Neural-Networks Models of Cognition
J.W. Donahoe and V. Packard Dorsel (Editors)

CHAPTER 19

REINFORCEMENT LEARNING IN ARTIFICIAL INTELLIGENCE

Andrew G. Barto and Richard S. Sutton
Department of Computer Science
University of Massachusetts, Amherst

ABSTRACT

This chapter provides an overview of an approach to the study of learning that, in broad terms, has developed as a part of the field of Artificial Intelligence (AI), where it is called *reinforcement learning* due to its roots in reinforcement theories of animal learning. We introduce the field from the perspective of AI and engineering, describing some of its key features, providing a formal model of the reinforcement-learning problem, and defining basic concepts that are exploited by solution methods. Detailed discussion of solution methods themselves and their history are very broad topics that we do not attempt to cover here.

Introduction

This chapter describes an approach to the study of learning that has developed largely as a part of the field of Artificial Intelligence (AI), where it is called *reinforcement learning* due to its roots in reinforcement theories that arose during the first half of this century. Reinforcement learning in AI consists of a collection of computational methods that, although inspired by animal-learning principles, are primarily motivated by their potential for solving practical problems.

Although the ideas of reinforcement learning have been present in AI since its earliest days (e.g., Minsky, 1954, 1961; Samuel, 1959), several factors limited their influence. Chief among them is that AI research in the 1960s followed the allied areas of psychology in shifting from approaches based in animal behavior toward more cognitive approaches. This shift left little room for reinforcement theories. Although critics have argued convincingly that one cannot understand or generate *all* intelligent behavior on the basis of reinforcement principles alone, reinforcement-learning theorists believe that AI systems and cognitive theories that steer clear of these basic learning principles are handicapped as well.

A related factor that limited the influence of reinforcement-learning principles in AI is the belief that they were too computationally weak to be of much use. However, there is now ample evidence that reinforcement learning can be very powerful. Some of the most impressive accomplishments of artificial learning systems have been achieved using reinforcement learning. For example, Tesauro (1994, 1995) designed a system that used reinforcement learning to learn how to play backgammon at a very strong masters level; Zhang and Dietterich (1995) used reinforcement learning to improve over the state of the art in a job-shop scheduling problem; and Crites and Barto (1996) obtained strong results on the problem of dispatching elevators in a multi-story building with the aim of minimizing a measure of passenger waiting time. These are all very large-scale problems that present formidable difficulties for conventional solution methods.

In addition to these successes, the growing interest in reinforcement learning among current AI researchers is fueled by the challenge of designing intelligent systems that must operate in dynamic real-world environments. For example, making robots, or robotic "agents," more autonomous (that is, less reliant on carefully controlled, fully anticipated conditions) requires decision-making methods that are effective in the presence of uncertainty and that can meet time constraints. Under these conditions, learning seems essential for achieving skilled behavior, and it is under these conditions that reinforcement learning can have significant advantages over other types of learning.

Despite much recent progress in machine learning, including new learning methods for artificial neural networks, most machine-learning research has focused on learning under the tutelage of a knowledgeable "teacher" that can explicitly tell the system how it should respond to a set of training examples. Although supervised learning, or learning from examples, as this type of learning is called, is an important component of more complete systems, it is not by itself adequate for the kind of learning that an autonomous agent must accomplish. It is often very costly, or even impossible, to obtain instructions that are both correct and representative of the situations in which the agent will have to act. In uncharted territory—where one would expect learning to be most beneficial—an agent has to learn from its own experiences rather than from a knowledgeable teacher. The primary source of information and feedback in reinforcement learning is this interaction with an environment. Of course, an agent should also be able to take advantage of the knowledge and experience of other agents to the extent that it can, but it should not subordinate its own intrinsic goals, determined by its definition of what events are intrinsically reinforcing, to the more superficial goal of meeting the specifications set of another, possibly fallible, agent.

Reinforcement learning has developed into an unusually multidisciplinary research area. Researchers from AI, artificial neural networks, robotics, control theory, operations research, and psychology are actively involved. In this chapter we introduce the field largely from the perspective of AI and engineering. We describe some of the key features of reinforcement learning, provide a formal model of the reinforcement-learning problem, and define basic concepts that are exploited by solution methods. Reinforcement-learning methods themselves and their histories are very broad topics that we do not attempt to cover here. The reader should consult Barto (1992); Barto, Bradtke, & Singh (1995); Kaelbling (1993); and Sutton (1992) for some of these details and extensive bibliographies. We also do not discuss how this model of reinforcement learning relates to details of animal-learning theory or to neuroscience. The reader should consult Barto (1992, 1994) for some references to this literature.

Some Key Features

A good way to introduce some of the key features of reinforcement learning is to consider a few of the examples and possible applications that have motivated and guided its development:

(1) A master chess player makes a move. The choice is informed both by planning—anticipating possible replies and counter-replies—and by immediate, intuitive judgments of the desirability of particular positions and moves.

(2) An adaptive controller adjusts parameters of a petroleum refinery's operation in real time. The controller optimizes the yield/cost/quality tradeoff based on specified marginal costs without sticking strictly to the set points originally suggested by human engineers.

(3) Phil prepares his breakfast. When closely examined, even this apparently mundane activity reveals itself as a complex web of conditional behavior and interlocking goal-subgoal relationships: Walking to the cupboard, opening it, selecting a cereal box, then reaching for, grasping, and retrieving it. Other complex, tuned, interactive sequences of behavior are required to obtain a bowl, spoon, and milk jug. Each step involves a series of eye movements to obtain information and to guide reaching and locomotion. Rapid judgments are continually made about how to carry the objects or whether it is better to ferry some of them to the dining table before obtaining others. Each step is guided by goals, such as grasping a spoon, or getting to the refrigerator, and is in service of other goals, such as having the spoon to eat with once the cereal is prepared and of ultimately obtaining nourishment.

(4) A mobile robot decides whether it should enter a new room in search of more trash to collect or start trying to find its way back to its battery-recharging station. It makes its decision based on how quickly and easily it has been able to find the recharger in the past.

These examples share features that are so basic that they are often overlooked. All involve an *interaction* between an active decision-making agent and its environment in which the agent seeks to achieve a *goal* despite variations or *uncertainties* in the environment. The agent's actions are permitted to affect the future state of the environment (e.g., the next chess position, the level of reservoirs of the refinery, the next location of the robot), thereby affecting the options and opportunities available to the agent at later times. Correct choice requires taking into account indirect, delayed consequences of action, and thus may require foresight or planning. At the same time, the effects of actions cannot be fully predicted, so the agent must frequently monitor its environment and react appropriately. These three features—interactivity, uncertainty, and explicit goals—are key features of problems requiring intelligent adaptive behavior. Reinforcement learning is centered on such problems.

Another key feature of the reinforcement-learning approach is that it explicitly considers the *whole* problem of a goal-directed agent interacting with an uncertain environment. This is in contrast to many approaches that address a putative subproblem without addressing how it fits within a larger picture. We have already mentioned, for example, that much of machine-learning research is concerned with supervised learning without explicitly specifying how such an ability would finally be useful. Other AI researchers have developed theories of planning without considering its role in real-time decision making or the question of where the predictive models necessary for planning would come from. Whether or not these approaches are yielding useful results, it is clear that their focus on isolated subproblems has now become an important limitation.

Reinforcement learning takes the opposite tack by starting with a complete, interactive, goal-seeking system. All reinforcement-learning systems have an explicit goal, can sense aspects of their environments, and can choose actions to influence their environments and goals. Goals that involve planning address its interplay with real-time action selection and the question of how environmental models are acquired. Goals that involve supervised learning do so informed by a very specific role that specifies which capabilities and features are critical, and which are not.

Being complete in this sense does not, of course, mean that the reinforcement-learning approach currently fills in all the details, or even suggests how they should be filled in. Reinforcement learning is developing in an *abstract* framework that, while very broad in scope, requires imposing additional structure to address certain kinds of questions. There are many directions in which the reinforcement-learning model we describe here can be profitably specialized and extended.

An Example

The familiar children's game of naughts and crosses (Tic-Tac-Toe) provides a very simple example of reinforcement learning. Two players take turns playing on a three-by-three board. One player plays crosses (X's) and the other naughts (O's) until one player wins by placing three marks in a row—horizontally, vertically, or diagonally—as the "X" player has in **Figure 1**.

If the board fills up with neither player getting three in a row, the game is a draw. Because a skilled player can play so as never to lose, let us assume that we are playing against an imperfect player whose play is sometimes incorrect,

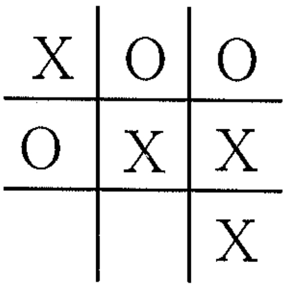


FIGURE 1. Naughts and Crosses (Tic-Tac-Toe). Two players take turns until one of them wins by placing three of his marks (X or O) in a row in any direction.

thereby allowing us to win occasionally. How might one construct a player that will find the imperfections in its opponent's actions and learn to maximize its chances of winning?

Although this is a very simple problem, it cannot readily be solved in a fully satisfactory way by classical techniques. For example, the classical "minimax" solution from game theory is not correct here because it assumes perfect play by the opponent. A minimax solution would never reach a game state from which it could lose, even if in fact it always won from that state because of incorrect play by the opponent. Classical optimization methods for sequential decision problems, such as dynamic programming (e.g., Bertsekas, 1987), can *compute* the optimal solution for any opponent, but require a complete specification of that opponent, including the probabilities with which the opponent would make each move in each board state. We assume this information is not available a priori for this problem, as it is not for the vast majority of problems of practical interest. On the other hand, such information can be estimated from experience, in this case by playing many games against the opponent. About the best one can do with classical methods is to first build from experience a model of the opponent's behavior up to some level of confidence, and then apply dynamic programming to compute an optimal solution given the approximate opponent model. Functionally, this is not that different from some reinforcement-learning methods.

Here is how the naughts-and-crosses problem could be solved most easily using a simple reinforcement-learning approach. First we set up a table of numbers, one for each possible state of the game—i.e., one for each possible configuration of X's and O's on the three-by-three board. Each number provides an estimate of the probability of our winning from that state. Assuming we always play X's, then for all states with three X's in a row the probability of winning is 1, because we have already won. Similarly, for all states with three O's in a row, the correct probability is 0 as we cannot win from them. All the other states, the nonterminals, we set initially to the same value, say 0.5, representing a 50% chance of winning.

Now we play a great many games against the opponent. To select our moves we examine the states that would result from each of our possible moves (one for each blank space on the board) and look up their estimated probabilities of winning. Most of the time we select as our move the one that leads to the state with the highest estimated probability of winning. Occasionally, however, we select randomly from one of the other moves instead; these are called *exploratory* moves because they cause us to experience states that might otherwise never occur. A sequence of moves made and considered during a game can be diagrammed as in **Figure 2**.

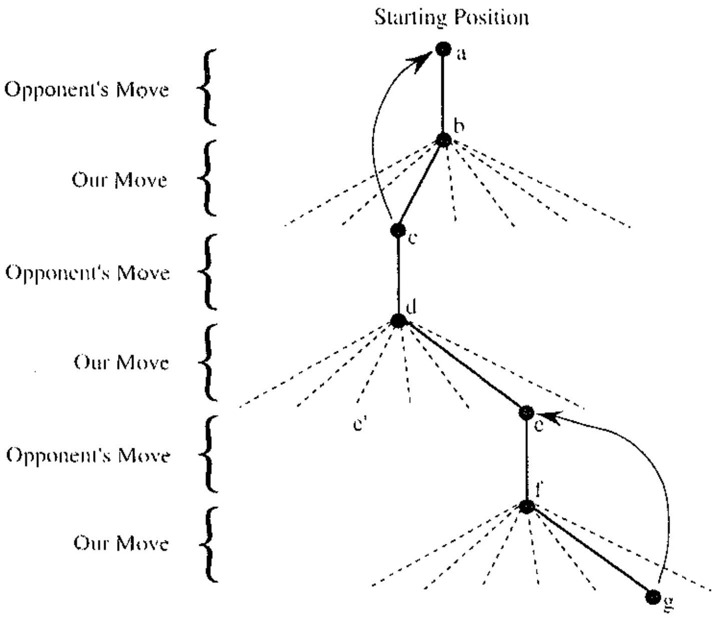


FIGURE 2. Moves in Naughts and Crosses. The bold lines represent the moves taken during a game. The dashed lines represent moves that we (our algorithm) considered but did not make. Our second move was an exploratory move, meaning that it was taken even though some other alternative move, that leading to e', was more highly ranked. Exploratory moves do not result in learning, but each of our other moves does, causing backups as suggested by the curved arrows and detailed in the text.

Now, while we are playing, we change the probability estimates for the states in which we find ourselves during the game. We attempt to make them more accurate estimates of the probabilities of winning from those states.

Informally, we say that the probability estimate for the state after each regular move is "backed up" to the estimate for the state after our preceding move, as suggested by the arrows in **Figure 2**. More precisely, the probability estimate for the earlier state is moved a fraction of the way from its current value to the value of the later state. Letting x_k denote the state after our *k*th move, and letting $V(x_k)$ denote the estimated probability of winning from that state (the *value* of state x_k), the update rule can be written:

$$V(x_k) := V(x_k) + \alpha[V(x_{k+1}) - V(x_k)],$$

where α is a small positive fraction called the *step-size parameter*.

This update rule performs quite well with this task. For example, if the step-size parameter is reduced properly over time, this method will converge for any fixed opponent to the true probabilities of winning from each state given optimal play by the algorithm (Singh, Jaakkola, & Jordan, 1994). Furthermore, the moves then taken (except on exploratory moves) will, in fact, be the optimal moves against the opponent. If the step-size parameter is not reduced to zero over time, then a player using this rule will also play well against opponents that change their play slowly over time. This update rule is closely related to the method Samuel used in his 1959 program for learning how to play the game of checkers (Samuel, 1959). Sutton (1988), who refined and analyzed algorithms like this, called them *temporal-difference methods*.

This example is very simple, but it illustrates some of the key features of reinforcement-learning methods. First, there is the emphasis on learning while interacting with an environment, in this case with an opponent player. Second, there is a clear goal, and correct behavior requires planning or foresight that takes into account delayed effects of one's choices. The simple reinforcement-learning player of naughts and crosses will, for example, learn to set up multi-move traps for a short-sighted opponent. It is a striking feature of reinforcement learning that it can achieve the effects of planning and lookahead *without* using a world model or carrying out an explicit search over sequences of choices. To be sure, planning using world models can be useful, but it is not always worth the effort.

On the other hand, the naughts-and-crosses example is so simple that it might give the false impression that reinforcement learning is restricted to such tasks. Although naughts and crosses is a two-person game, reinforcement learning also applies in the more natural context in which there is no explicit external adversary. Naughts and crosses involves a relatively small, finite-state set, whereas reinforcement learning can be applied to very large or even infinite-state sets. For example, Tesauro (1994, 1995) combined the algorithm described above with an artificial neural network to acquire impressive skill in playing backgammon, which has a huge number of states—approximately 10^{20}. The neural network provides this program with the ability to *generalize* from its past experiences, so that in new situations it selects moves based on information saved from similar situations faced in the past, as determined by its network. Thus, how well a reinforcement-learning system can work with problems having very large state sets is intimately tied to how appropriately it can generalize from past experience. Methods for supervised learning, which focus almost exclusively on the problem of forming appropriate generalizations, are most relevant to this aspect of reinforcement learning. A neural network is clearly not the only, or necessarily the best, way to do this.

Other features of the naughts-and-crosses example are not essential to reinforcement learning, for example, learning with no prior knowledge beyond the rules of the game. However, although many other reinforcement-learning examples begin similarly devoid of knowledge, reinforcement learning by no means entails a *tabula rasa* view of learning and intelligence. On the contrary, prior information can be incorporated into a reinforcement-learning system in a variety of ways that can be critical for efficient performance (Clause & Utgoff, 1992; Lin, 1992; Maclin & Shavlik, 1994; Mitchell & Thrun, 1993).

The naughts-and-crosses player also had to look ahead one step in order to evaluate the possible immediate results of a move. To be able to do this, it had to have a model of the game that allows it to "think about" how its environment will change in response to moves that it may never make. However, the naughts-and-crosses player used its model in only a very simple way, whereas other reinforcement-learning systems make much more extensive use of environmental models (e.g., Barto et al, 1995; Moore & Atkeson, 1993; Sutton, 1990, 1991). They can generate hypothetical experiences from which they can learn in the same way that the naughts-and-crosses player learns from real experience, or they can "reason" about the consequences of possible behavior and make various kinds of plans. These more complicated model-based reinforcement-learning systems can include a full range of high-level, symbolic processing, and an important aspect of reinforcement learning is the improvement of environmental models through learning. Thus, although reinforcement learning is often associated only with very low-level processing, this is by no means an essential aspect of the approach.

On the other hand, there are reinforcement-learning methods that do not need any kind of environmental model at all. Watkins (1989) called these *primitive methods*. Systems using only primitive methods cannot even think about how their environments will change in response to a single action. Because models have to be reasonably accurate to be useful, primitive methods can have advantages over more complex methods when the crucial bottleneck in solving a problem is difficulty in constructing a sufficiently accurate environmental model. Primitive methods are also important building blocks for model-based methods.

The naughts-and-crosses player had access to the complete state of the game, but reinforcement learning can also be applied when part of the state is hidden, or when different states appear to the learner to be the same (e.g., Whitehead & Ballard, 1990; Jaakkola, Singh, & Jordan, 1995). Finally, the naughts-and-crosses player is a reinforcement-learning system on just one level. The decisions refined by learning are about the primitive moves of the game. Recalling our comments about the abstract nature of the reinforcement-learning framework, nothing prevents reinforcement learning from working at higher levels, for example, where each of the "actions" is itself the application of a possibly elaborate problem-solving method (e.g., Maes & Brooks, 1990; Mahadevan & Connell, 1991; Singh, Barto, Grupen, & Connolly, 1994). In hierarchical learning systems, reinforcement learning can work simultaneously on several levels (Dayan & Hinton, 1993; Singh, 1991, 1992).

Fully satisfactory solutions are of course not yet available in all cases. Most of the theoretical results that exist so far, in fact, apply only to problems that share with the naughts-and-crosses problem the use of a tabular representation of a finite set of state values and access to complete environmental states (Barto et al, 1995). However, many reinforcement-learning researchers, like many other AI researchers, are willing to forge ahead when theoretical guarantees are lacking, and many applications of reinforcement-learning methods have been realized in ways that go considerably beyond available theory. Moreover, some of these applications have been very successful, as in the examples mentioned above by Tesauro (1994, 1995), Zhang and Dietterich (1995), and Crites and Barto (1996).

The Credit-Assignment Problem

In his famous paper "Steps Toward Artificial Intelligence," Minsky (1961) presented the basic ideas of "success-reinforced decision models" and discussed the major computational problem that complex reinforcement-learning systems would have to solve to be successful. He called this the *credit-assignment problem*:

> In applying such methods to complex problems, one encounters a serious difficulty—in distributing credit for success of a complex strategy among the many decisions that were involved (p. 17).

Later researchers distinguished between *temporal* and *spatial* aspects of the problem. Temporal credit assignment concerns determining which actions in the sequence of preceding actions were responsible for an eventual success (or failure). For example, if you win a chess game, how should you apportion credit among all the moves you made? The spatial aspect of the problem, on the other hand, concerns allocating credit to the many, possibly simultaneous, decisions that finally yielded an overt action. For example, if in winning the chess game your temporal credit-assignment mechanism assigned a certain amount of credit to a particular move, how should you further apportion this credit among the various decisions that caused you to select it? Both aspects of the credit-assignment problem remain central problems for modern reinforcement-learning systems.

The approach to temporal credit assignment used by many reinforcement-learning systems, including the naughts-and-crosses player described above, was introduced to AI in Samuel's program for learning how to play checkers (Samuel, 1959). The idea is that a reinforcement-learning system should not have to wait to learn until an externally supplied reinforcement signal occurs. The checkers player, for example, should not have to wait until the end of a game to receive reinforcement. The player should be able to produce for itself internal reinforcement when it achieves important subgoals during a game. Moreover, the player should be able to *learn* to recognize when important subgoals are achieved. Samuel's method for doing this is related, as previously noted by Minsky (1961), to the phenomenon of conditioned reinforcement in animal learning. An event that regularly precedes a reinforcing event can itself acquire the ability to reinforce still earlier activity; i.e., the event becomes a conditioned reinforcer. Conditioned reinforcers can, in turn, confer reinforcing qualities upon earlier events, making them into conditioned reinforcers as well. This suggests a recursive mechanism by which a system can learn long sequences of actions that ultimately bring about real success, that is, success as determined by the ultimate primary reinforcer. Much of modern reinforcement learning exploits this process.

The spatial aspects of credit assignment are not unique to reinforcement learning. For example, the error backpropagation method for adjusting the weights of a multi-layer artificial neural network is a spatial credit-assignment method widely used in supervised learning (Rumelhart, Hinton, & Williams, 1986). Although this algorithm can be adapted to address temporal aspects of credit assignment, it ordinarily only addresses the spatial aspects by apportioning the credit (in this case, the blame) among the weights of a complex network for the errors made by the network as a whole. Similarly, reinforcement-learning systems have to adjust their decision rules even if some other mechanism produces timely reinforcement. Reinforcement-learning systems can use a variety of methods developed for supervised learning, although some methods are better suited than others due to the different demands of reinforcement learning.

The general approach to credit assignment taken by reinforcement-learning systems is the major feature distinguishing them from methods based more directly on evolutionary metaphors, such as genetic algorithms (Goldberg, 1989; Holland, 1975). Like reinforcement learning, evolutionary methods can be used to adapt the interactive behavior of an agent to achieve an explicit goal, but they do so without assigning credit on an intra-individual basis. For example, if an agent does well, credit is assigned to *all* of its behavior, independently of how specific components of this behavior were related to success; full credit will even be given to behavior that was not expressed during the agent's lifetime. As a consequence, evolutionary methods, when used alone, may be inherently less efficient than methods that assign credit by taking into account intra-individual details about an agent's decision mechanisms and how they are marshaled over time. On the other hand, by not attempting intra-individual credit assignment, evolutionary methods are not misled by credit's being incorrectly assigned. In any event, we do not consider evolutionary methods to be especially well adapted to the reinforcement-learning problem. Although evolution and learning, especially reinforcement learning, share many features and can naturally work together, as they do in nature, they do not have equal access to the same credit-assignment mechanisms.

The Reinforcement-Learning Problem

Although certain learning algorithms are commonly associated with reinforcement learning, it is more useful to define reinforcement learning in terms of learning *problems*, or collections of problems, rather than as a collection of algorithms. Here we present a model of the problem that many AI researchers have adopted in their approaches to reinforcement learning. This model is based on the *Markov Decision Process* formalism that has been widely studied by decision theorists (e.g., Bertsekas, 1987; Ross, 1983).

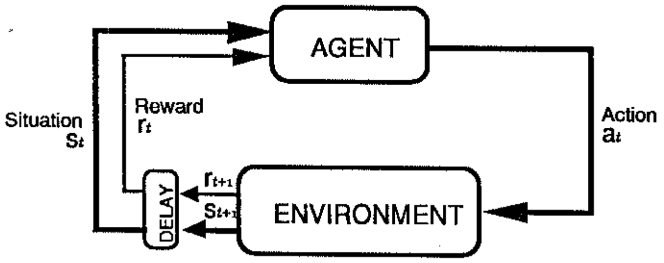


FIGURE 3. A Reinforcement-Learning Model. A reinforcement-learning agent and its environment interact over a sequence of discrete time steps. The *actions* are the choices made by the agent; the *situations* provide the agent's basis for making the choices; and the *rewards* are the basis for evaluating these choices.

The agent-environment interface

Reinforcement learning is about learning how to act to achieve a goal. A fruitful way of modeling such learning is based on viewing a decision maker, or *agent*, as a control system that is trying to develop a strategy by which it can make its environment behave in a favorable way (where "favorable" has a precise meaning). A simple type of strategy maps each situation to a probability distribution over the actions that are possible for that situation. Upon determining that it is in a new situation, the agent selects an action according to the probability distribution for that situation. As the agent learns, it changes this mapping, called its *policy*, based on its accumulating experience.

To make the model more specific, think of the agent and its environment as interacting over a potentially infinite sequence of discrete time steps $t = 1,2,3,\ldots$ At each time step t, the reinforcement-learning agent finds itself in a *situation*, $s_t \in S$, and on that basis uses its current policy to choose an *action*, $a_t \in A(s_t)$, where $A(s_t)$ is the set of actions available for situation s_t. One time step later, in part as a consequence of its action, the agent receives a numerical *reward*, $r_{t+1} \in \Re$, and finds itself in a new situation, $s_{t+1} \in S$ (**Figure 3**). Reinforcement-learning methods specify how such experiences produce changes in the agent's policy, which tells it how to select an action in any situation. Roughly speaking, the agent's objective is to find a policy that maximizes the amount of reward it receives over the long run.

It is important to understand the degree of abstraction this model involves. It is a very abstract and flexible model that can be applied at many different levels to many different problems. The actions, for example, could be low-level controls such as the voltages applied to the motors of a robot arm, or high-level decisions such as whether or not to have lunch or go to graduate school. Similarly, the situations can take a wide variety of forms. They could be low-level situations, such as direct sensor readings, or high-level ones, such as symbolic descriptions of the objects in a room. Some of the things making up a situation could even be entirely mental or subjective. For example, the agent could be in the situation of not being sure where an object is, or of having just been "surprised" in some clearly defined sense. Similarly, some actions could also be totally mental or computational, e.g., they may control what the agent chooses to think about, or where it focuses its attention. In general, actions can be the results of any decisions we learn how to make, and the situations can be anything we can sense that might be useful in making the decisions.

In particular, it is a mistake to think of the interface between a reinforcement-learning agent and its environment as the physical boundary between a robot's or an animal's body and the external environment. Usually the boundary is drawn closer to the agent. For example, the motors and mechanical linkages of a robot and its sensing hardware should usually be considered parts of the environment rather than parts of the learning agent, even though these parts were probably designed to make the learning agent's task easier. Similarly, if we apply the model to a person or animal, the skeleton, muscles, and sensory organs should all be considered part of the learning agent's environment. Reinforcers, too, may presumably be computed inside the physical bodies of natural and artificial learning systems, but are considered external to the reinforcement-learning agent.

The general rule we follow is that anything that cannot be changed arbitrarily by the learning agent is considered external to it and, thus, part of its environment. Note that we do not assume that events in the environment are unknown to the agent, only that they are incompletely controllable. For example, the agent will often know quite a bit about how its reinforcers are computed as a function of its actions and the situations in which they occur. But we always consider the reward computation to be external to the agent because it defines the problem facing the agent and, thus, is beyond its ability to change arbitrarily. In some cases, in fact, the agent may know *everything* about its environment and still face a difficult reinforcement-learning problem, just as we may know exactly how a puzzle like Rubik's cube works but still be unable to solve it. The agent-environment boundary represents the limit of the agent's *control*, not of its knowledge.

The agent-environment boundary can even be located at different places for different purposes. In a complicated robot, many separate reinforcement-learning agents may be operating at once, each with its own boundary. For example, one agent may make high-level decisions that form part of the situations faced by a lower-level agent that implements the high-level decisions. In practice, the agent-environment boundary is determined once one has selected particular sensations, actions, and reinforcers, and thus identified a particular decision-making problem of interest.

The reinforcement-learning model is a considerable abstraction of the problem of learning to make decisions based on their consequences. It proposes that whatever the details of the sensory and control apparatus, and whatever objective one is trying to achieve, any problem of learning goal-directed behavior can be reduced to three signals passing back and forth between an agent and its environment: One signal represents the choices made by the agent (the actions); a second signal represents the basis on which the choices are made (the situations); and a third signal defines the goal of learning (the rewards). We do not claim that this framework is adequate to usefully model *all* decision-learning problems, but it has proven to be widely applicable. Of course, the situation and action representations will vary greatly from application to application, and will strongly affect performance. In reinforcement learning, as in other kinds of learning, such representational choices are at present more art than science.

Goals, rewards, and returns

In reinforcement learning, the concept of goal is modeled by a special scalar signal called the *reward* that passes from the environment to the agent. Informally, the agent's goal is to maximize the total reward it receives. This means not just immediate reward, but reward over the long run.

The use of a scalar reward signal to formalize the idea of a goal is one of the most distinctive features of reinforcement learning. Although this way of formulating goals might at first appear limiting, in practice it has proven to be very flexible and very widely applicable. The best way to see this is to consider examples of how it may be used. For example, to train a robot to walk, researchers have provided reward on each time step proportional to the robot's forward motion. In learning to run a maze, the reward is often zero except upon reaching the goal, when it becomes +1. Another common approach in maze learning is to give a reward of -1 for every time step that passes prior to reaching the goal; this encourages the agent to reach the goal as quickly as possible. To train a robot to find and collect empty soda cans for recycling, one might give it a reward of +1 for each empty can collected. One might also give the robot punishers when it bumps into things, or when people yell at it. For an agent learning to play backgammon or chess, the natural rewards for winning, losing, and drawing are +1, -1, and 0, respectively.

It is important to remember that rewards define the ultimate goal of the learning process. The rewards delivered to a reinforcement-learning agent should represent what you really want the agent to do. In particular, the reward signal is not the place to impart to the agent prior knowledge about *how* to achieve what you want it to do. For example, a chess-playing agent should be rewarded only for actually winning, not for achieving subgoals such as taking its opponent's pieces or gaining control of the center of the board. If these kinds of subgoals are rewarded, the agent might find a way to achieve the subgoals without achieving the real goal, e.g., taking the opponent's pieces even at the cost of losing the game. The reward signal is a way of communicating to the robot *what* it should achieve, not *how* it should be achieved.

Newcomers to this model of reinforcement learning are sometimes surprised that the rewards—the definition of the goal of learning—are computed in the environment rather than in the agent. Certainly, most ultimate goals for animals are recognized by computations occurring inside their bodies, e.g., by sensors for recognizing food and hunger, pain and pleasure, etc. However, as we discussed in the previous section, one can simply redraw the agent-environment interface such that these parts of the body are considered to be outside of the agent (and thus part of the agent's environment). For example, if the goal concerns a robot's "internal" energy reservoirs, then these are considered part of the environment; if the goal concerns the positions of the robot's limbs, then these too are considered part of the environment—the boundary is drawn at the interface between the limbs and their control systems.

Roughly speaking, structures and processes are considered part of the agent if they are completely, directly, and with certainty, controllable; otherwise they are considered part of the environment. The ultimate goal is always something over which the reinforcement-learning agent has imperfect control: It cannot, for example, simply decree that the goal has been achieved (in the same way that it can arbitrarily set an internal parameter of its decision-making process). Therefore, we place the reward source outside of the agent. Note that this does not preclude the agent from defining for itself an internal goal, or a sequence of internal goals. Indeed, the commonly used method for temporal-credit assignment, based on Samuel's approach described above, does just that: It effectively defines internal goals.

Until this point, we have been imprecise when we spoke of the goal of learning as maximizing reward over the long run. How might this be formally defined? If the sequence of rewards received after time step t is denoted r_{t+1}, r_{t+2}, r_{t+3}, ..., then what aspect of this sequence do we wish to maximize? There are several useful answers to this question. The simplest is to maximize the *total reward*:

$$r_{t+1} + r_{t+2} + r_{t+3} + \dots + r_T \tag{1}$$

where T is a final time step. This approach makes sense in applications in which there is a natural notion of final time step in a trial, that is, when the agent-environment interaction breaks naturally into subsequences, such as plays of a game, trips through a maze, or any sort of repeated attempt where each repetition ends with a reset to a standard state. In these cases Equation 1 defines the *return* for time step t, i.e., the return that accumulates *after* time step t.

On the other hand, suppose that the agent-environment interaction does not naturally break into identifiable subsequences but simply goes on without limit. This would be the natural way to characterize a continuous process-control application, or an application to a robot with a long expected lifespan. The total-reward formulation then becomes problematic because the final time step becomes T approaches ∞, and the return as given by Equation 1 becomes a sum of an infinite number of terms. Thus the return, which is what the agent is trying to maximize, could itself be infinite (e.g, if the agent receives a reward of $+1$ at each time step).

The additional concept we need is that of *discounted return*. According to this approach, the agent's objective is to learn how to select actions so that, at every time step, the discounted sum of the rewards received over the future is maximized. That is, the objective is to learn to maximize the following definition of return for each time step t:

$$r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots = \sum_{k=1}^{\infty} \gamma^{k-1} r_{t+k} \tag{2}$$

where γ is a positive number called the *discount factor*.

The discount factor determines the present value of future rewards: A reward received k time steps in the future is worth γ^{k-1} times what it would be worth if it were received immediately. If $0 \le \gamma < 1$, this infinite discounted sum is finite as long as each individual reward is finite. If $\gamma = 0$, the agent is "myopic," i.e., only concerned with maximizing immediate rewards. Its objective in this case would be to learn how to act at each time step t so as to maximize only r_{t+1}. If each of the agent's actions happened only to influence the immediate reward, not future rewards as well, then a myopic agent could maximize Equation 2 by separately maximizing each immediate reward. But, in general, acting to maximize immediate reward can reduce access to future rewards so that the total reward may actually be reduced. As γ approaches one, the objective takes future rewards into account more strongly: The agent becomes more farsighted. Other definitions of return for infinite-duration problems are possible (Mahadevan, 1996), but the discounted return is the simplest mathematically.

Example: A problem that served as an early illustration of reinforcement learning is the problem of *pole-balancing* (Michie & Chambers, 1968). The objective here was to apply forces to a cart moving along a track so as to keep a pole hinged to the cart from falling over (**Figure 4**). We define a *balancing failure* as the fall of the pole past a given angle from vertical or the cart's exceeding the limits of the track. The pole is reset to vertical after each balancing failure. This problem could be treated as a total-reward problem, where the natural subsequences are the repeated attempts to balance the pole. The reward

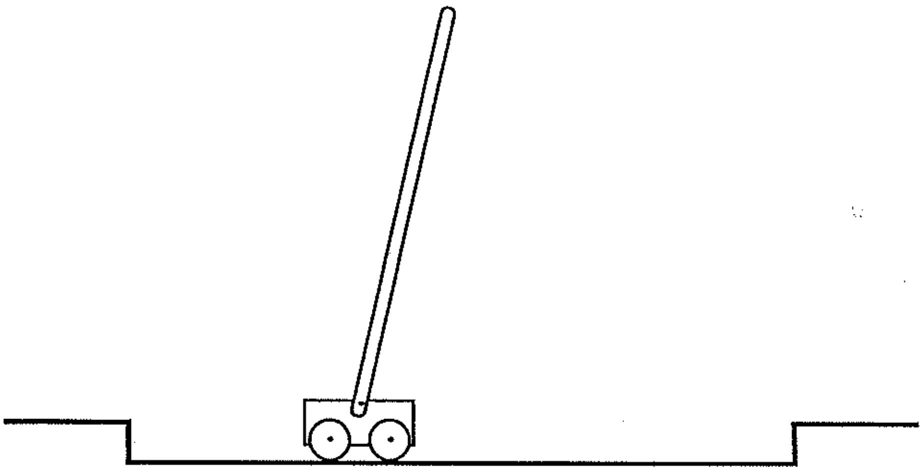


FIGURE 4. The Pole-Balancing Problem. The objective is to apply forces to a cart moving along a track so as to keep a pole hinged to the cart from falling over.

in this case would be +1 for every time step on which failure did not occur, so that the return at each time would be the number of steps before failure. Alternatively, a punisher of -1 could be given for each failure and zero reward at all other times. The return at each time would then be related to $-\gamma^k$, where k is the number of time steps before failure. In either case, the return is maximized by keeping the pole balanced for as long as possible.

Situations and states

The *state* of a system with respect to an external observer is a summary of the observer's past experience with the system. The summary need not be a complete history of every observed input and output, but it must contain all the information that makes a difference as far as the system's future behavior is concerned. In particular, the observer must be able (in principle) to predict the system's future behavior just as well from knowledge of its current state as from knowledge of its complete history. For example, the state of a cannonball in flight is its current position vector and velocity vector. It doesn't matter how its current position and velocity came about.

In reinforcement learning, the external observer is the agent, and the state of interest is the state of the environment. In fact, the notion of "situation" is meant to be an approximation of the environment's state. What exactly is the state of the environment? The agent's past experience with the environment consists of all of the previous situations, actions, and rewards. Assuming that interaction began at $t = 0$, the complete *history* at time t is

$$H_t = \{s_t, r_t, a_{t-1}, s_{t-1}, r_{t-1}, a_{t-2}, s_{t-2}, \ldots, r_1, a_0, s_0\}. \tag{3}$$

Any signal $x_t \in X$, $t \geq 0$, gives the state of the environment at time step t if and only if the joint probability of the state, situation, and reward at time step $t+1$, given x_t and a_t, the action at time step t, is the same as their joint probability given H_t and a_t. When the number of possible states, situations, rewards, and actions are finite, this can be written simply as follows:

$$P\{x_{t+1}=x, s_{t+1}=s, r_{t+1}=r \mid x_t, a_t\} = P\{x_{t+1}= x, s_{t+1}=s, r_{t+1}=r \mid H_t, a_t\}, \tag{4}$$

for all $t \geq 0$, $x \in X$, $s \in S$, $r \in \Re$, $a \in A(s_t)$, and all possible H_t, where X, S, $\Re$, and $A(s_t)$ are finite sets of possible states, situations, rewards, and actions, respectively. In reinforcement learning, situations are intended to approximate the environment's states. The situations are in fact true states if and only if

$$P\{s_{t+1}=s, r_{t+1}=r \mid s_t, a_t\} = P\{s_{t+1}=s, r_{t+1}=r \mid H_t, a_t\} \tag{5}$$

for all $t \geq 0$, $s \in S$, $r \in \Re$, $a \in A(s_t)$, and all possible H_t. (If any of these sets are not finite, e.g., if a reward can be any real number, then the same conditions can be written in terms of probability density functions.) In this important special case, the environment and its interface define a Markov Decision Process, or MDP. If an MDP has a finite number of states, and a finite number of actions are available for each state, then it is a *finite MDP*. Because it is particularly easy to conceptualize and to prove theorems about finite MDPs, they play a central role in the theoretical analysis of reinforcement learning.

The conditional probability distributions given by Equation 5 constitute a complete description of the *dynamics* of the MDP. As far as the agent is concerned, the dynamics specify how the environment changes over time in response to its actions. If a reinforcement-learning agent has complete knowledge of its environment's dynamics, then it faces a reinforcement-learning problem under conditions of *complete information*. Most problem-solving methods in AI have addressed problems of complete information, whereas reinforcement learning focuses primarily on problems of incomplete information. The reader should be careful not to confuse complete and incomplete *information* with complete and incomplete *observation* of the environmental state. We refer to the case of complete observation by saying that the environment has the Markov property.

One can show that by iterating Equation 4 or Equation 5 an agent can predict *any* future state and reward from knowledge only of the current state and its proposed course of action (together with knowledge of the dynamics) as well as would be possible given the complete history. It also follows that the situations in MDPs provide the best possible basis for choosing actions. That is, the best policy for choosing actions as a function of situations is just as good as the best policy for choosing actions as a function of complete histories.

Even when the situations are not technically states in the sense of exactly satisfying Equation 5, it may still be appropriate to think of the situation in reinforcement learning as an approximation to the environment's state. In particular, we always want the situation to be a good basis for predicting future rewards and for selecting actions. For some purposes, it is also desirable to use present situations to accurately predict following situations. States provide an unsurpassed basis for doing all of these things. To the extent that situations approximate states in these ways, one can obtain better performance from reinforcement-learning systems. For all of these reasons, it is useful to think of the situation at each time step as an approximation to an MDP's state, although one should remember that a situation is often not precisely a state. Although most reinforcement-learning algorithms can be applied when the situations are not states, sometimes with good results, almost all of the formal theory rests on the assumption that situations are actual states.

Example: In the pole-balancing problem introduced in the previous section, a situation would be a state if it exactly specified, or made it possible to exactly reconstruct, the position and velocity of the cart along the track, the angle between the cart and the pole, and the rate at which this angle is changing (the angular velocity). In an idealized cart-pole system, this information would be sufficient to exactly predict the future behavior of the cart and pole, given the actions taken by the controller. In practice, however, it is never possible to know this information exactly because any real sensor would introduce some distortion and delay in its measurements. Furthermore, in any real cart-pole system there are always other components of the state, such as the bending of the pole, the temperatures of the wheel and pole bearings, and various forms of backlash, which slightly affect the behavior of the system. These factors would cause violations of Equation 5 if the role of state were played by only the positions and velocities of the cart and the pole.

However, often the situations of the positions and velocities serve quite well as approximate states. In several of the early studies of learning the pole-balancing problem, in fact, learning was successful despite the fact that each situation provided only a very coarse representation of the true state. For example, in our work (Barto, Sutton, & Anderson, 1983), the possible cart positions were divided into three regions: right, left, and middle. The situations indicated only in which of these three large regions the cart was located (and there were similarly rough quantizations of the other three intrinsic state variables). These rough approximations to the state were sufficient to easily solve the problem using reinforcement learning. In fact, this coarse representation of the state probably facilitated learning because it forced the learning agent to ignore fine distinctions that would not have been particularly useful in solving the problem.

Example: In draw poker, each player is dealt a hand of five cards. There is a round of betting in which each player exchanges some of his cards for new ones, and then there is a final round of betting. At each round of betting, a player must match the highest bets of the other players or else drop out (fold). After the second round of betting, the player with the best hand and who has not folded is the winner and collects all the bets.

The relevant state in draw poker is different for each player. Each player knows the cards in his own hand, but can only guess at those in the other players' hands. A common mistake is to think that the state must include the contents of all the players' hands and the cards remaining in the deck. However, this would provide more information than the state. In a fair game, one assumes that the players are in principle unable to determine these things from their past observations. If a player did have such information, some future events (such as the cards one could exchange for) could be *better* predicted than by remembering all past observations.

In addition to knowledge of one's own cards, the state in draw poker includes knowledge of the bets and the numbers of cards drawn by the other players. For example, if a player draws three new cards, you may suspect he retained a pair and adjust your estimate of the strength of his hand accordingly. The players' bets also influence your assessment of their hands. In fact, all of your past history with these particular players is part of the state. Does Ellen like to bluff, or does she play conservatively? Does her face or demeanor provide clues to the strength of her hand? How does Joe's play change when it is late at night, or when he has already won a lot of money?

Although everything ever observed about the other players may have an effect on the probabilities that they are holding various kinds of hands, in practice this is far too much to remember and analyze, and most of it will have no clear effect on one's predictions and decisions. Very good poker players are adept at remembering just the key clues and at sizing up new players quickly, but no one remembers everything that may be relevant. As a result, the situations people use to make their poker decisions are imperfect state models, and the decisions themselves are presumably imperfect. Nevertheless, people can still make very good decisions in such problems. The inability to have access to a *perfect* representation of the environment's state is probably not a severe problem for an AI agent.

Value Functions

Almost all reinforcement-learning algorithms are based on estimating *value functions*—functions of situations, or of situation-action pairs, that estimate *how good* it is for the agent to be in that situation. The notion of "how good" is defined in terms of expected future rewards or, to be precise, as the expected return given, by Equation 1 or Equation 2, for example. Of course, the rewards an agent can expect to receive in the future depend on what actions it takes. Accordingly, value functions are defined with respect to a particular policy. Recall that a policy, let us call it π, is a mapping from situations $s \in S$ to probability distributions over possible actions $a \in A(s)$. Informally, the value of a situation under a policy π, denoted $V^\pi(s)$, is the expected return when starting in s and following π. For MDPs, we can define $V^\pi(s)$ formally as:

$$V^\pi(s) = E_\pi \{\sum_{k=t}^{T} r_k \mid s_t = s\}, \tag{6}$$

for the total-reward case, where the return is defined by Equation 1, and where $E_\pi\{\}$ denotes the expected value given that the agent follows policy π. For the discounted case, in which the return is given by Equation 2, $V^\pi(s)$ is defined as:

$$V^\pi(s) = E_\pi \{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s\}. \tag{7}$$

Similarly, following Watkins (1989), we define the *action-value*, or *quality*, of taking action a in situation s under a policy π, denoted $Q^\pi(s,a)$, as the expected return starting from s, taking the action a, and thereafter following policy π:

$$Q^{\pi}(s,a) = E_{\pi}\{\sum_{k=t}^{T} r_k \mid s_t=s,\, a_t=a\}, \tag{8}$$

for the total-reward case, and

$$Q^{\pi}(s,a) = E_{\pi}\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t=s,\, a_t=a\}, \tag{9}$$

for the discounted-reward case. In either case, the (situation) value function is related to the action value function by

$$V^{\pi}(s) = E\,\{Q^{\pi}(s,a_{\pi(s)})\}, \tag{10}$$

where $a_{\pi(s)}$ is the action selected according to the probability distribution over actions given by $\pi(s)$.

For MDPs, the value functions $V^{\pi}(s)$ and $Q^{\pi}(s,a)$ can be estimated from experience. For example, if an agent follows policy π and maintains an average, for each situation encountered, of the actual returns that have followed that situation, then the averages will converge to the situation's value, $V^{\pi}(s)$, as the number of times that situation is encountered approaches infinity. If separate averages are kept for each action taken in a situation, then these averages will similarly converge to the action values, $Q^{\pi}(s,a)$. Estimation methods of this kind are often called *Monte Carlo methods* because they involve averaging over many random samples of actual returns. Of course, if there are very many situations, or very many actions possible in each situation, then it may not be practical to keep separate averages for each situation individually. Instead, the agent may maintain V^{π} and Q^{π} as parameterized functions and adjust the parameters to better match the observed returns. This can also produce accurate estimates, although much depends on the nature of the parameterized function approximation.

A fundamental property of value functions used throughout reinforcement learning and dynamic programming is that they satisfy particular recursive relationships if the situations are genuine states. For any policy π and any state s the following consistency condition holds between the values of any "neighboring" states:

$$\begin{aligned} V^{\pi}(s) &= E_{\pi}\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t=s\} \\ &= E_{\pi}\{r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \mid s_t=s\} \\ &= E_{\pi}\{r_{t+1} + \gamma V^{\pi}(s_{t+1}) \mid s_t=s\}, \end{aligned} \tag{11}$$

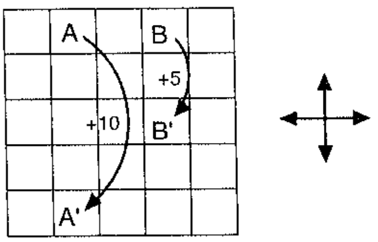
for the discounted-reward case, and

$$V^{\pi}(s) = E_{\pi}\{r_{t+1} + V^{\pi}(s_{t+1}) \mid s_t = s\} \tag{12}$$

for the total-reward case for $t \neq T$. Similar consistency conditions hold for Q^{π}. These conditions can be used in several different ways to compute or approximate V_{π} and Q^{π}.

Example: **Figure 5a** uses a rectangular grid to illustrate a simple finite MDP. The cells of the grid correspond to the states (situations) of the problem. At each cell, four actions are possible: NORTH, SOUTH, EAST, and WEST, which deterministically cause the agent to move one cell in the respective direction in the grid. Actions that would take the agent off the grid leave its location unchanged, but also result in a reward of -1. Other actions result in a reward of 0, except those that move the agent out of the special states A and B. From state A, all four actions yield a reward of +10 and take the system to A'. From state B, all actions yield a reward of +5 and take the system to B'.

Suppose the agent selects the four actions with equal probabilities in all states. **Figure 5b** shows the value function, V^{π}, for this policy, for the discounted-reward case with $\gamma = 0.9$. This value function was computed by solving the system of equations given by Equation 11. Notice the negative values near the lower edge; these are the result of the high probability of hitting the edge of the grid there under the random policy. Notice that A is the best state to be in under this policy, but that its expected return is less than 10, its immediate reward. The expected return is reduced because from A the agent is taken to A', from which it is likely to run into the edge of the grid. State B, on the other hand, is valued more than 5, its immediate reward, because from B the agent is taken to B', which has a positive value. From B'



3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

(a) (b)

FIGURE 5. Rectangular-Grid Illustration of a Value Function. **a.** Each cell is a state, and the agent can move one cell in any of the cardinal directions. Cells A, A', B, and B' are special states. Rewards are generated as described in the text. **b.** The value function for the policy of selecting the four actions with equal probabilities in all states.

the expected penalty (negative reward) for possibly running into an edge is more than compensated for by the expected gain for possibly stumbling onto A or B.

The Optimality Equation

Solving a reinforcement-learning problem means finding a policy that maximizes the expected return for each situation. For finite MDPs, we can precisely define the optimal solution in the following simple way. Value functions define a partial ordering over policies. A policy π is better than or equal to a policy π' if its expected return is greater than or equal to that of π' for all states. In other words, $\pi \geq \pi'$ if and only if $V^{\pi}(s) \geq V^{\pi'}(s)$ for all $s \in S$. There is always at least one policy that is better than or equal to all other policies. This is an *optimal policy*. Although there may be more than one, we denote all the optimal policies by π^*. They share the same value function, denoted V^*, defined as

$$V^*(s) = V^{\pi^*}(s) = \max_{\pi} V^{\pi}(s) \quad \text{for all } s \in S \tag{13}$$

and the same action-value function, denoted Q^*, defined as

$$Q^*(s,a) = Q^{\pi^*}(s,a) = \max_{\pi} Q^{\pi}(s,a) \quad \text{for all } s \in S, \text{ for all } a \in A(s). \tag{14}$$

Because V^* is the value function for a policy, it must satisfy a consistency condition such as Equation 11 or Equation 12. Because it is the optimal value function, however, V^*'s consistency condition can be written in a special form, often called the *optimality equation*, which is independent of the policy. Intuitively, the optimality equation is based on the fact that the value of a state under an optimal policy must equal the expected return for the best action from that state:

$$V^*(s) = \max_{a} E_{\pi^*} \{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\}, \tag{15}$$

for the discounted-reward case. From this follows the optimality equation for the discounted-reward case:

$$V^*(s) = \max_{a} E \{r_{t+1} + \gamma V^*(s_{t+1}) \mid s_t = s, a_t = a\}, \tag{16}$$

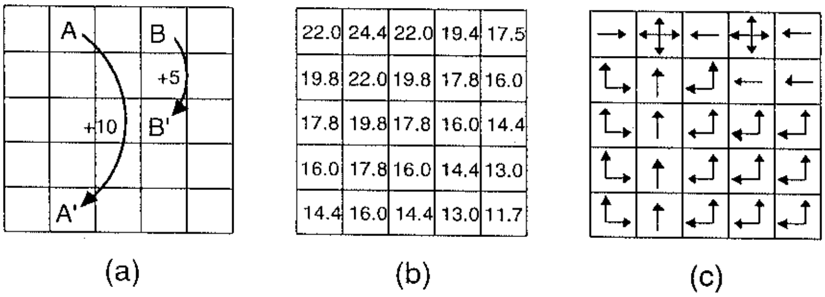
for all $s \in S$. For finite MDPs, the optimality equation has a unique solution independent of the policy, unlike the ordinary consistency condition (11), whose solution depends on the policy. The optimality equation is actually a system of equations, one for each state, so that there are $|S|$ equations in $|S|$ unknowns. If the dynamics of the environment are known, then in principle one can solve this system of equations for V^* using one of a variety of methods for solving systems of nonlinear equations.

Once one has V^*, it is relatively easy to determine an optimal policy. For each state s, there will be one or more actions at which the maximum is obtained in the optimality equation. These are all equally good actions. Any policy that selects only from among these is an optimal policy. Another way of saying this is that any policy that is *greedy* with respect to the optimal evaluation function V^* is an optimal policy. The term greedy is used in computer science to describe any search or decision procedure that selects alternatives based only on local or immediate considerations, without considering the possibility that such a selection may prevent future access to even better alternatives (Pearl, 1984). Consequently, it is descriptive of policies that select actions based only on their short-term consequences. The beauty of V^* is that if one uses it to evaluate the short-term consequences of actions, specifically the one-step consequences, then a greedy policy is actually optimal in the long-term sense in which we are interested because V^* already takes into account the reward consequences of all possible future behavior. By means of V^*, the optimal expected long-term return is turned into a quantity that is locally and immediately available for each state.

The optimality equation therefore provides one route for finding an optimal policy, and thus for solving a reinforcement-learning problem. Unfortunately, the solution outlined above is almost never directly useful. This solution relies on three assumptions that are rarely true in practice: (1) Situations are actual states, i.e., the agent-environment interaction can be modeled as an MDP; (2) We accurately know the complete dynamics of the environment, required to even obtain the optimality equation; and (3) We need enough computational resources to complete the computation of the solution. For the kinds of problems in which we are interested, one is generally not able to implement this solution exactly because various combinations of these assumptions are violated. For example, although the first two assumptions present no problems for the game of backgammon, the third is a major impediment. Since the game has about 10^{20} states, it would take thousands of years on today's fastest computers to solve the optimality equation for V^*. Unless there is some special additional mathematical structure that can be exploited, one has to settle for approximate solutions.

Many different decision-making methods can be viewed as ways of approximately solving the optimality equation. For example, heuristic search methods of AI (Pearl, 1984) can be viewed as expanding the right-hand side of Equation 16 several times, up to some depth, forming a "tree" of possibilities, and then using a heuristic evaluation function to approximate V^* at the "leaf" nodes. (Heuristic search methods such as A^* are almost always based on the total-reward case, e.g., when the rewards are negative costs.) The methods of dynamic programming can be related even more closely to the optimality equation (Bertsekas, 1987). Many primitive reinforcement-learning methods are well understood as approximately solving the optimality equation, using actual experienced transitions in place of knowledge of the expected transitions.

Example: Suppose we solve the optimality equation for the simple grid problem introduced in the previous example and shown again in **Figure 6a.** Recall that state A is followed by a reward of +10 and transition to state A', while state B is followed by a reward of +5 and transition to state B'. **Figure 6b** shows the optimal value function, and **Figure 6c** shows the corresponding optimal policies. Where there are multiple arrows in a cell, either action is optimal.



22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

FIGURE 6. Rectangular-Grid Solution. **a.** The rectangular-grid illustration. **b.** The optimal value function, V^*. **c.** Optimal policies, π^*. Where there are multiple arrows in a cell, either action is optimal.

Learning

We have said that an agent's objective in a reinforcement-learning problem is, roughly speaking, to find a policy that maximizes the amount of reward it receives over the long run. We can now be more precise about what this means. The agent may be thought of as interacting with its environment over an infinite number of time steps. If the agent is trying to maximize total reward given by Equation 1 over subsequences of finite length, then we can imagine that it experiences an infinite number of such subsequences (e.g., can play an infinite number of games of backgammon). A reinforcement-learning agent has successfully completed learning when all the actions that it selects are optimal, i.e., when all of its actions are given by some optimal policy. This means that *after* successful learning, the agent always acts to maximize the expected return from each situation it encounters. Obviously, this ideal situation is usually only achievable in the limit, if at all, as the time step of interaction goes to infinity. Most of the algorithms we consider have been designed to achieve, under idealized circumstances, this kind of learning-in-the-limit, or *asymptotic learning*. This is what we mean by saying that an agent should *eventually* learn to act optimally.

Optimal learning may be contrasted with asymptotic learning. One might imagine that the agent should not only achieve optimal behavior in the limit, but should also improve its behavior as quickly as possible. That is, it should *optimally learn how to behave optimally* by making the best use of all of the experience it accumulates during its lifetime, as well as any prior knowledge it might bring to the task. An agent that learns optimally would maximize the total amount of reward it receives over its entire lifetime, not just over some time period in the infinite future. Although an ideal reinforcement-learning agent would be capable of optimal learning, we do not regard optimal learning as a realistic goal in designing reinforcement-learning agents. For the kinds of problems in which we are interested, optimal learning strategies can be generated only with extreme computational cost. We are, however, interested in algorithms by which an agent can improve its performance efficiently over time but without undertaking the complex process of designing an optimal learning strategy.

Even the ability to asymptotically learn to act optimally is usually beyond what is possible for a reinforcement-learning agent. We do not realistically expect that a reinforcement-learning agent would ever really achieve optimality even if it could learn over an infinite time period. A well-defined notion of optimality organizes the approach to learning we describe in this chapter and provides a way to understand the theoretical properties of various learning algorithms. However, it is an ideal that reinforcement-learning agents can only approximate to varying degrees. We noted previously that even if the reinforcement-learning agent has a complete and accurate model of its environment's dynamics, it is usually impossible for the agent to simply compute an optimal policy by solving the optimality equation. For example, board games such as chess are a tiny fraction of human experience, yet large, custom-designed computers still cannot compute the optimal moves. A critical aspect of the problem facing a reinforcement-learning agent will always be the computational power available to it, in particular, the amount of computation it can perform in a single time step. Although it is unclear how to quantify computational demands, they must be recognized.

The memory available to the agent is also an important constraint. The agent may require memory to build up approximations of value functions, policies, and models. In problems with small, finite situation sets, it is often possible to form these approximations using arrays or *tables*. In most cases of practical interest, however, there are far more situations than could possibly be entries in a table. In these cases the functions must be approximated using some sort of *compact representation*. Most of the theory of reinforcement learning applies to the tabular case, but many practical applications have used more compact representations.

So, our model of the reinforcement-learning problem forces us to settle for approximations. However, it also presents some unique opportunities for achieving useful approximations. For instance, we said above that a reinforcement-learning agent has successfully completed learning when all the actions it selects are given by some optimal policy. But, this does not mean that the agent's policy has to be optimal. An optimal policy specifies an optimal action for every possible situation, but for an agent to behave optimally its policy has to be optimal only for the situations it actually encounters. How the agent would act in situations it never encounters has no impact on the total amount of reward it will receive. Similarly, in approximating optimal behavior, there may be many situations that the agent will encounter with such a low probability that selecting suboptimal actions for them will have little impact on the amount of reward it receives. Tesauro's backgammon player, for example, plays with exceptional skill even though it might make very bad decisions on board configurations that occur rarely in games against experts. In fact, such rare configurations may make up a very large fraction of the game's state set. The interactive nature of reinforcement learning makes it possible to approximate optimal policies in ways that put more effort into learning to make good decisions for frequently encountered situations, at the expense of less effort for infrequently encountered situations. This is a key property that distinguishes reinforcement learning from other approaches to approximately solving MDPs.

Summary

Let us summarize the elements of the model of a reinforcement-learning problem that we have presented. Reinforcement learning is about learning how to behave in order to achieve a goal. The reinforcement-learning *agent* and its *environment* interact over a sequence of discrete time steps. The specification of their interface defines a particular problem: The *actions* are the choices made by the agent; the *situations* provide the agent's basis for making the choices; and the *rewards* are the basis for evaluating these choices. Everything inside the agent is completely known and controllable by the agent; everything outside is incompletely controllable but may or may not be completely known. A *policy* is a stochastic rule by which the agent selects actions as a function of situations. Roughly, the agent's objective is to learn a policy that maximizes the amount of reward it receives over the long run.

The *state* of the environment is a summary of the history of its situations, inputs (agent actions), and rewards that is sufficient to determine how it will behave in the future. The situation is meant to approximate the state. The *dynamics* of the environment are the stochastic relationships between the state and action at one time step and the situation and reward at the next. If an environment has the *Markov property*, then knowledge of the situation is sufficient to predict the environment's future behavior given a proposed course of action; the situation is a sufficient proxy for the state. This is rarely exactly true, but often nearly so, and situations should be chosen or constructed so that the Markov property approximately holds. If the Markov property does hold, then the environment is called a *Markov Decision Process*, or MDP. A *finite MDP* is an MDP with finite situation and action sets. Most of the current theory of reinforcement learning is restricted to finite MDPs.

The *return* is the function of future rewards that the agent seeks to maximize. It has several different definitions depending upon whether one is interested in *total reward* or *discounted reward*. A policy's *value function* assigns to each situation the expected return from that situation given that the agent uses the policy. The *optimal value function* assigns to each state the largest expected return from that state achievable by any policy. A policy whose evaluation function is the optimal evaluation function is an *optimal policy*. Whereas there is only one optimal evaluation function for a given MDP, there may be many optimal policies. Any policy that is greedy with respect to the optimal evaluation function is an optimal policy. The *optimality equation* is a special consistency condition that the optimal value function must satisfy and that can, in principle, be solved for the optimal value function, from which an optimal policy can be determined with relative ease.

Most of the algorithms that have been developed for reinforcement learning were designed for *asymptotic learning*, which means that, under ideal circumstances, as learning continues indefinitely, all the agent's actions approach optimal actions. We pointed out that this does not mean that the agent's policy must become an optimal policy; it only has to be optimal for the situations the agent actually encounters. We contrasted this with *optimal learning*, in which the agent should improve its behavior as quickly as possible by making the best possible use of all of the experience it accumulates during its lifetime, as well as any prior knowledge it might bring to the task. Although the rate of learning is a central issue in reinforcement learning, we do not regard optimal learning as a realistic goal in designing reinforcement-learning algorithms due to the extreme computational cost of obtaining optimal learning strategies for the kinds of problems that interest us.

The ability to asymptotically learn to act optimally is also usually impossible for a realistic reinforcement-learning agent due to limitations in computational resources. Even if the agent has a complete and accurate model of its environment, the agent may not be able to perform enough computation per time step to fully use it. The memory available to the agent is also an important constraint. The agent may require memory to build up approximations of value functions, policies, and models. In most cases of practical interest, there are far more situations than could possibly be entries in a look-up table, and the functions must be approximated using some sort of *compact representation*.

Although most of the theory of reinforcement learning is restricted to the tabular case, many practical applications have used more compact representations.

Reinforcement-learning problems differ according to the level of knowledge initially available to the agent. In problems of *complete information*, the agent has a complete and accurate model of its environment's dynamics. In problems of *incomplete information*, this level of knowledge is not available. It is important not to confuse complete and incomplete *information* with complete and incomplete *observation* of the environmental state. We refer to the case of complete observation by saying that the environment has the Markov property. For problems of incomplete information, *model-based* reinforcement-learning methods attempt to make up for the lack of a model by learning a model on line based on experience with the environment. *Primitive* methods attempt to optimize the policy without constructing a model of the environment's dynamics. Intermediate cases are possible as well.

A well-defined notion of optimality organizes the model of reinforcement learning we have described in this chapter. Optimality provides a way to understand the theoretical properties of various learning algorithms, but it is an ideal that reinforcement-learning agents can only approximate to varying degrees. In reinforcement learning, we are concerned with cases in which optimal solutions cannot be found but can be approximated in some way.