福昕PDF编辑器

·永久 ·轻巧 ·自由

升级会员　　批量购买

永久使用
无限制使用次数

极速轻巧
超低资源占用，告别卡顿慢

自由编辑
享受Word一样的编辑自由

扫一扫，关注公众号

1 Introduction

The idea that we learn by interacting with our environment is probably the first to occur to us when we think about the nature of learning. When an infant plays, waves its arms, or looks about, it has no explicit teacher, but it does have a direct sensori-motor connection to its environment. Exercising this connection produces a wealth of information about cause and effect, about the consequences of actions, and about what to do in order to achieve goals. Throughout our lives, such interactions are undoubtedly a major source of knowledge about our environment and ourselves. Whether we are learning to drive a car or to hold a conversation, we are all acutely aware of how our environment responds to what we do, and we seek to influence what happens through our behavior. Learning from interaction is a foundational idea underlying nearly all theories of learning and intelligence.

In this book we explore a *computational* approach to learning from interaction. Rather than directly theorizing about how people or animals learn, we explore idealized learning situations and evaluate the effectiveness of various learning methods. That is, we adopt the perspective of an artificial intelligence researcher or engineer. We explore designs for machines that are effective in solving learning problems of scientific or economic interest, evaluating the designs through mathematical analysis or computational experiments. The approach we explore, called *reinforcement learning*, is much more focused on goal-directed learning from interaction than are other approaches to machine learning.

Richard Sutton
Sat May 31 14:27:51 EDT 1997

1.1 Reinforcement Learning

Reinforcement learning is learning what to do---how to map situations to actions---so as to maximize a numerical reward signal. The learner is not told which actions to take, as in most forms of machine learning, but instead must discover which actions yield the most reward by trying them. In the most interesting and challenging cases, actions may affect not only the immediate reward, but also the next situation and, through that, all subsequent rewards. These two characteristics---trial-and-error search and delayed reward---are the two most important distinguishing features of reinforcement learning.

Reinforcement learning is defined not by characterizing learning algorithms, but by characterizing a learning *problem*. Any algorithm that is well suited to solving that problem we consider to be a reinforcement learning algorithm. A full specification of the reinforcement learning problem in terms of optimal control of Markov decision processes must wait until Chapter 3, but the basic idea is simply to capture the most important aspects of the real problem facing a learning agent interacting with its environment to achieve a goal. Clearly such an agent must be able to sense the state of the environment to some extent and must be able ato take actions that affect that state. The agent must also have a goal or goals relating to the state of the environment. Our formulation is intended to include just these three aspects---sensation, action, and goal---in the simplest possible form without trivializing any of them.

Reinforcement learning is different from supervised learning, the kind of learning studied in most current research in machine learning, statistical pattern recognition, and artificial neural networks. Supervised learning is learning from examples provided by some knowledgable external supervisor. This is an important kind of learning, but alone it is not adequate for learning from interaction. In interactive problems it is often impractical to obtain examples of desired behavior that are both correct and representative of all the situations in which the agent has to act. In uncharted territory---where one would expect learning to be most beneficial---an agent must be able to learn from its own experience.

One of the challenges that arises in reinforcement learning and not in other kinds of learning is the tradeoff between exploration and exploitation. To obtain a lot of reward, a reinforcement learning agent must prefer actions that it has tried in the past and found to

be effective in producing reward. But to discover such actions it has to try actions that it has not selected before. The agent has to *exploit* what it already knows in order to obtain reward, but it also has to *explore* in order to make better action selections in the future. The dilemma is that neither exploitation nor exploration can be pursued exclusively without failing at the task. The agent must try a variety of actions *and* progressively favor those that appear to be best. On a stochastic task, each action must be tried many times to reliably estimate its expected reward. The exploration--exploitation dilemma has been intensively studied by mathematicians for many decades (see Chapter 2). For now we simply note that the entire issue of balancing exploitation and exploration does not even arise in supervised learning as it is usually defined.

Another key feature of reinforcement learning is that it explicitly considers the *whole* problem of a goal-directed agent interacting with an uncertain environment. This is in contrast with many approaches that address subproblems without addressing how they might fit into a larger picture. For example, we have mentioned that much of machine learning research is concerned with supervised learning without explicitly specifying how such an ability would finally be useful. Other researchers have developed theories of planning with general goals, but without considering planning's role in real-time decision-making, or the question of where the predictive models necessary for planning would come from. Although these approaches have yielded many useful results, their focus on isolated subproblems is a significant limitation.

Reinforcement learning takes the opposite tack, by starting with a complete, interactive, goal-seeking agent. All reinforcement learning agents have explicit goals, can sense aspects of their environments, and can choose actions to influence their environments. Moreover, it is usually assumed from the beginning that the agent has to operate despite significant uncertainty about the environment it faces. When reinforcement learning involves planning, it has to address the interplay between planning and real-time action selection, as well as the question of how environmental models are acquired and improved. When reinforcement learning involves supervised learning, it does so for very specific reasons that determine which capabilities are critical, and which are not. For learning research to make progress, important subproblems have to be isolated and studied, but they should be subproblems that are motivated by clear roles in complete, interactive, goal-seeking agents, even if all the details of the complete agent cannot yet be filled in.

One of the larger trends of which reinforcement learning is a part is that towards greater contact between artificial intelligence and other engineering disciplines. Not all that long ago, artificial intelligence was viewed as almost entirely separate from control theory and statistics. It had to do with logic and symbols, not numbers. Artificial intelligence was large LISP programs, not linear algebra, differential equations, or statistics. Over the last decades this view has gradually eroded. Modern artificial intelligence researchers accept statistical and control-theory algorithms, for example, as relevant competing methods or

simply as tools of their trade. The previously ignored areas lying between artificial intelligence and conventional engineering are now among the most active of all, including new fields such as neural networks, intelligent control, and our topic, reinforcement learning. In reinforcement learning we extend ideas from optimal control theory and stochastic approximation to address the broader and more ambitious goals of artificial intelligence.

Richard Sutton
Sat May 31 14:27:51 EDT 1997

1.2 Examples

A good way to understand reinforcement learning is to consider some of the examples and possible applications that have guided its development:

- 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.

- 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.

- A gazelle calf struggles to its feet minutes after being born. Half an hour later it is running at 30 miles per hour.

- 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.

- 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 the box. 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.

These examples share features that are so basic that they are easy to overlook. All involve *interaction* between an active decision-making agent and its environment, within which the agent seeks to achieve a *goal* despite *uncertainty* about its 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 actions, and thus may require foresight or planning.

At the same time, in all these examples the effects of actions cannot be fully predicted, and so the agent must frequently monitor its environment and react appropriately. For example, Phil must watch the milk he pours into his cereal bowl to keep it from overflowing. All these examples involve goals that are explicit in the sense that the agent can judge progress toward its goal on the basis of what it can directly sense. The chess player knows whether or not he wins, the refinery controller knows how much petroleum is being produced, the mobile robot knows when its batteries run down, and Phil knows whether or not he is enjoying his breakfast.

In all of these examples the agent can use its experience to improve its performance over time. The chess player refines the intuition he uses to evaluate positions, thereby improving his play; the gazelle calf improves the efficiency with which it can run; Phil learns to streamline his breakfast making. The knowledge the agent brings to the task at the start---either from previous experience with related tasks or built into it by design or evolution---influences what is useful or easy to learn, but interaction with the environment is essential for adjusting behavior to exploit specific features of the task.

Richard Sutton
Sat May 31 14:27:51 EDT 1997

1.3 Elements of Reinforcement Learning

Beyond the agent and the environment, one can identify four main sub-elements to a reinforcement learning system: a *policy*, a *reward function*, a *value function*, and, optionally, a *model* of the environment.

A *policy* defines the learning agent's way of behaving at a given time. Roughly speaking, a policy is a mapping from perceived states of the environment to actions to be taken when in those states. It corresponds to what in psychology would be called a set of stimulus-response rules or associations. In some cases the policy may be a simple function or lookup table, whereas in others it may involve extensive computation such as a search process. The policy is the core of a reinforcement learning agent in the sense that it alone is sufficient to determine behavior. In general, policies may be stochastic.

A *reward function* defines the goal in a reinforcement learning problem. Roughly speaking, it maps perceived states (or state-action pairs) of the environment to a single number, a *reward*, indicating the intrinsic desirability of the state. A reinforcement-learning agent's sole objective is to maximize the total reward it receives in the long run. The reward function defines what are the good and bad events for the agent. In a biological system, it would not be inappropriate to identify rewards with pleasure and pain. They are the immediate and defining features of the problem faced by the agent. As such, the reward function must necessarily be fixed. It may, however, be used as a basis for changing the policy. For example, if an action selected by the policy is followed by low reward then the policy may be changed to select some other action in that situation in the future. In general, reward functions may also be stochastic.

Whereas a reward function indicates what is good in an immediate sense, a *value function* specifies what is good in the long run. Roughly speaking, the *value* of a state is the total amount of reward an agent can expect to accumulate over the future starting from that state. Whereas rewards determine the immediate, intrinsic desirability of environmental states, values indicate the *long-term* desirability of states after taking into account the states that are likely to follow, and the rewards available in those states. For example, a

state might always yield a low immediate reward, but still have a high value because it is regularly followed by other states that yield high rewards. Or the reverse could be true. To make a human analogy, rewards are like pleasure (if high) and pain (if low), whereas values correspond to a more refined and far-sighted judgment of how pleased or displeased we are that our environment is in a particular state. Expressed this way, we hope it is clear that value functions formalize a very basic and familiar idea.

Rewards are in a sense primary, whereas values, as predictions of rewards, are secondary. Without rewards there could be no values, and the only purpose of estimating values is to achieve more reward. Nevertheless, it is values with which we are most concerned when making and evaluating decisions. Action choices are made on the basis of value judgments. We seek actions that bring about states of highest value, not highest reward, because these actions obtain for us the greatest amount of reward over the long run. In decision-making and planning, the derived quantity called value is the one with which we are most concerned. Unfortunately, it is also much harder to determine values than it is to determine rewards. Rewards are basically given directly by the environment, but values must be estimated and re-estimated from the sequences of observations an agent makes over its entire lifetime. In fact, the most important component of almost all reinforcement learning algorithms is a method for efficiently estimating values. The importance and centrality of estimating values is perhaps the most important thing we have learned about reinforcement learning in the last two decades.

Although all the reinforcement learning methods we consider in this book are structured around estimating value functions, it is not strictly necessary to do this to solve reinforcement learning problems. For example, search methods such as genetic algorithms, genetic programming, simulated annealing, and other function optimization methods, have been used to solve reinforcement learning problems. These methods search directly in the space of policies without ever appealing to value functions. We call these *evolutionary* methods because their operation is analogous to how biological evolution produces organisms with skilled behavior even though they do not themselves learn during their individual lifetimes. If the space of policies is sufficiently small, or can be structured so that good policies are common or easy to find, then evolutionary methods are often effective. In addition, evolutionary methods have advantages on problems in which the learning agent cannot accurately sense the state of its environment.

Nevertheless, what we mean by reinforcement learning involves learning while interacting with the environment, which evolutionary methods do not do. It is our belief that methods able to take advantage of the details of individual behavioral interactions can be much more efficient than evolutionary methods in a great many cases. Evolutionary methods ignore much of the useful structure of the reinforcement learning problem: they do not use the fact that the policy they are searching for is a function from states to actions; they do not notice which states an individual passes through during its lifetime, or which actions it selects. In some cases this information can be misleading (e.g., when states are mis-

perceived), but more often it should enable more efficient search. Although evolution and learning share many features and can naturally work together as they do in nature, we do not consider evolutionary methods by themselves to be especially well-suited to reinforcement learning problems. For simplicity, in this book when we use the term ``reinforcement learning'' we do not include evolutionary methods.

The fourth and final element of some reinforcement learning systems is a *model* of the environment. This is something that mimics the behavior of the environment. For example, given a state and action, the model might predict the resultant next state and next reward. Models are used for *planning*, by which we mean any way of deciding on a course of action by considering possible future situations before they are actually experienced. The incorporation of models and planning into reinforcement learning systems is a relatively new development. Early reinforcement learning systems were explicitly trial-and-error learners; what they did was viewed as almost the *opposite* of planning. Nevertheless, it gradually became clear that reinforcement learning methods are closely related to dynamic programming methods, which do use models, and that they in turn are closely related to state-space planning methods. In Chapter 9 we explore reinforcement learning systems that simultaneously learn by trial and error, learn a model of the environment, and use the model for planning. Modern reinforcement learning spans the spectrum from low-level, trial-and-error learning to high-level, deliberative planning.

Richard Sutton
Sat May 31 14:27:51 EDT 1997

1.4 An Extended Example: Tic-Tac-Toe

To illustrate the general idea of reinforcement learning and contrast it with other approaches, we next consider a single example in more detail.

Consider the familiar child's game of Tic-Tac-Toe. Two players take turns playing on a three-by-three board. One player plays X s and the other Ø s until one player wins by placing three of his marks in a row, horizontally, vertically, or diagonally, as the `X ' player has in this game:

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 that he never loses, let us assume that we are playing against an imperfect player, one whose play is sometimes incorrect and allows us to win. For the moment, in fact, let us consider draws and losses to be equally bad for us. How might we construct a player that will find the imperfections in its opponent's play and learn to maximize its chances of winning?

Although this is a very simple problem, it cannot readily be solved in a satisfactory way by classical techniques. For example, the classical ``minimax'' solution from game theory is not correct here because it assumes a particular way of playing by the opponent. For example, a minimax player 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, can *compute* an optimal solution for any opponent, but require as input a complete specification of that opponent, including the probabilities with which the opponent makes each move in each board state. Let us assume that 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 on this problem is to first learn 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. In the end, this is not that different from some of the reinforcement learning methods we examine later in this book.

An evolutionary approach to this problem would directly search the space of possible policies for one with a high probability of winning against the opponent. Here, a policy is a rule that tells the player what move to make for every state of the game---every possible configuration of X s and Ø s on the three-by-three board. For each policy considered, an estimate of its winning probability would be obtained by playing some number of games against the opponent. This evaluation would then direct which policy or policies were next considered. A typical evolutionary method would hillclimb in policy space, successively generating and evaluating policies in an attempt to obtain incremental improvements. Or, perhaps, a genetic-style algorithm could be used which would maintain and evaluate a population of policies. Literally hundreds of different optimization algorithms could be applied. By *directly* searching the policy space we mean that *entire policies* are proposed and compared on the basis of scalar evaluations.

Here is how the Tic-Tac-Toe problem would be approached using reinforcement learning and approximate value functions. First we set up a table of numbers, one for each possible state of the game. Each number will be the latest estimate of the probability of our winning from that state. We treat this estimate as the state's current *value*, and the whole table is the learned value function. State A has higher value than state B, or is considered ``better" than state B, if the current estimate of the probability of our winning from A is higher than it is from B. 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 Ø s in a row, or that are ``filled up", the correct probability is 0, as we cannot win from them. We set the initial values of all the other states, the nonterminals, to 0.5, representing an informed guess that we have a 50% chance of winning.

Now we play 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 current values in the table. Most of the time we move *greedily*, selecting the move that leads to the state with greatest value, that is, 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 we might otherwise never see. A sequence of moves made and considered during a game can be diagrammed as in Figure 1.1.
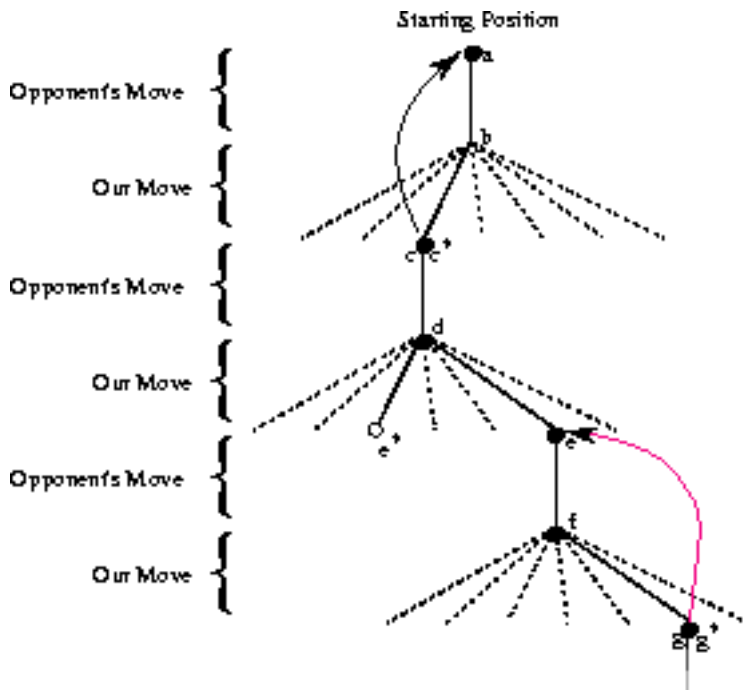
Figure 1.1: A sequence of Tic-Tac-Toe moves. The solid lines represent the moves taken during a game; the dashed lines represent moves that we (our algorithm) considered but did not make. Our second move was an exploratory move, meaning that is was taken even though another sibling move, that leading to e^*, was ranked higher. Exploratory moves do not result in any learning, but each of our other moves do, causing *backups* as suggested by the curved arrows and detailed in the text.

While we are playing, we change the values of 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. To do this, we ``back up" the value of the state after each greedy move to the state before the move, as suggested by the arrows in Figure 1.1. More precisely, the current value of the earlier state is adjusted to be closer to the value of the later state. This can be done by moving the earlier state's value a fraction of the way toward the value of the later state. If we let **s** denote the state before the greedy move, and s' the state after, then the update to the estimated value of **s**, denoted $V(s)$, can be written:

$$V(s) \leftarrow V(s) + \alpha[V(s') - V(s)],$$

where α is a small positive fraction called the *step-size parameter*, which influences the rate of learning. This update rule is an example of a *temporal-difference* learning method, so called because it's changes are based on a difference, $V(s') - V(s)$, between estimates at two different times.

The method described above performs quite well on this task. For example, if the step-size parameter is reduced properly over time, this method converges, for any fixed opponent,

to the true probabilities of winning from each state given optimal play by our player. Furthermore, the moves then taken (except on exploratory moves) are in fact the optimal moves against the opponent. In other words, the method converges to an optimal policy for playing the game. If the step-size parameter is not reduced all the way to zero over time, then this player also plays well against opponents that change their way of playing slowly over time.

This example illustrates the differences between evolutionary methods and methods that learn value functions. To evaluate a policy, an evolutionary method must hold it fixed and play many games against the opponent, or simulate many games using a model of the opponent. The frequency of wins gives an unbiased estimate of the probability of winning with that policy, and can be used to direct the next policy selection. But each policy change is made only after many games, and only the final outcome of each game is used: what happens *during* the games is ignored. For example, if the player wins, then *all* of its behavior in the game is given credit, independently of how specific moves might have been critical to the win. Credit is even given to moves that never occurred! Value function methods, in contrast, allow individual states to be evaluated. In the end, both evolutionary and value function methods search the space of policies, but learning a value function takes advantage of information available during the course of play.

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. For example, the simple reinforcement learning player would learn to set up multi-move traps for a short-sighted opponent. It is a striking feature of the reinforcement learning solution that it can achieve the effects of planning and lookahead without using a model of the opponent and without carrying out an explicit search over possible sequences of future states and actions.

While this example illustrates some of the key features of reinforcement learning, it is so simple that it might give the impression that reinforcement learning is more limited than it really is. Although Tic-Tac-Toe is a two-person game, reinforcement learning also applies when there is no explicit external adversary, that is, in the case of a ``game against nature.'' \ Reinforcement learning is also not restricted to problems in which behavior breaks down into separate episodes, like the separate games of Tic-Tac-Toe, with reward only at the end of each episode. It is just as applicable when behavior continues indefinitely and when rewards of various magnitudes can be received at any time.

Tic-Tac-Toe has a relatively small, finite state set, whereas reinforcement learning can be applied to very large, or even infinite, state sets. For example, Gerry Tesauro (1992, 1995) combined the algorithm described above with an artificial neural network to learn to play the game of backgammon, which has approximately 10^{20} states. Notice that with this

many states it is impossible to ever experience more than a small fraction of them. Tesauro's program learned to play far better than any previous program, and now plays at the level of the world's best human players (see Chapter 11). The neural network provides the program with the ability to generalize from its experience, so that in new states it selects moves based on information saved from similar states faced in the past, as determined by its network. How well a reinforcement learning agent can work in problems with such large state sets is intimately tied to how appropriately it can generalize from past experience. It is in this role that we have the greatest need for supervised learning methods with reinforcement learning. Neural networks are not the only, or necessarily the best, way to do this.

In this Tic-Tac-Toe example, learning started with no prior knowledge beyond the rules of the game, but reinforcement learning by no means entails a *tabula rasa* view of learning and intelligence. On the contrary, prior information can be incorporated into reinforcement learning in a variety of ways that can be critical for efficient learning. We also had access to the true state in the Tic-Tac-Toe example, whereas 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. That case, however, is substantially more difficult, and we do not cover it significantly in this book.

Finally, the Tic-Tac-Toe player was able to look ahead and know the states that would result from each of its possible moves. To do this, it had to have a model of the game that allows it to ``think about'' how its environment would change in response to moves that it may never make. Many problems are like this, but in others even a short-term model of the effects of actions is lacking. Reinforcement learning can be applied in either case. No model is required, but models can easily be used if they are available or can be learned.

Exercise 1.1

Self Play. Suppose, instead of playing against a random opponent, the reinforcement learning algorithm described above played against itself. What do you think would happen in this case? Would it learn a different way of playing?

Exercise 1.2

Symmetries. Many Tic-Tac-Toe positions appear different but are really the same because of symmetries. How might we amend the reinforcement learning algorithm described above to take advantage of this? In what ways would this improve it? Now think again, suppose the opponent did not take advantage of symmetries. In that case, should we? It is true then that symmetrically equivalent positions should necessarily have the same value?

Exercise 1.3 *Greedy Play*. Suppose the reinforcement learning player was *greedy*, that is, it always played the move that brought it to the position that it rated the best. Would it

learn to play better, or worse, than a non-greedy player? What problems might occur?

Exercise 1.4

Learning from Exploration. Suppose learning updates occurred after *all* moves, including exploratory moves. If the step-size parameter is reduced over time appropriately, then the state values would converge to a set of probabilities. What are the two sets of probabilities computed when we do, and when we do not, learn from exploratory moves? Assuming that we do continue to make exploratory moves, which set of probabilities might be better to learn? Which would result in more wins?

Exercise 1.5

Other Improvements. Can you think of other ways to improve the reinforcement learning player? Can you think of any better way to solve the Tic-Tac-Toe problem as posed?

Richard Sutton
Sat May 31 14:27:51 EDT 1997

1.5 Summary

Reinforcement learning is a computational approach to understanding and automating goal-directed learning and decision-making. It is distinguished from other computational approaches by its emphasis on learning by the individual from direct interaction with its environment, without relying on exemplary supervision or complete models of the environment. In our opinion, reinforcement learning is the first field to seriously address the computational issues that arise when learning from interaction with an environment in order to achieve long-term goals.

Reinforcement learning uses a formal framework defining the interaction between agent and environment in terms of states, actions, and rewards. This framework is intended to be a simple way of representing essential features of the artificial intelligence problem. These features include a sense of cause and effect, of uncertainty and nondeterminism, and the existence of explicit goals. Most relevant is the formalism of Markov decision processes, which provides a precise, and relatively neutral, way of including the key features.

The concepts of value and value functions are the key features of the reinforcement learning methods that we consider in this book. We take the position that value functions are essential for efficient search in the space of policies. Their use of value functions distinguish reinforcement learning methods from evolutionary methods that search directly in policy space guided by scalar evaluations of entire policies.

Richard Sutton
Sat May 31 14:27:51 EDT 1997

1.6 History of Reinforcement Learning

The history of reinforcement learning has two main threads, both long and rich, which were pursued independently before intertwining in modern reinforcement learning. One thread concerns learning by trial and error and started in the psychology of animal learning. This thread runs through some of the earliest work in artificial intelligence and led to the revival of reinforcement learning in the early 1980s. The other thread concerns the problem of optimal control and its solution using value functions and dynamic programming. For the most part, this thread did not involve learning. Although the two threads were largely independent, the exceptions revolve around a third, less distinct thread concerning temporal-difference methods such as used in the Tic-Tac-Toe example in this chapter. All three threads came together in the late 1980s to produce the modern field of reinforcement learning as we present it in this book.

The thread focusing on trial-and-error learning is the one with which we are most familiar and about which we have the most to say in this brief history. Before doing that, however, we briefly discuss the optimal control thread.

The term ``optimal control'' came into use in the late 1950s to describe the problem of designing a controller to minimize a measure of a dynamical system's behavior over time. One of the approaches to this problem was developed in the mid-1950s by Richard Bellman and colleagues by extending a 19th century theory of Hamilton and Jacobi. This approach uses the concept of a dynamical system's state and of a value function, or ``optimal return function,'' to define a functional equation, now often called the Bellman equation. The class of methods for solving optimal control problems by solving this equation came to be known as dynamic programming (Bellman, 1957a). Bellman (1957b) also introduced the discrete stochastic version of the optimal control problem known as Markovian decision processes (MDPs), and

Ron Howard (1960) devised the policy iteration method for MDPs. All of these are essential elements underlying the theory and algorithms of modern reinforcement learning.

Dynamic programming is widely considered the only feasible way of solving general stochastic optimal control problems. It suffers from what Bellman called ``the curse of dimensionality,'' meaning that its computational requirements grow exponentially with the number of state variables, but it is still far more efficient and more widely applicable than any other method. Dynamic programming has been extensively developed in the last four decades, including extensions to partially observable MDPs (surveyed by Lovejoy, 1991), many applications (surveyed by White, 1985, 1988, 1993), approximation methods (surveyed by Rust, 1996), and asynchronous methods (Bertsekas, 1982, 1983). Many excellent modern treatments of dynamic programming are available (e.g., Bertsekas, 1995; Puterman, 1994; Ross, 1983; and Whittle, 1982, 1983). Bryson (1996) provides a detailed authoritative history of optimal control.

In this book, we consider all of the work on optimal control to also be work in reinforcement learning. We define reinforcement learning as any effective way of solving reinforcement learning problems, and it is now clear that these problems are very closely related to optimal control problems, particularly those formulated as MDPs. Accordingly we must consider the solution methods of optimal control, such as dynamic programming, to also be reinforcement learning methods. Of course, almost all of these methods require complete knowledge of the system to be controlled, and for this reason it feels a little unnatural to say that they are part of reinforcement *learning*. On the other hand, many dynamic programming methods are incremental and iterative. Like true learning methods, they gradually reach the correct answer through successive approximations. As we show in the rest of this book, these similarities are far more than superficial. The theories and solution methods for the cases of complete and incomplete knowledge are so closely related that we feel they must be considered together as part of the same subject matter.

Let us return now to the other major thread leading to the modern field of reinforcement learning, that centered around the idea of trial-and-error learning. This thread began in psychology, where ``reinforcement'' theories of learning are common. Perhaps the first to succintly express the essence of trial-and-error learning was Edward Thorndike. We take this essence to be the idea that actions followed by good or bad outcomes have their tendency to be re-selected altered accordingly. In Thorndike's words:

> Of several responses made to the same situation, those which are accompanied or closely followed by satisfaction to the animal will, other things being equal, be more firmly connected with the situation, so that, when it recurs, they will be more likely to recur; those which are accompanied or closely followed by discomfort to the animal will, other things being equal, have their connections with that situation weakened, so that, when it recurs, they will be less likely to occur. The greater the satisfaction or discomfort, the greater the strengthening or weakening of the bond. (Thorndike, 1911, p. 244)

Thorndike called this the ``Law of Effect'' because it describes the effect of reinforcing events on the tendency to select actions. Although sometimes controversial (e.g., see Kimble, 1961, 1967; Mazur, 1994) the Law of Effect is widely regarded as an obvious basic principle underlying much behavior (e.g., Hilgard and Bower, 1975; Dennett, 1978; Campbell, 1958; Cziko, 1995).

The Law of Effect includes the two most important aspects of what we mean by trial-and-error learning. First, it is *selectional* rather than instructional in the sense of Monod, meaning that it involves trying alternatives and selecting among them by comparing their consequences. Second, it is *associative*, meaning that the alternatives found by selection are associated with particular situations. Natural selection in evolution is a prime example of a selectional process, but it is not associative. Supervised learning is associative, but not selectional. It is the combination of these two that is essential to the Law of Effect and to trial-and-error learning. Another way of saying this is that the Law of Effect is an elementary way of combining *search* and *memory*: search in the form of trying and selecting among many actions in each situation, and memory in the form of remembering what actions worked best, associating them with the situations in which they were best. Combining search and memory in this way is essential to reinforcement learning.

In early artificial intelligence, before it was distinct from other branches of engineering, several researchers began to explore trial-and-error learning as an engineering principle. The earliest computational investigations of trial-and-error learning were perhaps by Minsky and Farley and Clark, both in 1954. In his Ph.D. dissertation, Minsky discussed computational models of reinforcement learning and described his construction of an analog machine, composed of components he called SNARCs (Stochastic Neural-Analog Reinforcement Calculators). Farley and Clark described another neural-network learning machine designed to learn by trial-and-error. In the 1960s one finds the terms ``reinforcement'' and ``reinforcement learning'' being widely used in the engineering literature for the first time (e.g., Waltz and Fu, 1965; Mendel, 1966; Fu, 1970; Mendel and McClaren, 1970). Particularly influential was Minsky's paper ``Steps Toward Artificial Intelligence'' (Minsky, 1961), which discussed several issues relevant to reinforcment learning, including what he called the *credit-assignment problem*: how do you distribute credit for success among the many decisions that may have been involved in producing it? All of the methods we discuss in this book are, in a sense, directed toward solving this problem.

The interests of Farley and Clark (1954; Clark and Farley, 1955) shifted from trial-and-error learning to generalization and pattern recognition, that is, from reinforcement learning to supervised learning. This began a pattern of confusion about the relationship between these types of learning. Many researchers seemed to believe that they were studying reinforcement learning when they were actually studying supervised learning. For example, neural-network pioneers such as Rosenblatt (1958) and Widrow and Hoff (1960) were clearly motivated by reinforcement learning---they used the language of

rewards and punishments---but the systems they studied were supervised learning systems suitable for pattern recognition and perceptual learning. Even today, reseachers and textbooks often minimize or blur the distinction between these types of learning. Some modern neural-network textbooks use the term trial-and-error to describe networks that learn from training examples because they use error information to update connection weights. This is an understandable confusion, but it substantually misses the essential selectional character of trial-and-error learning.

Partly as a result of these confusions, research into genuine trial-and-error learning became rare in the the 1960s and 1970s. In the next few paragraphs we discuss some of the exceptions and partial exceptions to this trend.

One of these was the work by a New Zealand researcher named John Andreae. Andreae (1963) developed a system called STeLLA that learned by trial and error in interaction with its environment. This system included an internal model of the world and, later, an ``internal monologue'' to deal with problems of hidden state (Andreae, 1969a). Andreae's later work placed more emphasis on learning from a teacher, but still included trial-and error (Andreae, 1977). Unfortunately, Andreae's pioneering research was not well-known, and did not greatly impact subsequent reinforcement learning research.

More influential was the work of Donald Michie. In 1961 and 1963 he described a simple trial-and-error learning system for learning how to play Tic-Tac-Toe (or Noughts and Crosses) called MENACE (for Matchbox Educable Noughts and Crosses Engine). It consisted of a matchbox for each possible game position, each containing a number of colored beads, a color for each possible move from that position. By drawing a bead at random from the matchbox corresponding to the current game position, one could determine MENACE's move. When a game was over, beads were added or removed from the boxes used during play to reinforce or punish MENACE's decisions. Michie and Chambers (1968) described another Tic-Tac-Toe reinforcement learner called GLEE (Game Learning Expectimaxing Engine) and a reinforcement learning controller called BOXES. They applied BOXES to the task of learning to balance a pole hinged to a movable cart on the basis of a failure signal occuring only when the pole fell or the cart reached the end of a track. This task was adapted from the earlier work of Widrow and Smith (1964), who used supervised learning methods, assuming instruction from a teacher already able to balance the pole. Michie and Chamber's version of pole-balancing is one of the best early examples of a reinforcement learning task under conditions of incomplete knowledge. It influenced much later work in reinforcement learning, beginning with some of our own studies (Barto, Sutton and Anderson, 1983; Sutton, 1984). Michie has consistently emphasized the role of trial-and-error and learning as essential aspects of artificial intelligence (Michie, 1974).

Widrow, Gupta, and Maitra (1973) modified the LMS algorithm of Widrow and Hoff (1960) to produce a reinforcement learning rule that could learn from success and failure

signals instead of from training examples. They called this form of learning ``selective bootstrap adaptation'' and described it as ``learning with a critic'' instead of ``learning with a teacher.'' They analyzed this rule and showed how it could learn to play blackjack. This was an isolated foray into reinforcement learning by Widrow, whose contributions to supervised learning were much more influential.

Research on *learning automata* had a more direct influence on the trial-and-error thread of modern reinforcement learning research. These were methods for solving a non-associative, purely selectional learning problem known as the *n -armed bandit* by anology to a slot-machine, or ``one-armed bandit,'' except with **n** levers (see Chapter 2). Learning automata were simple, low-memory machines for solving this problem. Tsetlin (1973) introduced these machines in the west, and excellent surveys are provided by Narendra and Thatachar (1974, 1989). Barto and Anadan (1985) extended these methods to the associative case.

John Holland (1975) outlined a general theory of adaptive systems based on selectional principles. His early work concerned trial and error primarily in its non-associative form, as in evolutionary methods and the **n** -armed bandit. In 1986 he introduced *classifier systems*, true reinforcement-learning systems including association and value functions. A key component of Holland's classifer systems was always a *genetic algorithm*, an evolutionary method whose role was to evolve useful representations. Classifier systems have been extensively developed by many researchers to form a major branch of reinforcement learning research (e.g., see text by Goldberg, 1989; and Wilson, 1994), but genetic algorithms---which by themselves are not reinforcement learning systems---have received much more attention.

The individual most responsible for reviving the trial-and-error thread to reinforcement learning within artificial intelligence is Harry Klopf (1972, 1975, 1982). Klopf recognized that essential aspects of adaptive behavior were being lost as learning researchers came to focus almost exclusively on supervised learning. What was missing, according to Klopf, were the hedonic aspects of behavior, the drive to achieve some result from the environment, to control the environment toward desired ends and away from undesired ends. This is the essential idea of reinforcement learning. Klopf's ideas were especially influential on the authors because our assessment of them (Barto and Sutton, 1981b) led to our appreciation of the distinction between supervised and reinforcement learning, and to our eventual focus on reinforcement learning. Much of the early work that we and colleagues accomplished was directed toward showing that reinforcement learning and supervised learning were indeed very different (Barto, Sutton, and Brouwer, 1981; Barto and Sutton, 1981a; Barto and Anandan, 1985). Other studies showed how reinforcement learning could address important problems in neural-network learning, in particular, how it could produce learning algorithms for multi-layer networks (Barto, Anderson, and Sutton, 1982; Barto and Anderson, 1985; Barto and Anandan, 1985; Barto, 1985; Barto, 1986; Barto and Jordan, 1986).

We turn now to the third thread to the history of reinforcement learning, that concerning temporal-difference learning. Temporal-difference learning methods are distinctive in being driven by the difference between temporally successive estimates of the same quantity, for example, of the probability of winning in the Tic-Tac-Toe example. This thread is smaller and less distinct than the other two, but has played a particularly important role in the field, in part because temporal-difference methods seem to be new and unique to reinforcement learning.

The origins of temporal-difference learning are in part in animal learning psychology, in particular, in the notion of *secondary reinforcers*. A secondary reinforcer is a stimulus that has been paired with a primary reinforcer such as food or pain and, as a result, has come to take on similar reinforcing properties. Minsky (1954) may have been the first to realize that this psychological principle could be important for artificial learning systems. Arthur Samuel (1959) was the first to propose and implement a learning method that included temporal-difference ideas, as part of his celebrated checkers-playing program. Samuel made no reference to Minsky's work or to possible connections to animal learning. His inspiration apparently came from Claude Shannon's (1950a) suggestion that a computer could be programmed to use an evaluation function to play chess, and that it might be able to to improve its play by modifying this function online. (It is possible that these ideas of Shannon also influenced Bellman, but we know of no evidence for this.) Minsky (1961) extensively discussed Samuel's work in his ``Steps'' paper, suggesting the connection to secondary reinforcement theories, natural and artificial.

As we have discussed, in the decade following the work of Minsky and Samuel, little computational work was done on trial-and-error learning, and apparently no computational work at all was done on temporal-difference learning.

In 1972, Klopf brought trial-and-error learning together with an important component of temporal-difference learning. Klopf was interested in principles that would scale to learning in very large systems, and thus was intrigued by notions of local reinforcement, whereby subcomponents of an overall learning system could reinforce one another. He developed the idea of ``generalized reinforcement,'' whereby every component (nominally, every neuron) viewed all of its inputs in reinforcement terms, excitatory inputs as rewards and inhibitory inputs as punishments. This is not the same idea as what we now know as temporal-difference learning, and in retrospect it is farther from it than was Samuel's work. Nevertheless, Klopf's work also linked the idea with trial-and-error learning and with the massive empirical database of animal learning psychology.

Sutton (1978a--c) developed Klopf's ideas further, particularly the links to animal learning theories, describing learning rules driven by changes in temporally successive predictions. He and Barto refined these ideas and developed a psychological model of classical conditioning based on temporal-difference learning (Sutton and Barto, 1981; Barto and Sutton, 1982). There followed several other influential psychological models of classical

conditioning based on temporal-difference learning (e.g., Klopf, 1988; Moore et al., 1986; Sutton and Barto, 1987, 1990). Some neuroscience models developed at this time are well interpreted in terms of temporal-difference learning (Hawkins and Kandel, 1984; Byrne, 1990; Gelperin et al., 1985; Tesauro, 1986; Friston et al, 1994), although in most cases there was no historical connection. A recent summary of links between temporal-difference learning and neuroscience ideas is provided by Schultz, Dayan, and Montague, 1997).

In our work on temporal-difference learning up until 1981 we were strongly influenced by animal learning theories and by Klopf's work. Relationships to Minsky's ``Steps" paper and to Samuel's checkers players appear to have been recognized only afterwards. By 1981, however, we were fully aware of all the prior work mentioned above as part of the temporal-difference and trial-and-error threads. At this time we developed a method for using temporal-difference learning in trial-and-error learning, known as the actor-critic architecture, and applied this method to Michie and Chambers' pole-balancing problem (Barto, Sutton and Anderson, 1983). This method was extensively studied in Sutton's (1984) PhD thesis and extended to use backpropagation neural networks in Anderson's (1986) PhD thesis. Around this time, Holland (1986) incorporated temporal-difference methods explicitly into his classifier systems. A key step was taken by Sutton in 1988 by separating temporal-difference learning from control, treating it as a general prediction method. That paper also introduced the TD(λ) \ algorithm and proved some of its convergence properties.

As we were finalizing our work on the actor-critic architecture in 1981, we discovered a paper by Ian Witten (1977) which contains the earliest known publication of a temporal-difference learning rule. He proposed the method that we now call TD(0) for use as part of an adaptive controller for solving MDPs. Witten's work was a descendant of Andreae's early experiments with STeLLA and other trial-and-error learning systems. Thus, Witten's 1977 paper spanned both major threads of reinforcement-learning research---trial-and-error and optimal control---while making a distinct early contribution to temporal-difference learning.

Finally, the temporal-difference and optimal control threads were fully brought together in 1989 with Chris Watkins's development of Q-learning. This work extended and integrated prior work in all three threads of reinforcement learning research. Paul Werbos (1987) also contributed to this integration by arguing for the convergence of trial-and-error learning and dynamic programming since 1977. By the time of Watkins's work there had been tremendous growth in reinforcement learning research, primarily in the machine learning subfield of artificial intelligence, but also in neural networks and artificial intelligence more broadly. In 1992, the remarkable success of Gerry Tesauro's backgammon playing program, TD-Gammon, brought additional attention to the field. Other important contributions made in the recent history of reinforcement learning are too numerous to mention in this brief account; we cite these at the end of the individual chapters in which

they arise.

Richard Sutton
Sat May 31 14:27:51 EDT 1997

1.7 Bibliographical Remarks

For additional general coverage of reinforcement learning, we refer the reader to the books by Bertsekas and Tsitsiklis (1996) and Kaelbling (1993). Two special issues of the journal *Machine Learning* focus on reinforcement learning: Sutton (1992) and Kaelbling (1996). Useful surveys are provided by Barto (1995), Kaelbling, Littman, and Moore (1996), and Keerthi and Ravindran (1997).

The example of Phil's breakfast in this chapter was inspired by Agre (1988). We refer the reader to Chapter 6 for references to the kind of temporal difference method we used in the Tic-Tac-Toe example.

Modern attempts to relate the kinds of algorithms used in reinforcement learning to the nervous system are made by Barto (1995), Friston et al. (1994), Hampson (1989), Houk et al. (1995), Montague et al. (1996), and Schultz et al. (1997).

Richard Sutton
Sat May 31 14:27:51 EDT 1997