152. Oliver and Boyd, Edinburgh.

Miller, R. (1981). *Meaning and Purpose in the Intact Brain: A Philosophical, Psychological, and Biological Account of Conscious Process*. Clarendon Press, Oxford.

Miller, W. T., An, E., Glanz, F., Carter, M. (1990). The design of CMAC neural networks for control. *Adaptive and Learning Systems, 1*:140–145.

Miller, W. T., Glanz, F. H. (1996). *UNH_CMAC verison 2.1: The University of New Hampshire Implementation of the Cerebellar Model Arithmetic Computer - CMAC*. Robotics Laboratory Technical Report, University of New Hampshire, Durham.

Miller, S., Williams, R. J. (1992). Learning to control a bioreactor using a neural net Dyna-Q system. In *Proceedings of the Seventh Yale Workshop on Adaptive and Learning Systems*, pp. 167–172. Center for Systems Science, Dunham Laboratory, Yale University, New Haven.

Miller, W. T., Scalera, S. M., Kim, A. (1994). Neural network control of dynamic balance for a biped walking robot. In *Proceedings of the Eighth Yale Workshop on Adaptive and Learning Systems*, pp. 156–161. Center for Systems Science, Dunham Laboratory, Yale University, New Haven.

Minton, S. (1990). Quantitative results concerning the utility of explanation-based learning. *Artificial Intelligence, 42*(2-3):363–391.

Minsky, M. L. (1954). *Theory of Neural-Analog Reinforcement Systems and Its Application to the Brain-Model Problem*. PhD thesis, Princeton University.

Minsky, M. L. (1961). Steps toward artificial intelligence. *Proceedings of the Institute of Radio Engineers*, 49:8–30. Reprinted in E. A. Feigenbaum and J. Feldman (Eds.), *Computers and Thought*, pp. 406–450. McGraw-Hill, New York, 1963.

Minsky, M. L. (1967). *Computation: Finite and Infinite Machines*. Prentice-Hall, Englewood Cliffs, NJ.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M. (2013). Playing atari with deep reinforcement learning. ArXiv:1312.5602.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature, 518*(7540):529–533.

Modayil, J., Sutton, R. S. (2014). Prediction driven behavior: Learning predictions that drive fixed responses. In *AAAI-14 Workshop on Artificial Intelligence and Robotics*, Quebec City, Canada.

Modayil, J., White, A., Sutton, R. S. (2014). Multi-timescale nexting in a reinforcement learning robot. *Adaptive Behavior, 22*(2):146–160.

Monahan, G. E. (1982). State of the art—a survey of partially observable Markov decision processes: theory, models, and algorithms. *Management Science, 28*(1):1–16.

Montague, P. R., Dayan, P., Nowlan, S. J., Pouget, A., Sejnowski, T. J. (1993). Using aperiodic reinforcement for directed self-organization during development. In *Advances in Neural Information Processing Systems 5*, pp. 969–976. Morgan Kaufmann.

Montague, P. R., Dayan, P., Person, C., Sejnowski, T. J. (1995). Bee foraging in uncertain environments using predictive hebbian learning. *Nature, 377*(6551):725–728.

Montague, P. R., Dayan, P., Sejnowski, T. J. (1996). A framework for mesencephalic dopamine systems based on predictive Hebbian learning. *The Journal of Neuroscience, 16*(5):1936–1947.

Montague, P. R., Dolan, R. J., Friston, K. J., Dayan, P. (2012). Computational psychiatry. *Trends in Cognitive Sciences, 16*(1):72–80.

Montague, P. R., Sejnowski, T. J. (1994). The predictive brain: Temporal coincidence and temporal order in synaptic learningmechanisms. *Learning & Memory, 1*(1):1–33.

Moore, A. W. (1990). *Efficient Memory-Based Learning for Robot Control.* PhD thesis, University of Cambridge.

Moore, A. W., Atkeson, C. G. (1993). Prioritized sweeping: Reinforcement learning with less data and less real time. *Machine Learning, 13*(1):103–130.

Moore, A. W., Schneider, J., Deng, K. (1997). Efficient locally weighted polynomial regression predictions. In *Proceedings of the 14th International Conference on Machine Learning.* Morgan Kaufmann.

Moore, J. W., Blazis, D. E. J. (1989). Simulation of a classically conditioned response: A cerebellar implementation of the sutton-barto-desmond model. In J. H. Byrne and W. O. Berry (Eds.), *Neural Models of Plasticity*, pp. 187–207. Academic Press, San Diego, CA.

Moore, J. W., Choi, J.-S., Brunzell, D. H. (1998). Predictive timing under temporal uncertainty: The time derivative model of the conditioned response. In D. A. Rosenbaum and C. E. Collyer (Eds.), *Timing of Behavior*, pp. 3–34. MIT Press, Cambridge, MA.

Moore, J. W., Desmond, J. E., Berthier, N. E., Blazis, E. J., Sutton, R. S., Barto, A. G. (1986). Simulation of the classically conditioned nictitating membrane response by a neuron-like adaptive element: I. Response topography, neuronal firing, and interstimulus intervals. *Behavioural Brain Research, 21*(2):143–154.

Moore, J. W., Marks, J. S., Castagna, V. E., Polewan, R. J. (2001). Parameter stability in the TD model of complex CR topographies. In *Society for Neuroscience Abstracts, 27*:642.

Moore, J. W., Schmajuk, N. A. (2008). Kamin blocking. *Scholarpedia, 3*(5):3542.

Moore, J. W., Stickney, K. J. (1980). Formation of attentional-associative networks in real time:Role of the hippocampus and implications for conditioning. *Physiological Psychology, 8*(2):207–217.

Mukundan, J., Martínez, J. F. (2012). MORSE, Multi-objective reconfigurable self-optimizing memory scheduler. In *IEEE 18th International Symposium on High Performance Computer Architecture*, pp. 1–12.

Müller, M. (2002). Computer Go. *Artificial Intelligence, 134*(1):145–179.

Munos, R., Stepleton, T., Harutyunyan, A., Bellemare, M. (2016). Safe and efficient off-policy reinforcement learning. In *Advances in Neural Information Processing Systems 29*, pp. 1046–1054. Curran Associates, Inc.

Naddaf, Y. (2010). *Game-Independent AI Agents for Playing Atari 2600 Console Games.* PhD thesis, University of Alberta, Edmonton.

Narendra, K. S., Thathachar, M. A. L. (1974). Learning automata—A survey. *IEEE Transactions on Systems, Man, and Cybernetics, 4*:323–334.

Narendra, K. S., Thathachar, M. A. L. (1989). *Learning Automata: An Introduction.* Prentice-Hall, Englewood Cliffs, NJ.

Narendra, K. S., Wheeler, R. M. (1983). An N-player sequential stochastic game with identical payoffs. *IEEE Transactions on Systems, Man, and Cybernetics, 6*:1154–1158.

Narendra, K. S., Wheeler, R. M. (1986). Decentralized learning in finite Markov chains. *IEEE Transactions on Automatic Control, 31*(6):519–526.

Nedić, A., Bertsekas, D. P. (2003). Least squares policy evaluation algorithms with linear function approximation. *Discrete Event Dynamic Systems, 13*(1-2):79–110.

Ng, A. Y. (2003). *Shaping and Policy Search in Reinforcement Learning*. PhD thesis, University of California, Berkeley.

Ng, A. Y., Harada, D., Russell, S. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In I. Bratko and S. Dzeroski (Eds.), *Proceedings of the 16th International Conference on Machine Learning*, pp. 278–287.

Ng, A. Y., Russell, S. J. (2000). Algorithms for inverse reinforcement learning. In *Proceedings of the 17th International Conference on Machine Learning*, pp. 663–670.

Niv, Y. (2009). Reinforcement learning in the brain. *Journal of Mathematical Psychology, 53*(3):139–154.

Niv, Y., Daw, N. D., Dayan, P. (2006). How fast to work: Response vigor, motivation and tonic dopamine. In *Advances in Neural Information Processing Systems 18*, pp. 1019–1026. MIT Press, Cambridge, MA.

Niv, Y., Daw, N. D., Joel, D., Dayan, P. (2007). Tonic dopamine: opportunity costs and the control of response vigor. *Psychopharmacology, 191*(3):507–520.

Niv, Y., Joel, D., Dayan, P. (2006). A normative perspective on motivation. *Trends in Cognitive Sciences, 10*(8):375–381.

Nouri, A., Littman, M. L. (2009). Multi-resolution exploration in continuous spaces. In *Advances in Neural Information Processing Systems 21*, pp. 1209–1216. Curran Associates, Inc.

Nowé, A., Vrancx, P., Hauwere, Y.-M. D. (2012). Game theory and multi-agent reinforcement learning. In M. Wiering and M. van Otterlo (Eds.), *Reinforcement Learning: State-of-the-Art*, pp. 441–467. Springer-Verlag Berlin Heidelberg.

Nutt, D. J., Lingford-Hughes, A., Erritzoe, D., Stokes, P. R. A. (2015). The dopamine theory of addiction: 40 years of highs and lows. *Nature Reviews Neuroscience, 16*(5):305–312.

O'Doherty, J. P., Dayan, P., Friston, K., Critchley, H., Dolan, R. J. (2003). Temporal difference models and reward-related learning in the human brain. *Neuron, 38*(2):329–337.

O'Doherty, J. P., Dayan, P., Schultz, J., Deichmann, R., Friston, K., Dolan, R. J. (2004). Dissociable roles of ventral and dorsal striatum in instrumental conditioning. *Science, 304*(5669):452–454.

Ólafsdóttir, H. F., Barry, C., Saleem, A. B., Hassabis, D., Spiers, H. J. (2015). Hippocampal place cells construct reward related sequences through unexplored space. *Elife, 4*:e06063.

Oh, J., Guo, X., Lee, H., Lewis, R. L., Singh, S. (2015). Action-conditional video prediction using deep networks in Atari games. In *Advances in Neural Information Processing Systems 28*, pp. 2845–2853. Curran Associates, Inc.

Olds, J., Milner, P. (1954). Positive reinforcement produced by electrical stimulation of the septal area and other regions of rat brain. *Journal of Comparative and Physiological Psychology, 47*(6):419–427.

O'Reilly, R. C., Frank, M. J. (2006). Making working memory work: A computational model of learning in the prefrontal cortex and basal ganglia. *Neural Computation, 18*(2):283–328.

O'Reilly, R. C., Frank, M. J., Hazy, T. E., Watz, B. (2007). PVLV, the primary value and learned value Pavlovian learning algorithm. *Behavioral Neuroscience, 121*(1):31–49.

Omohundro, S. M. (1987). Efficient algorithms with neural network behavior. Technical Report, Department of Computer Science, University of Illinois at Urbana-Champaign.

Ormoneit, D., Sen, Ś. (2002). Kernel-based reinforcement learning. *Machine Learning, 49*(2-3):161–178.

Oudeyer, P.-Y., Kaplan, F. (2007). What is intrinsic motivation? A typology of computational approaches. *Frontiers in Neurorobotics, 1*:6.

Oudeyer, P.-Y., Kaplan, F., Hafner, V. V. (2007). Intrinsic motivation systems for autonomous mental development. *IEEE Transactions on Evolutionary Computation, 11*(2):265–286.

Padoa-Schioppa, C., Assad, J. A. (2006). Neurons in the orbitofrontal cortex encode economic value. *Nature, 441*(7090):223–226.

Page, C. V. (1977). Heuristics for signature table analysis as a pattern recognition technique. *IEEE Transactions on Systems, Man, and Cybernetics, 7*(2):77–86.

Pagnoni, G., Zink, C. F., Montague, P. R., Berns, G. S. (2002). Activity in human ventral striatum locked to errors of reward prediction. *Nature Neuroscience, 5*(2):97–98.

Pan, W.-X., Schmidt, R., Wickens, J. R., Hyland, B. I. (2005). Dopamine cells respond to predicted events during classical conditioning: Evidence for eligibility traces in the reward-learning network. *The Journal of Neuroscience, 25*(26):6235–6242.

Park, J., Kim, J., Kang, D. (2005). An RLS-based natural actor–critic algorithm for locomotion of a two-linked robot arm. *Computational Intelligence and Security*:65–72.

Parks, P. C., Militzer, J. (1991). Improved allocation of weights for associative memory storage in learning control systems. In *IFAC Design Methods of Control Systems*, Zurich, Switzerland, pp. 507–512.

Parr, R. (1988). *Hierarchical Control and Learning for Markov Decision Processes.* PhD thesis, University of California, Berkeley.

Parr, R., Li, L., Taylor, G., Painter-Wakefield, C., Littman, M. L. (2008). An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning. In *Proceedings of the 25th international conference on Machine learning*, pp. 752–759).

Parr, R., Russell, S. (1995). Approximating optimal policies for partially observable stochastic domains. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pp. 1088–1094. Morgan Kaufmann.

Pavlov, I. P. (1927). *Conditioned Reflexes.* Oxford University Press, London.

Pawlak, V., Kerr, J. N. D. (2008). Dopamine receptor activation is required for corticostriatal spike-timing-dependent plasticity. *The Journal of Neuroscience, 28*(10):2435–2446.

Pawlak, V., Wickens, J. R., Kirkwood, A., Kerr, J. N. D. (2010). Timing is not everything: neuromodulation opens the STDP gate. *Frontiers in Synaptic Neuroscience, 2*:146. doi:10.3389/fnsyn.2010.00146.

Pearce, J. M., Hall, G. (1980). A model for Pavlovian learning: Variation in the effectiveness of conditioning but not unconditioned stimuli. *Psychological Review, 87*(6):532–552.

Pearl, J. (1984). *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* Addison-Wesley, Reading, MA.

Pearl, J. (1995). Causal diagrams for empirical research. *Biometrika*, 82(4):669-688.

Pecevski, D., Maass, W., Legenstein, R. A. (2008). Theoretical analysis of learning with reward-modulated spike-timing-dependent plasticity. In *Advances in Neural Information Processing Systems 20*, pp. 881–888. Curran Associates, Inc.

Peng, J. (1993). *Efficient Dynamic Programming-Based Learning for Control.* PhD thesis, Northeastern University, Boston MA.

Peng, J. (1995). Efficient memory-based dynamic programming. In *Proceedings of the 12th International Conference on Machine Learning*, pp. 438–446.

Peng, J., Williams, R. J. (1993). Efficient learning and planning within the Dyna framework. *Adaptive Behavior, 1*(4):437–454.

Peng, J., Williams, R. J. (1994). Incremental multi-step Q-learning. In *Proceedings of the 11th International Conference on Machine Learning*, pp. 226–232. Morgan Kaufmann, San Francisco.

Peng, J., Williams, R. J. (1996). Incremental multi-step Q-learning. *Machine Learning, 22*(1):283–290.

Perkins, T. J., Pendrith, M. D. (2002). On the existence of fixed points for Q-learning and Sarsa in partially observable domains. In *Proceedings of the 19th International Conference on Machine Learning*, pp. 490–497.

Perkins, T. J., Precup, D. (2003). A convergent form of approximate policy iteration. In *Advances in Neural Information Processing Systems 15*, pp. 1627–1634. MIT Press, Cambridge, MA.

Peters, J., Büchel, C. (2010). Neural representations of subjective reward value. *Behavioral Brain Research, 213*(2):135–141.

Peters, J., Schaal, S. (2008). Natural actor–critic. *Neurocomputing, 71*(7):1180–1190.

Peters, J., Vijayakumar, S., Schaal, S. (2005). Natural actor–critic. In *European Conference on Machine Learning*, pp. 280–291. Springer Berlin Heidelberg.

Pezzulo, G., van der Meer, M. A. A., Lansink, C. S., Pennartz, C. M. A. (2014). Internally generated sequences in learning and executing goal-directed behavior. *Trends in Cognitive Science, 18*(12):647–657.

Pfeiffer, B. E., Foster, D. J. (2013). Hippocampal place-cell sequences depict future paths to remembered goals. *Nature, 497*(7447):74–79.

Phansalkar, V. V., Thathachar, M. A. L. (1995). Local and global optimization algorithms for generalized learning automata. *Neural Computation, 7*(5):950–973.

Poggio, T., Girosi, F. (1989). A theory of networks for approximation and learning. A.I. Memo 1140. Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA.

Poggio, T., Girosi, F. (1990). Regularization algorithms for learning that are equivalent to multilayer networks. *Science, 247*(4945):978–982.

Polyak, B. T. (1990). New stochastic approximation type procedures. *Automat. i Telemekh, 7*(98-107):2 (in Russian).

Polyak, B. T., Juditsky, A. B. (1992). Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization, 30*(4):838–855.

Powell, M. J. D. (1987). Radial basis functions for multivariate interpolation: A review. In J. C. Mason and M. G. Cox (Eds.), *Algorithms for Approximation*, pp. 143–167. Clarendon Press, Oxford.

Powell, W. B. (2011). *Approximate Dynamic Programming: Solving the Curses of Dimensionality*, Second edition. John Wiley and Sons.

Powers, W. T. (1973). *Behavior: The Control of Perception.* Aldine de Gruyter, Chicago. 2nd expanded edition 2005.

Precup, D. (2000). *Temporal Abstraction in Reinforcement Learning.* PhD thesis, University of Massachusetts, Amherst.

Precup, D., Sutton, R. S., Dasgupta, S. (2001). Off-policy temporal-difference learning with function approximation. In *Proceedings of the 18th International Conference on Machine Learning*, pp. 417–424.

Precup, D., Sutton, R. S., Paduraru, C., Koop, A., Singh, S. (2006). Off-policy learning with options and recognizers. In *Advances in Neural Information Processing Systems 18*, pp. 1097–1104. MIT Press, Cambridge, MA.

Precup, D., Sutton, R. S., Singh, S. (2000). Eligibility traces for off-policy policy evaluation. In *Proceedings of the 17th International Conference on Machine Learning*, pp. 759–766. Morgan Kaufmann.

Puterman, M. L. (1994). *Markov Decision Problems*. Wiley, New York.

Puterman, M. L., Shin, M. C. (1978). Modified policy iteration algorithms for discounted Markov decision problems. *Management Science, 24*(11):1127–1137.

Quartz, S., Dayan, P., Montague, P. R., Sejnowski, T. J. (1992). Expectation learning in the brain using diffuse ascending connections. In *Society for Neuroscience Abstracts, 18*:1210.

Randløv, J., Alstrøm, P. (1998). Learning to drive a bicycle using reinforcement learning and shaping. In *Proceedings of the 15th International Conference on Machine Learning*, pp. 463–471.

Rangel, A., Camerer, C., Montague, P. R. (2008). A framework for studying the neurobiology of value-based decision making. *Nature Reviews Neuroscience, 9*(7):545–556.

Rangel, A., Hare, T. (2010). Neural computations associated with goal-directed choice. *Current Opinion in Neurobiology, 20*(2):262–270.

Rao, R. P., Sejnowski, T. J. (2001). Spike-timing-dependent Hebbian plasticity as temporal difference learning. *Neural Computation, 13*(10):2221–2237.

Ratcliff, R. (1990). Connectionist models of recognition memory: Constraints imposed by learning and forgetting functions. *Psychological Review, 97*(2):285–308.

Reddy, G., Celani, A., Sejnowski, T. J., Vergassola, M. (2016). Learning to soar in turbulent environments. *Proceedings of the National Academy of Sciences, 113*(33):E4877–E4884.

Redish, D. A. (2004). Addiction as a computational process gone awry. *Science, 306*(5703):1944–1947.

Reetz, D. (1977). Approximate solutions of a discounted Markovian decision process. *Bonner Mathematische Schriften, 98*:77–92.

Rescorla, R. A., Wagner, A. R. (1972). A theory of Pavlovian conditioning: Variations in the effectiveness of reinforcement and nonreinforcement. In A. H. Black and W. F. Prokasy (Eds.), *Classical Conditioning II*, pp. 64–99. Appleton-Century-Crofts, New York.

Revusky, S., Garcia, J. (1970). Learned associations over long delays. In G. Bower (Ed.), *The Psychology of Learning and Motivation*, v. 4, pp. 1–84. Academic Press, Inc., New York.

Reynolds, J. N. J., Wickens, J. R. (2002). Dopamine-dependent plasticity of corticostriatal synapses. *Neural Networks, 15*(4):507–521.

Ring, M. B. (in preparation). Representing knowledge as forecasts (and state as knowledge).

Ripley, B. D. (2007). *Pattern Recognition and Neural Networks*. Cambridge University Press.

Rixner, S. (2004). Memory controller optimizations for web servers. In *Proceedings of the 37th annual IEEE/ACM International Symposium on Microarchitecture*, p. 355–366. IEEE Computer Society.

Robbins, H. (1952). Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society, 58*:527–535.

Robertie, B. (1992). Carbon versus silicon: Matching wits with TD-Gammon. *Inside Backgammon, 2*(2):14–22.

Romo, R., Schultz, W. (1990). Dopamine neurons of the monkey midbrain: Contingencies of responses to active touch during self-initiated arm movements. *Journal of Neurophysiology, 63*(3):592–624.

Rosenblatt, F. (1962). *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Spartan Books, Washington, DC.

Ross, S. (1983). *Introduction to Stochastic Dynamic Programming*. Academic Press, New York.

Ross, T. (1933). Machines that think. *Scientific American, 148*(4):206–208.

Rubinstein, R. Y. (1981). *Simulation and the Monte Carlo Method*. Wiley, New York.

Rumelhart, D. E., Hinton, G. E., Williams, R. J. (1986). Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland (Eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, vol. I, *Foundations*. Bradford/MIT Press, Cambridge, MA.

Rummery, G. A. (1995). *Problem Solving with Reinforcement Learning*. PhD thesis, University of Cambridge.

Rummery, G. A., Niranjan, M. (1994). On-line Q-learning using connectionist systems. Technical Report CUED/F-INFENG/TR 166. Engineering Department, Cambridge University.

Ruppert, D. (1988). Efficient estimations from a slowly convergent Robbins-Monro process. Cornell University Operations Research and Industrial Engineering Technical Report No. 781.

Russell, S., Norvig, P. (2009). *Artificial Intelligence: A Modern Approach*, 3rd edition. Prentice-Hall, Englewood Cliffs, NJ.

Russo, D. J., Van Roy, B., Kazerouni, A., Osband, I., Wen, Z. (2018). A tutorial on Thompson sampling, *Foundations and Trends in Machine Learning*. ArXiv:1707.02038.

Rust, J. (1996). Numerical dynamic programming in economics. In H. Amman, D. Kendrick, and J. Rust (Eds.), *Handbook of Computational Economics*, pp. 614–722. Elsevier, Amsterdam.

Saddoris, M. P., Cacciapaglia, F., Wightmman, R. M., Carelli, R. M. (2015). Differential dopamine release dynamics in the nucleus accumbens core and shell reveal complementary signals for error prediction and incentive motivation. *The Journal of Neuroscience, 35*(33):11572–11582.

Saksida, L. M., Raymond, S. M., Touretzky, D. S. (1997). Shaping robot behavior using principles from instrumental conditioning. *Robotics and Autonomous Systems, 22*(3):231–249.

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal on Research and Development, 3*(3), 210–229.

Samuel, A. L. (1967). Some studies in machine learning using the game of checkers. II—Recent progress. *IBM Journal on Research and Development, 11*(6):601–617.

Schaal, S., Atkeson, C. G. (1994). Robot juggling: Implementation of memory-based learning. *IEEE Control Systems, 14*(1):57–71.

Schmajuk, N. A. (2008). Computational models of classical conditioning. *Scholarpedia, 3*(3):1664.

Schmidhuber, J. (1991a). Curious model-building control systems. In *Proceedings of the IEEE International Joint Conference on Neural Networks*, pp. 1458–1463. IEEE.

Schmidhuber, J. (1991b). A possibility for implementing curiosity and boredom in model-building neural controllers. In *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior*, pp. 222–227. MIT Press, Cambridge, MA.

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks, 6*:85–117.

Schmidhuber, J., Storck, J., Hochreiter, S. (1994). Reinforcement driven information acquisition in nondeterministic environments. Technical report, Fakultät für Informatik, Technische Universität München, München, Germany.

Schraudolph, N. N. (1999). Local gain adaptation in stochastic gradient descent. In *Proceedings of the International Conference on Artificial Neural Networks*, pp. 569–574. IEEE, London.

Schraudolph, N. N. (2002). Fast curvature matrix-vector products for second-order gradient descent. *Neural Computation, 14*(7):1723–1738.

Schraudolph, N. N., Yu, J., Aberdeen, D. (2006). Fast online policy gradient learning with SMD gain vector adaptation. In *Advances in Neural Information Processing Systems*, pp. 1185–1192.

Schulman, J., Chen, X., Abbeel, P. (2017). Equivalence between policy gradients and soft Q-Learning. ArXiv:1704.06440.

Schultz, D. G., Melsa, J. L. (1967). *State Functions and Linear Control Systems.* McGraw-Hill, New York.

Schultz, W. (1998). Predictive reward signal of dopamine neurons. *Journal of Neurophysiology, 80*(1):1–27.

Schultz, W., Apicella, P., Ljungberg, T. (1993). Responses of monkey dopamine neurons to reward and conditioned stimuli during successive steps of learning a delayed response task. *The Journal of Neuroscience, 13*(3):900–913.

Schultz, W., Dayan, P., Montague, P. R. (1997). A neural substrate of prediction and reward. *Science, 275*(5306):1593–1598.

Schultz, W., Romo, R. (1990). Dopamine neurons of the monkey midbrain: contingencies of responses to stimuli eliciting immediate behavioral reactions. *Journal of Neurophysiology, 63*(3):607–624.

Schultz, W., Romo, R., Ljungberg, T., Mirenowicz, J., Hollerman, J. R., Dickinson, A. (1995). Reward-related signals carried by dopamine neurons. In J. C. Houk, J. L. Davis, and D. G. Beiser (Eds.), *Models of Information Processing in the Basal Ganglia*, pp. 233–248. MIT Press, Cambridge, MA.

Schwartz, A. (1993). A reinforcement learning method for maximizing undiscounted rewards. In *Proceedings of the 10th International Conference on Machine Learning*, pp. 298–305. Morgan Kaufmann.

Schweitzer, P. J., Seidmann, A. (1985). Generalized polynomial approximations in Markovian decision processes. *Journal of Mathematical Analysis and Applications, 110*(2):568–582.

Selfridge, O. G. (1978). Tracking and trailing: Adaptation in movement strategies. Technical report, Bolt Beranek and Newman, Inc. Unpublished report.

Selfridge, O. G. (1984). Some themes and primitives in ill-defined systems. In O. G. Selfridge, E. L. Rissland, and M. A. Arbib (Eds.), *Adaptive Control of Ill-Defined Systems*, pp. 21–26. Plenum Press, NY. Proceedings of the NATO Advanced Research Institute on Adaptive Control of Ill-defined Systems, NATO Conference Series II, Systems Science, Vol. 16.

Selfridge, O. J., Sutton, R. S., Barto, A. G. (1985). Training and tracking in robotics. In A. Joshi (Ed.), *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pp. 670–672. Morgan Kaufmann.

Seo, H., Barraclough, D., Lee, D. (2007). Dynamic signals related to choices and outcomes in the dorsolateral prefrontal cortex. *Cerebral Cortex, 17*(suppl 1):110–117.

Seung, H. S. (2003). Learning in spiking neural networks by reinforcement of stochastic synaptic transmission. *Neuron, 40*(6):1063–1073.

Shah, A. (2012). Psychological and neuroscientific connections with reinforcement learning. In M. Wiering and M. van Otterlo (Eds.), *Reinforcement Learning: State-of-the-Art*, pp. 507–537. Springer-Verlag Berlin Heidelberg.

Shannon, C. E. (1950). Programming a computer for playing chess. *Philosophical Magazine and Journal of Science, 41*(314):256–275.

Shannon, C. E. (1951). Presentation of a maze-solving machine. In H. V. Forester (Ed.), *Cybernetics. Transactions of the Eighth Conference*, pp. 173–180. Josiah Macy Jr. Foundation.

Shannon, C. E. (1952). "Theseus" maze-solving mouse. http://cyberneticzoo.com/mazesolvers/1952—theseus-maze-solving-mouse—claude-shannon-american/.

Shelton, C. R. (2001). *Importance Sampling for Reinforcement Learning with Multiple Objectives.* PhD thesis, Massachusetts Institute of Technology, Cambridge MA.

Shepard, D. (1968). A two-dimensional interpolation function for irregularly-spaced data. In *Proceedings of the 23rd ACM National Conference*, pp. 517–524. ACM, New York.

Sherman, J., Morrison, W. J. (1949). Adjustment of an inverse matrix corresponding to changes in the elements of a given column or a given row of the original matrix (abstract). *Annals of Mathematical Statistics, 20*(4):621.

Shewchuk, J., Dean, T. (1990). Towards learning time-varying functions with high input dimensionality. In *Proceedings of the Fifth IEEE International Symposium on Intelligent Control*, pp. 383–388. IEEE Computer Society Press, Los Alamitos, CA.

Shimansky, Y. P. (2009). Biologically plausible learning in neural networks: a lesson from bacterial chemotaxis. *Biological Cybernetics, 101*(5-6):379–385.

Si, J., Barto, A., Powell, W., Wunsch, D. (Eds.) (2004). *Handbook of Learning and Approximate Dynamic Programming.* John Wiley and Sons.

Silver, D. (2009). *Reinforcement Learning and Simulation Based Search in the Game of Go.* PhD thesis, University of Alberta, Edmonton.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature, 529*(7587):484–489.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., Riedmiller, M. (2014). Deterministic policy gradient algorithms. In *Proceedings of the 31st International Conference on Machine Learning*, pp. 387–395.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, L., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., Hassibis, D. (2017a). Mastering the game of Go without human knowledge. *Nature, 550*(7676):354–359.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simoyan, K., Hassibis, D. (2017b). Mastering chess and shogi by self-play with a general reinforcement learning algorithm. ArXiv:1712.01815.

Şimşek, Ö., Algórta, S., Kothiyal, A. (2016). Why most decisions are easy in tetris—And perhaps in other sequential decision problems, as well. In *Proceedings of the 33rd International Conference on Machine Learning*, pp. 1757-1765.

Simon, H. (2000). Lecture at the Earthware Symposium, Carnegie Mellon University. https://www.youtube.com/watch?v=EZhyi-8DBjc.

Singh, S. P. (1992a). Reinforcement learning with a hierarchy of abstract models. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pp. 202–207. AAAI/MIT Press, Menlo Park, CA.

Singh, S. P. (1992b). Scaling reinforcement learning algorithms by learning variable temporal resolution models. In *Proceedings of the 9th International Workshop on Machine Learning*, pp. 406–415. Morgan Kaufmann.

Singh, S. P. (1993). *Learning to Solve Markovian Decision Processes.* PhD thesis, University of Massachusetts, Amherst.

Singh, S. P. (Ed.) (2002). Special double issue on reinforcement learning, *Machine Learning, 49*(2-3).

Singh, S., Barto, A. G., Chentanez, N. (2005). Intrinsically motivated reinforcement learning. In *Advances in Neural Information Processing Systems 17*, pp. 1281–1288. MIT Press, Cambridge, MA.

Singh, S. P., Bertsekas, D. (1997). Reinforcement learning for dynamic channel allocation in cellular telephone systems. In *Advances in Neural Information Processing Systems 9*, pp. 974–980. MIT Press, Cambridge, MA.

Singh, S. P., Jaakkola, T., Jordan, M. I. (1994). Learning without state-estimation in partially observable Markovian decision problems. In *Proceedings of the 11th International Conference on Machine Learning*, pp. 284–292. Morgan Kaufmann.

Singh, S., Jaakkola, T., Littman, M. L., Szepesvári, C. (2000). Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning, 38*(3):287–308.

Singh, S. P., Jaakkola, T., Jordan, M. I. (1995). Reinforcement learning with soft state aggregation. In *Advances in Neural Information Processing Systems 7*, pp. 359–368. MIT Press, Cambridge, MA.

Singh, S., Lewis, R. L., Barto, A. G. (2009). Where do rewards come from? In N. Taatgen and H. van Rijn (Eds.), *Proceedings of the 31st Annual Conference of the Cognitive Science Society*, pp. 2601–2606. Cognitive Science Society.

Singh, S., Lewis, R. L., Barto, A. G., Sorg, J. (2010). Intrinsically motivated reinforcement learning: An evolutionary perspective. *IEEE Transactions on Autonomous Mental Development, 2*(2):70–82. Special issue on Active Learning and Intrinsically Motivated Exploration in Robots: Advances and Challenges.

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

Skinner, B. F. (1938). *The Behavior of Organisms: An Experimental Analysis.* Appleton-Century, New York.

Skinner, B. F. (1958). Reinforcement today. *American Psychologist, 13*(3):94–99.

Skinner, B. F. (1963). Operant behavior. *American Psychologist, 18*(8):503–515.

Sofge, D. A., White, D. A. (1992). Applied learning: Optimal control for manufacturing. In D. A. White and D. A. Sofge (Eds.), *Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches*, pp. 259–281. Van Nostrand Reinhold, New York.

Sorg, J. D. (2011). *The Optimal Reward Problem:Designing Effective Reward for Bounded Agents.* PhD thesis, University of Michigan, Ann Arbor.

Sorg, J., Lewis, R. L., Singh, S. P. (2010). Reward design via online gradient ascent. In *Advances in Neural Information Processing Systems 23*, pp. 2190–2198. Curran Associates, Inc.

Sorg, J., Singh, S. (2010). Linear options. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems*, pp. 31–38.

Sorg, J., Singh, S., Lewis, R. (2010). Internal rewards mitigate agent boundedness. In *Proceedings of the 27th International Conference on Machine Learning*, pp. 1007–1014.

Spence, K. W. (1947). The role of secondary reinforcement in delayed reward learning. *Psychological Review, 54*(1):1–8.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research, 15*(1):1929–1958.

Staddon, J. E. R. (1983). *Adaptive Behavior and Learning.* Cambridge University Press.

Stanfill, C., Waltz, D. (1986). Toward memory-based reasoning. *Communications of the ACM, 29*(12):1213–1228.

Steinberg, E. E., Keiflin, R., Boivin, J. R., Witten, I. B., Deisseroth, K., Janak, P. H. (2013). A causal link between prediction errors, dopamine neurons and learning. *Nature Neuroscience, 16*(7):966–973.

Sterling, P., Laughlin, S. (2015). *Principles of Neural Design.* MIT Press, Cambridge, MA.

Sternberg, S. (1963). Stochastic learning theory. In: Handbook of Mathematical Psychology, Volume II, R. D. Luce, R. R. Bush, and E. Galanter (Eds.). John Wiley & Sons.

Sugiyama, M., Hachiya, H., Morimura, T. (2013). *Statistical Reinforcement Learning: Modern Machine Learning Approaches.* Chapman & Hall/CRC.

Suri, R. E., Bargas, J., Arbib, M. A. (2001). Modeling functions of striatal dopamine modulation in learning and planning. *Neuroscience, 103*(1):65–85.

Suri, R. E., Schultz, W. (1998). Learning of sequential movements by neural network model with dopamine-like reinforcement signal. *Experimental Brain Research, 121*(3):350–354.

Suri, R. E., Schultz, W. (1999). A neural network model with dopamine-like reinforcement signal that learns a spatial delayed response task. *Neuroscience, 91*(3):871–890.

Sutton, R. S. (1978a). Learning theory support for a single channel theory of the brain. Unpublished report.

Sutton, R. S. (1978b). Single channel theory: A neuronal theory of learning. *Brain Theory Newsletter, 4*:72–75. Center for Systems Neuroscience, University of Massachusetts, Amherst, MA.

Sutton, R. S. (1978c). *A unified theory of expectation in classical and instrumental conditioning.* Bachelors thesis, Stanford University.

Sutton, R. S. (1984). *Temporal Credit Assignment in Reinforcement Learning.* PhD thesis, University of Massachusetts, Amherst.

Sutton, R. S. (1988). Learning to predict by the method of temporal differences. *Machine Learning, 3*(1):9–44 (important erratum p. 377).

Sutton, R. S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the 7th International Workshop on Machine Learning*, pp. 216–224. Morgan Kaufmann.

Sutton, R. S. (1991a). Dyna, an integrated architecture for learning, planning, and reacting. *SIGART Bulletin, 2*(4):160–163. ACM, New York.

Sutton, R. S. (1991b). Planning by incremental dynamic programming. In *Proceedings of the 8th International Workshop on Machine Learning*, pp. 353–357. Morgan Kaufmann.

Sutton, R. S. (Ed.) (1992a). *Reinforcement Learning.* Kluwer Academic Press. Reprinting of a special double issue on reinforcement learning, *Machine Learning, 8*(3-4).

Sutton, R. S. (1992b). Adapting bias by gradient descent: An incremental version of delta-bar-delta. *Proceedings of the Tenth National Conference on Artificial Intelligence*, pp. 171–176, MIT Press.

Sutton, R. S. (1992c). Gain adaptation beats least squares? *Proceedings of the Seventh Yale Workshop on Adaptive and Learning Systems*, pp. 161–166, Yale University, New Haven, CT.

Sutton, R. S. (1995a). TD models: Modeling the world at a mixture of time scales. In *Proceedings of the 12th International Conference on Machine Learning*, pp. 531–539. Morgan Kaufmann.

Sutton, R. S. (1995b). On the virtues of linear learning and trajectory distributions. In *Proceedings of the Workshop on Value Function Approximation* at *The 12th International Conference on Machine Learning.*

Sutton, R. S. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems 8*, pp. 1038–1044. MIT Press, Cambridge, MA.

Sutton, R. S. (2009). The grand challenge of predictive empirical abstract knowledge. *Working Notes of the IJCAI-09 Workshop on Grand Challenges for Reasoning from Experiences.*

Sutton, R. S. (2015a) Introduction to reinforcement learning with function approximation. Tutorial at the Conference on Neural Information Processing Systems, Montreal, December 7, 2015.

Sutton, R. S. (2015b) True online Emphatic TD(λ): Quick reference and implementation guide. ArXiv:1507.07147. Code is available in Python and C++ by downloading the source files of this arXiv paper as a zip archive.

Sutton, R. S., Barto, A. G. (1981a). Toward a modern theory of adaptive networks: Expectation and prediction. *Psychological Review, 88*(2):135–170.

Sutton, R. S., Barto, A. G. (1981b). An adaptive network that constructs and uses an internal model of its world. *Cognition and Brain Theory, 3*:217–246.

Sutton, R. S., Barto, A. G. (1987). A temporal-difference model of classical conditioning. In *Proceedings of the Ninth Annual Conference of the Cognitive Science Society*, pp. 355-378. Erlbaum, Hillsdale, NJ.

Sutton, R. S., Barto, A. G. (1990). Time-derivative models of Pavlovian reinforcement. In M. Gabriel and J. Moore (Eds.), *Learning and Computational Neuroscience: Foundations of Adaptive Networks*, pp. 497–537. MIT Press, Cambridge, MA.

Sutton, R. S., Maei, H. R., Precup, D., Bhatnagar, S., Silver, D., Szepesvári, Cs., Wiewiora, E. (2009a). Fast gradient-descent methods for temporal-difference learning with linear function approximation. In *Proceedings of the 26th International Conference on Machine Learning*, pp. 993–1000. ACM, New York.

Sutton, R. S., Szepesvári, Cs., Maei, H. R. (2009b). A convergent $O(d^2)$ temporal-difference algorithm for off-policy learning with linear function approximation. In *Advances in Neural Information Processing Systems 21*, pp. 1609–1616. Curran Associates, Inc.

Sutton, R. S., Mahmood, A. R., Precup, D., van Hasselt, H. (2014). A new Q(λ) with interim forward view and Monte Carlo equivalence. In *Proceedings of the International Conference on Machine Learning, 31. JMLR W&CP 32*(2).

Sutton, R. S., Mahmood, A. R., White, M. (2016). An emphatic approach to the problem of off-policy temporal-difference learning. *Journal of Machine Learning Research, 17*(73):1–29.

Sutton, R. S., McAllester, D. A., Singh, S. P., Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems 12*, pp. 1057–1063. MIT Press, Cambridge, MA.

Sutton, R. S., Modayil, J., Delp, M., Degris, T., Pilarski, P. M., White, A., Precup, D. (2011). Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *Proceedings of the Tenth International Conference on Autonomous Agents and Multiagent Systems*, pp. 761–768, Taipei, Taiwan.

Sutton, R. S., Pinette, B. (1985). The learning of world models by connectionist networks. In *Proceedings of the Seventh Annual Conference of the Cognitive Science Society*, pp. 54–64.

Sutton, R. S., Precup, D., Singh, S. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence, 112*(1-2):181–211.

Sutton, R. S., Rafols, E., Koop, A. (2006). Temporal abstraction in temporal-difference networks. In *Advances in neural information processing systems*, pp. 1313–1320.

Sutton, R. S., Singh, S. P., McAllester, D. A. (2000). Comparing policy-gradient algorithms. Unpublished manuscript.

Sutton, R. S., Szepesvári, Cs., Geramifard, A., Bowling, M., (2008). Dyna-style planning with linear function approximation and prioritized sweeping. In *Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence*, pp. 528–536.

Sutton, R. S., Tanner, B. (2005). Temporal-difference networks. In *Advances in Neural Information Processing Systems 17*, p. 1377–1384.

Szepesvári, Cs. (2010). Algorithms for reinforcement learning. In *Synthesis Lectures on Artificial Intelligence and Machine Learning, 4*(1):1–103. Morgan and Claypool.

Szita, I. (2012). Reinforcement learning in games. In M. Wiering and M. van Otterlo (Eds.), *Reinforcement Learning: State-of-the-Art*, pp. 539–577. Springer-Verlag Berlin Heidelberg.

Tadepalli, P., Ok, D. (1994). H-learning: A reinforcement learning method to optimize undiscounted average reward. Technical Report 94-30-01. Oregon State University, Computer Science Department, Corvallis.

Tadepalli, P., Ok, D. (1996). Scaling up average reward reinforcement learning by approximating the domain models and the value function. In *Proceedings of the 13th International Conference on Machine Learning*, pp. 471–479.

Takahashi, Y., Schoenbaum, G., and Niv, Y. (2008). Silencing the critics: Understanding the effects of cocaine sensitization on dorsolateral and ventral striatum in the context of an actor/critic model. *Frontiers in Neuroscience, 2*(1):86–99.

Tambe, M., Newell, A., Rosenbloom, P. S. (1990). The problem of expensive chunks and its solution by restricting expressiveness. *Machine Learning, 5*(3):299–348.

Tan, M. (1991). Learning a cost-sensitive internal representation for reinforcement learning. In L. A. Birnbaum and G. C. Collins (Eds.), *Proceedings of the 8th International Workshop on Machine Learning*, pp. 358–362. Morgan Kaufmann.

Tanner, B. (2006). Temporal-Difference Networks. MSc thesis, University of Alberta.

Taylor, G., Parr, R. (2009). Kernelized value function approximation for reinforcement learning. In *Proceedings of the 26th International Conference on Machine Learning*, pp. 1017–1024. ACM, New York.

Taylor, M. E., Stone, P. (2009). Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research, 10*:1633–1685.

Tesauro, G. (1986). Simple neural models of classical conditioning. *Biological Cybernetics, 55*(2-3):187–200.

Tesauro, G. (1992). Practical issues in temporal difference learning. *Machine Learning, 8*(3-4):257–277.

Tesauro, G. (1994). TD-Gammon, a self-teaching backgammon program, achieves master-level play. *Neural Computation, 6*(2):215–219.

Tesauro, G. (1995). Temporal difference learning and TD-Gammon. *Communications of the ACM, 38*(3):58–68.

Tesauro, G. (2002). Programming backgammon using self-teaching neural nets. *Artificial Intelligence, 134*(1-2):181–199.

Tesauro, G., Galperin, G. R. (1997). On-line policy improvement using Monte-Carlo search. In *Advances in Neural Information Processing Systems 9*, pp. 1068–1074. MIT Press, Cambridge, MA.

Tesauro, G., Gondek, D. C., Lechner, J., Fan, J., Prager, J. M. (2012). Simulation, learning, and optimization techniques in Watson's game strategies. *IBM Journal of Research and Development, 56*(3-4):16–1–16–11.

Tesauro, G., Gondek, D. C., Lenchner, J., Fan, J., Prager, J. M. (2013). Analysis of Watson's strategies for playing Jeopardy! *Journal of Artificial Intelligence Research, 47*:205–251.

Tham, C. K. (1994). *Modular On-Line Function Approximation for Scaling up Reinforcement Learning.* PhD thesis, University of Cambridge.

Thathachar, M. A. L., Sastry, P. S. (1985). A new approach to the design of reinforcement schemes for learning automata. *IEEE Transactions on Systems, Man, and Cybernetics, 15*(1):168–175.

Thathachar, M., Sastry, P. S. (2002). Varieties of learning automata: an overview. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, 36*(6):711–722.

Thathachar, M., Sastry, P. S. (2011). *Networks of Learning Automata: Techniques for Online Stochastic Optimization.* Springer Science & Business Media.

Theocharous, G., Thomas, P. S., Ghavamzadeh, M. (2015). Personalized ad recommendation for life-time value optimization guarantees. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence.* AAAI Press, Palo Alto, CA.

Thistlethwaite, D. (1951). A critical review of latent learning and related experiments. *Psychological Bulletin, 48*(2):97–129.

Thomas, P. S. (2014). Bias in natural actor–critic algorithms. In *Proceedings of the 31st International Conference on Machine Learning, JMLR W&CP 32*(1), pp. 441–448.

Thomas, P. S. (2015). *Safe Reinforcement Learning.* PhD thesis, University of Massachusetts, Amherst.

Thomas, P. S., Brunskill, E. (2017). Policy gradient methods for reinforcement learning with function approximation and action-dependent baselines. ArXiv:1706.06643.

Thomas, P. S., Theocharous, G., Ghavamzadeh, M. (2015). High-confidence off-policy evaluation. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, pp. 3000–3006. AAAI Press, Menlo Park, CA.

Thompson, W. R. (1933). On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika, 25*(3-4):285–294.

Thompson, W. R. (1934). On the theory of apportionment. *American Journal of Mathematics, 57*: 450–457.

Thon, M. (2017). *Spectral Learning of Sequential Systems.* PhD thesis, Jacobs University Bremen.

Thon, M., Jaeger, H. (2015). Links between multiplicity automata, observable operator models and predictive state representations: a unified learning framework. *The Journal of Machine Learning Research, 16*(1):103–147.

Thorndike, E. L. (1898). Animal intelligence: An experimental study of the associative processes in animals. *The Psychological Review, Series of Monograph Supplements*, II(4).

Thorndike, E. L. (1911). *Animal Intelligence.* Hafner, Darien, CT.

Thorp, E. O. (1966). *Beat the Dealer: A Winning Strategy for the Game of Twenty-One.* Random House, New York.

Tian, T. (in preparation) *An Empirical Study of Sliding-Step Methods in Temporal Difference Learning.* M.Sc thesis, University of Alberta, Edmonton.

Tieleman, T., Hinton, G. (2012). Lecture 6.5–RMSProp. COURSERA: Neural networks for machine learning 4.2:26–31.

Tolman, E. C. (1932). *Purposive Behavior in Animals and Men.* Century, New York.

Tolman, E. C. (1948). Cognitive maps in rats and men. *Psychological Review, 55*(4):189–208.

Tsai, H.-S., Zhang, F., Adamantidis, A., Stuber, G. D., Bonci, A., de Lecea, L., Deisseroth, K. (2009). Phasic firing in dopaminergic neurons is sufficient for behavioral conditioning. *Science, 324*(5930):1080–1084.

Tsetlin, M. L. (1973). *Automaton Theory and Modeling of Biological Systems.* Academic Press, New York.

Tsitsiklis, J. N. (1994). Asynchronous stochastic approximation and Q-learning. *Machine Learning, 16*(3):185–202.

Tsitsiklis, J. N. (2002). On the convergence of optimistic policy iteration. *Journal of Machine Learning Research, 3*:59–72.

Tsitsiklis, J. N., Van Roy, B. (1996). Feature-based methods for large scale dynamic programming. *Machine Learning, 22*(1-3):59–94.

Tsitsiklis, J. N., Van Roy, B. (1997). An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control, 42*(5):674–690.

Tsitsiklis, J. N., Van Roy, B. (1999). Average cost temporal-difference learning. *Automatica, 35*(11):1799–1808.

Turing, A. M. (1948). Intelligent machinery. In B. Jack Copeland (Ed.) (2004), *The Essential Turing*, pp. 410–432. Oxford University Press, Oxford.

Ungar, L. H. (1990). A bioreactor benchmark for adaptive network-based process control. In W. T. Miller, R. S. Sutton, and P. J. Werbos (Eds.), *Neural Networks for Control*, pp. 387–402. MIT Press, Cambridge, MA.

Unnikrishnan, K. P., Venugopal, K. P. (1994). Alopex: A correlation-based learning algorithm for feedforward and recurrent neural networks. *N*eural Computation, 6(3): 469–490.

Urbanczik, R., Senn, W. (2009). Reinforcement learning in populations of spiking neurons. *Nature neuroscience, 12*(3):250–252.

Urbanowicz, R. J., Moore, J. H. (2009). Learning classifier systems: A complete introduction, review, and roadmap. *Journal of Artificial Evolution and Applications*. 10.1155/2009/736398.

Valentin, V. V., Dickinson, A., O'Doherty, J. P. (2007). Determining the neural substrates of goal-directed learning in the human brain. *The Journal of Neuroscience, 27*(15):4019–4026.

van Hasselt, H. (2010). Double Q-learning. In *Advances in Neural Information Processing Systems 23*, pp. 2613–2621. Curran Associates, Inc.

van Hasselt, H. (2011). *Insights in Reinforcement Learning: Formal Analysis and Empirical Evaluation of Temporal-difference Learning*. SIKS dissertation series number 2011-04.

van Hasselt, H. (2012). Reinforcement learning in continuous state and action spaces. In M. Wiering and M. van Otterlo (Eds.), *Reinforcement Learning: State-of-the-Art*, pp. 207–251. Springer-Verlag Berlin Heidelberg.

van Hasselt, H., Sutton, R. S. (2015). Learning to predict independent of span. ArXiv:1508.04582.

Van Roy, B., Bertsekas, D. P., Lee, Y., Tsitsiklis, J. N. (1997). A neuro-dynamic programming approach to retailer inventory management. In *Proceedings of the 36th IEEE Conference on Decision and Control*, Vol. 4, pp. 4052–4057.

van Seijen, H. (2011). Reinforcement Learning under Space and Time Constraints. University of Amsterdam PhD thesis. Hague: TNO.

van Seijen, H. (2016). Effective multi-step temporal-difference learning for non-linear function approximation. ArXiv:1608.05151.

van Seijen, H., Sutton, R. S. (2013). Efficient planning in MDPs by small backups. In: *Proceedings of the 30th International Conference on Machine Learning*, pp. 361–369.

van Seijen, H., Sutton, R. S. (2014). True online TD(λ). In *Proceedings of the 31st International Conference on Machine Learning*, pp. 692–700. JMLR W&CP 32(1),

van Seijen, H., Mahmood, A. R., Pilarski, P. M., Machado, M. C., Sutton, R. S. (2016). True online temporal-difference learning. *Journal of Machine Learning Research, 17*(145):1–40.

van Seijen, H., Van Hasselt, H., Whiteson, S., Wiering, M. (2009). A theoretical and empirical analysis of Expected Sarsa. In *IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, pp. 177–184.

van Seijen, H., Whiteson, S., van Hasselt, H., Wiering, M. (2011). Exploiting best-match equations for efficient reinforcement learning. *Journal of Machine Learning Research 12*:2045–2094.

Varga, R. S. (1962). *Matrix Iterative Analysis.* Englewood Cliffs, NJ: Prentice-Hall.

Vasilaki, E., Frémaux, N., Urbanczik, R., Senn, W., Gerstner, W. (2009). Spike-based reinforcement learning in continuous state and action space: when policy gradient methods fail. *PLoS Computational Biology, 5*(12).

Viswanathan, R., Narendra, K. S. (1974). Games of stochastic automata. *IEEE Transactions on Systems, Man, and Cybernetics, 4*(1):131–135.

Wagner, A. R. (2008). Evolution of an elemental theory of Pavlovian conditioning. *Learning & Behavior, 36*(3):253–265.

Walter, W. G. (1950). An imitation of life. *Scientific American, 182*(5):42–45.

Walter, W. G. (1951). A machine that learns. *Scientific American, 185*(2):60–63.

Waltz, M. D., Fu, K. S. (1965). A heuristic approach to reinforcement learning control systems. *IEEE Transactions on Automatic Control, 10*(4):390–398.

Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards.* PhD thesis, University of Cambridge.

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

Werbos, P. J. (1977). Advanced forecasting methods for global crisis warning and models of intelligence. *General Systems Yearbook, 22*(12):25–38.

Werbos, P. J. (1982). Applications of advances in nonlinear sensitivity analysis. In R. F. Drenick and F. Kozin (Eds.), *System Modeling and Optimization*, pp. 762–770. Springer-Verlag.

Werbos, P. J. (1987). Building and understanding adaptive systems: A statistical/numerical approach to factory automation and brain research. *IEEE Transactions on Systems, Man, and Cybernetics, 17*(1):7–20.

Werbos, P. J. (1988). Generalization of back propagation with applications to a recurrent gas market model. *Neural Networks, 1*(4):339–356.

Werbos, P. J. (1989). Neural networks for control and system identification. In *Proceedings of the 28th Conference on Decision and Control*, pp. 260–265. IEEE Control Systems Society.

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

Werbos, P. J. (1994). *The Roots of Backpropagation: From Ordered Derivatives to Neural Networks and Political Forecasting* (Vol. 1). John Wiley and Sons.

Wiering, M., Van Otterlo, M. (2012). *Reinforcement Learning: State-of-the-Art.* Springer-Verlag Berlin Heidelberg.

White, A. (2015). *Developing a Predictive Approach to Knowledge.* PhD thesis, University of Alberta, Edmonton.

White, D. J. (1969). *Dynamic Programming.* Holden-Day, San Francisco.

White, D. J. (1985). Real applications of Markov decision processes. *Interfaces, 15*(6):73–83.

White, D. J. (1988). Further real applications of Markov decision processes. *Interfaces, 18*(5):55–61.

White, D. J. (1993). A survey of applications of Markov decision processes. *Journal of the Operational Research Society, 44*(11):1073–1096.

White, A., White, M. (2016). Investigating practical linear temporal difference learning. In *Proceedings of the 2016 International Conference on Autonomous Agents and Multiagent Systems*, pp. 494–502.

Whitehead, S. D., Ballard, D. H. (1991). Learning to perceive and act by trial and error. *Machine Learning, 7*(1):45–83.

Whitt, W. (1978). Approximations of dynamic programs I. *Mathematics of Operations Research, 3*(3):231–243.

Whittle, P. (1982). *Optimization over Time*, vol. 1. Wiley, New York.

Whittle, P. (1983). *Optimization over Time*, vol. 2. Wiley, New York.

Wickens, J., Kötter, R. (1995). Cellular models of reinforcement. In J. C. Houk, J. L. Davis and D. G. Beiser (Eds.), *Models of Information Processing in the Basal Ganglia*, pp. 187–214. MIT Press, Cambridge, MA.

Widrow, B., Gupta, N. K., Maitra, S. (1973). Punish/reward: Learning with a critic in adaptive threshold systems. *IEEE Transactions on Systems, Man, and Cybernetics, 3*(5):455–465.

Widrow, B., Hoff, M. E. (1960). Adaptive switching circuits. In *1960 WESCON Convention Record Part IV*, pp. 96–104. Institute of Radio Engineers, New York. Reprinted in J. A. Anderson and E. Rosenfeld, *Neurocomputing: Foundations of Research*, pp. 126–134. MIT Press, Cambridge, MA, 1988.

Widrow, B., Smith, F. W. (1964). Pattern-recognizing control systems. In J. T. Tou and R. H. Wilcox (Eds.), *Computer and Information Sciences*, pp. 288–317. Spartan, Washington, DC.

Widrow, B., Stearns, S. D. (1985). *Adaptive Signal Processing.* Prentice-Hall, Englewood Cliffs, NJ.

Wiener, N. (1964). *God and Golem, Inc: A Comment on Certain Points where Cybernetics Impinges on Religion.* MIT Press, Cambridge, MA.

Wiewiora, E. (2003). Potential-based shaping and Q-value initialization are equivalent. *Journal of Artificial Intelligence Research, 19*:205–208.

Williams, R. J. (1986). Reinforcement learning in connectionist networks: A mathematical analysis. Technical Report ICS 8605. Institute for Cognitive Science, University of California at San Diego, La Jolla.

Williams, R. J. (1987). Reinforcement-learning connectionist systems. Technical Report NU-CCS-87-3. College of Computer Science, Northeastern University, Boston.

Williams, R. J. (1988). On the use of backpropagation in associative reinforcement learning. In *Proceedings of the IEEE International Conference on Neural Networks*, pp. I-263–I-270. IEEE San Diego section and IEEE TAB Neural Network Committee.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning, 8*(3-4):229–256.

Williams, R. J., Baird, L. C. (1990). A mathematical analysis of actor–critic architectures for learning optimal controls through incremental dynamic programming. In *Proceedings of the Sixth Yale Workshop on Adaptive and Learning Systems*, pp. 96–101. Center for Systems Science, Dunham Laboratory, Yale University, New Haven.

Wilson, R. C., Takahashi, Y. K., Schoenbaum, G., Niv, Y. (2014). Orbitofrontal cortex as a cognitive map of task space. *Neuron, 81*(2):267–279.

Wilson, S. W. (1994). ZCS, A zeroth order classifier system. *Evolutionary Computation, 2*(1):1–18.

Wise, R. A. (2004). Dopamine, learning, and motivation. *Nature Reviews Neuroscience, 5*(6):1–12.

Witten, I. H. (1976a). Learning to Control. University of Essex PhD thesis.

Witten, I. H. (1976b). The apparent conflict between estimation and control—A survey of the two-armed problem. *Journal of the Franklin Institute, 301*(1-2):161–189.

Witten, I. H. (1977). An adaptive optimal controller for discrete-time Markov environments. *Information and Control, 34*(4):286–295.

Witten, I. H., Corbin, M. J. (1973). Human operators and automatic adaptive controllers: A comparative study on a particular control task. *International Journal of Man–Machine Studies, 5*(1):75–104.

Woodbury, T., Dunn, C., and Valasek, J. (2014). Autonomous soaring using reinforcement learning for trajectory generation. In *52nd Aerospace Sciences Meeting*, p. 0990.

Woodworth, R. S. (1938). *Experimental Psychology.* New York: Henry Holt and Company.

Xie, X., Seung, H. S. (2004). Learning in neural networks by reinforcement of irregular spiking. *Physical Review E, 69*(4):041909.

Xu, X., Xie, T., Hu, D., Lu, X. (2005). Kernel least-squares temporal difference learning. *International Journal of Information Technology, 11*(9):54–63.

Yagishita, S., Hayashi-Takagi, A., Ellis-Davies, G. C. R., Urakubo, H., Ishii, S., Kasai, H. (2014). A critical time window for dopamine actions on the structural plasticity of dendritic spines. *Science, 345*(6204):1616–1619.

Yee, R. C., Saxena, S., Utgoff, P. E., Barto, A. G. (1990). Explaining temporal differences to create useful concepts for evaluating states. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pp. 882–888. AAAI Press, Menlo Park, CA.

Yin, H. H., Knowlton, B. J. (2006). The role of the basal ganglia in habit formation. *Nature Reviews Neuroscience, 7*(6):464–476.

Young, P. (1984). *Recursive Estimation and Time-Series Analysis.* Springer-Verlag, Berlin.

Yu, H. (2010). Convergence of least squares temporal difference methods under general conditions. *International Conference on Machine Learning 27*, pp. 1207–1214.

Yu, H. (2012). Least squares temporal difference methods: An analysis under general conditions. *SIAM Journal on Control and Optimization, 50*(6):3310–3343.

Yu, H. (2015). On convergence of emphatic temporal-difference learning. In *Proceedings of the 28th Annual Conference on Learning Theory, JMLR W&CP 40.* Also ArXiv:1506.02582.

Yu, H. (2016). Weak convergence properties of constrained emphatic temporal-difference learning with constant and slowly diminishing stepsize. *Journal of Machine Learning Research, 17*(220):1–58.

Yu, H. (2017). On convergence of some gradient-based temporal-differences algorithms for off-policy learning. ArXiv:1712.09652.

Yu, H., Mahmood, A. R., Sutton, R. S. (2017). On generalized bellman equations and temporal-difference learning. ArXiv:17041.04463. A summary appeared in *Proceedings of the Canadian Conference on Artificial Intelligence*, pp. 3–14. Springer.

Index

Page numbers in *italics* are recommended to be consulted first. Page numbers in **bold** contain boxed algorithms.

Adaptive Computation and Machine Learning

Francis Bach, Editor

Bioinformatics: The Machine Learning Approach, Pierre Baldi and Søren Brunak

Reinforcement Learning: An Introduction, Richard S. Sutton and Andrew G. Barto

Graphical Models for Machine Learning and Digital Communication, Brendan J. Frey

Learning in Graphical Models, Michael I. Jordan

Causation, Prediction, and Search, second edition, Peter Spirtes, Clark Glymour, and Richard Scheines

Principles of Data Mining, David Hand, Heikki Mannila, and Padhraic Smyth

Bioinformatics: The Machine Learning Approach, second edition, Pierre Baldi and Søren Brunak

Learning Kernel Classifiers: Theory and Algorithms, Ralf Herbrich

Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond, Bernhard Schölkopf and Alexander J. Smola

Introduction to Machine Learning, Ethem Alpaydin

Gaussian Processes for Machine Learning, Carl Edward Rasmussen and Christopher K.I. Williams

Semi-Supervised Learning, Olivier Chapelle, Bernhard Schölkopf, and Alexander Zien, Eds.

The Minimum Description Length Principle, Peter D. Grünwald

Introduction to Statistical Relational Learning, Lise Getoor and Ben Taskar, Eds.

Probabilistic Graphical Models: Principles and Techniques, Daphne Koller and Nir Friedman

Introduction to Machine Learning, second edition, Ethem Alpaydin

Machine Learning in Non-Stationary Environments: Introduction to Covariate Shift Adaptation, Masashi Sugiyama and Motoaki Kawanabe

Boosting: Foundations and Algorithms, Robert E. Schapire and Yoav Freund

Machine Learning: A Probabilistic Perspective, Kevin P. Murphy

Foundations of Machine Learning, Mehryar Mohri, Afshin Rostami, and Ameet Talwalker

Introduction to Machine Learning, third edition, Ethem Alpaydin

Deep Learning, Ian Goodfellow, Yoshua Bengio, and Aaron Courville

Elements of Causal Inference, Jonas Peters, Dominik Janzing, and Bernhard Schölkopf

Machine Learning for Data Streams, with Practical Examples in MOA, Albert Bifet, Ricard Gavaldà, Geoffrey Holmes, Bernhard Pfahringer