Reinforcement Learning: Connections, Surprises, Challenges

Andrew G. Barto

■ *The idea of implementing reinforcement learning in a computer was one of the earliest ideas about the possibility of AI, but reinforcement learning remained on the margin of AI until relatively recently. Today we see reinforcement learning playing essential roles in some of the most impressive AI applications. This article presents observations from the author's personal experience with reinforcement learning over the most recent 40 years of its history in AI, focusing on striking connections that emerged between largely separate disciplines and on some of the findings that surprised him along the way. These connections and surprises place reinforcement learning in a historical context, and they help explain the success it is finding in modern AI. The article concludes by discussing some of the challenges that need to be faced as reinforcement learning moves out into real world.*

The idea of implementing reinforcement learning (RL) in a computer was one of the earliest ideas about the possibility of AI. In a 1948 report, Alan Turing described a design for a pleasure-pain system:

> When a configuration is reached for which the action is undetermined, a random choice for the missing data is made and the appropriate entry is made in the description, tentatively, and is applied. When a pain stimulus occurs all tentative entries are cancelled, and when a pleasure stimulus occurs they are all made permanent. (Turing [1948] 2004, 425)

Turing did little to develop this idea, and it was not until the year of his death, 1954, that Wesley Clark and Belmont Farley simulated RL in a neural net on a digital computer (Farley and Clark 1954). In the same year, Marvin Minsky described an analog RL neural net in his Princeton PhD dissertation (Minsky 1954). There were earlier ingenious RL devices, though electromechanical rather than computer implementations, including Claude Shannon's maze-run-

ning mouse, Theseus, that used a kind of RL to find its way through a maze (Shannon 1951). In the 70 years since Turing's report, mathematical formulations of RL have appeared in fields such as psychology, economics, and control engineering. RL algorithms known as *learning automata* date back to the early 1960s and the work of the Russian mathematician and physicist M. L. Tsetlin (published posthumously in Tsetlin [1973]; surveyed by Narendra and Thathachar [1989]).

Despite featuring prominently in Minsky's famous "Steps" paper (Minsky 1961), and despite the extensive mathematical study of algorithms like learning automata, RL largely remained on the margin of AI until relatively recently. Today we see RL playing essential roles in some of the most impressive applications of machine learning (ML), including DeepMind's Go-playing programs (Silver, Huang, et al. 2016; Silver, Schrittwieser, et al. 2017).

Instead of recounting this history, my aim here is to focus on observations from my personal experience with RL over the most recent 40 years of this history. Though personal, these observations will be of general interest, I believe, because they are instructive about RL's place in AI and its promise for future developments. RL has continued to fascinate me for this long — even in the face of skeptics and naysayers — for two major reasons. First, the study of RL has exposed deep connections between largely separate disciplines, ranging from computer science and engineering to psychology and neuroscience. More than any one discovery, or collection of discoveries, this rich fabric of interconnected facts and ideas has improved our understanding of both the human-made and the natural worlds. The second reason I have stuck with RL for so long is that studying it has surprised me in several interesting and instructive ways. I have had to revise preconceptions in some instances; in others, unexpected new insights — at least new to me — emerged from the results of computational explorations. Here, I attempt to convey a sense of the richness of this fabric by describing the most striking connections and surprises. Finally, I discuss some of the challenges that need to be faced in the future.

First, a bit of background. In the late 1970s I had the opportunity to work as a postdoc on a project aimed at assessing the scientific merit of a hypothesis proposed by physiologist A. Harry Klopf, a senior scientist with the Avionics Directorate of the Air Force Office of Scientific Research. Klopf hypothesized that neurons, the major components of our brains, are individually hedonists that work to maximize a local analog of pleasure while minimizing a local analog of pain (Klopf 1972, 1982). Under the direction of Michael Arbib, William Kilmer, and Nico Spinelli, professors at the University of Massachusetts, Amherst, and founders of the Cybernetics Center for Systems Neuroscience, a farsighted center focusing on the intersection of neuroscience and AI, and later joined by graduate student Richard Sutton, we explored the early history of learning in AI, including connections to theories of animal learning from psychology and theories about the neural machinery underlying learning. Were Klopf's ideas novel, and were they worth pursuing?

Klopf's hypothesis is basically that the synaptic weights of neurons change with experience according to a neuron-local version of what is known in psychology as the law of effect, proposed by the psychologist Edward Thorndike, who stated it as follows:

> 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, 244)

This law is controversial in animal learning theory and has been modified in various ways to better account for data (such as differences between the effects of reward and the effects of punishment), but it is widely regarded as a basic principle underlying a lot of behavior. It describes the commonsense learning mechanism widely known as learning by trial and error. This is the basic principle of RL.

RL Is Neither Supervised nor Unsupervised Learning

Our first surprise in exploring Klopf's idea was discovering that ML researchers did not capture key properties of trial-and-error learning, despite their clear intent to implement this form of learning in a computer. With the exception of the early work mentioned in the preceding section and that of a few others, genuine trial-and-error learning largely disappeared from ML research until modern RL came on the scene. How can this be? Is not trial-and-error learning at the root of most of today's ML, including the current deep learning algorithms, which learn by correcting errors? The answer is that trial-and-error learning is not the same as error-correction learning. The differences are subtle and important. This distinction is now understood by most ML researchers, but it is worth going into a bit of detail because it is still a source of confusion.

An error, as we use the term in ML, is a measure of the mismatch between a learning system's output and what that output should have been as given by the label in a labeled training example. If the actual and desired outputs are numbers, this measure is usually the numerical difference between them. If the

system's outputs are class labels in a classification problem, the measure might be just a binary indication of whether the output was correct or not. (Errors can also be lists, technically vectors, of numbers or binary indicators when the actual and desired outputs have many components, as in a neural network with many output units.) In either case, the error computation uses knowledge of both the actual and desired outputs of the learning system. This is the basic supervised learning paradigm.

On the other hand, an error in trial-and-error learning as expressed by the law of effect indicates a degree of satisfaction or discomfort. In RL, it is usually a number, which we call the *reward signal*, and we think of it as being generated by a critic. A reward signal might have a sign (though it does not always need one), with a plus or minus respectively indicating a degree of satisfaction or discomfort. Where an error is a measure of a mismatch between a desired and an actual output, a reward signal is an evaluation of the RL system's output, or action. The difference is important. An evaluation might be the result of comparing the system's action with a given desired action, in which case the reward signal is derived from the mismatch. However, in general, the reward signal need not be based on any knowledge of what the correct action should be. This is a key strength of RL.

To draw an analogy with a movie critic's evaluation of a movie, how many tomatoes, fresh or rotten, the critic bestows upon a movie could well be based on the critic's comparison of the movie with their idea of a better one, but this is not necessary. The critic might be responding to a general sense of satisfaction or dissatisfaction, without having a concrete idea of what the movie should have been or how it could have been better.

In fact, the critic does not even need to know the actions of the RL system in order to evaluate them. A reward signal can be determined from the consequences of the system's actions on some other system whose details are completely unknown to the critic. This other system is interposed between the RL system and the critic, translating the RL system's actions into patterns of behavior that the critic evaluates. Continuing the movie example, the critic might base their evaluation on the effect a movie had on another person, or on the consensus of a group of viewers, without the ever having seen the movie themselves. Admittedly not a desirable practice for a movie critic, but it is analogous to situations in which RL might be indispensable, an example being where the brain's neural firing patterns influence a complex system of joints and muscles. Movements can be evaluated in the complete absence of knowledge about the neural activity that produced them. This is likely one reason that RL is so important in the nervous system, a topic I take up below.

Another key feature of RL is that, unlike supervised learning (at least in its basic form), RL is selectional. Being selectional means that RL is like evolution by natural selection in requiring the generation of a variety of alternatives and then selecting among them by trying them out and evaluating their consequences. In supervised learning, by contrast, the labels of labeled training examples directly tell the learning system what the outputs should be: supervised learning (at least in its basic form) is instructional rather than selectional. It is worth pointing out here that RL's process of generating alternatives to be tried out does not have to be uninformed or random. It can be very sophisticated, the only requirement being that it is blind in the sense that the outcome of a trial should not be completely known beforehand; otherwise, RL's capacity for discovery is lost.

Then is not RL simply search, which of course has long been a centerpiece of AI? Generate and test is a hallowed principle of problem-solving, and simple hill-climbing is a prototype of a selectional process. Trial-and-error learning is indeed search (usually optimization search rather than search for a recognizable goal), but it is also learning. Unlike basic search, trial-and-error learning is associative, meaning that the good alternatives found by selection become associated with particular situations, or states, so that search becomes easier, perhaps eventually becoming unnecessary, with accumulating experience. Like learning described by the law of effect, RL is not just the process of finding actions that produce satisfaction, but also of connecting those actions to situations, or states. Thorndike used the phrase "learning by selecting and connecting." The law of effect describes 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 associations linking situations with the actions found — so far — to work best in those situations. Search and memory are essential components of RL algorithms.

This use of memory is also at the base of the closely related process called *memoization* (Popplestone 1967; Michie 1967). This process saves results of a calculation in memory so that results that have been calculated previously can be retrieved from memory instead of being calculated again. Michie was well aware of the connection between memoization and trial-and-error learning (Michie 1968). In fact, it is not misleading to think of trial-and-error learning as memoized search. Although the original memoization idea was to store the results of computations purely by rote in a lookup table, Popplestone ended his 1967 report by suggesting that an interpolation scheme could be used to generalize beyond the individual cases, thus anticipating the current use of function approximation methods, such as deep neural nets, in RL.

Despite being a selectional process akin to evolu-

tion, RL differs from evolutionary algorithms that mimic evolution's natural selection through updating a population of candidate solutions. Evolutionary algorithms do not focus on forming associative linkages between situations, or states, and actions. The population maintained by an evolutionary algorithm is a kind of memory, and one can evolve situation-action linkages, but evolutionary algorithms typically do not form associative linkages in the way that an RL algorithm does.

Sometimes RL is considered to be a version of unsupervised learning, which is about finding structure hidden in collections of unlabeled data. RL algorithms do not use labeled data as supervised learning algorithms do, but they do receive training information in the form of reward signals. Uncovering structure in an agent's experience can certainly be useful in RL, but by itself does not address the problem of improving behavior as evaluated by a reward signal. RL is therefore a third ML paradigm, alongside supervised learning and unsupervised learning, and perhaps alongside other paradigms as well.

For the reasons just outlined, we concluded from our study of early ML that law-of-effect-type learning, or trial-and-error learning, was in fact mostly neglected by AI researchers, and we concluded that Klopf's ideas were indeed worth pursuing. This early exploration also gave us an idea about why computational research with such a simple commonsense form of learning did not flourish in AI. One reason is that researchers apparently thought they were studying trial-and-error learning when they were actually studying error-correcting supervised learning. We saw this in the work of Farley and Clarke, mentioned earlier. In their 1954 paper, Farley and Clark describe a neural network that learned via RL, with the neuron-like units behaving very much like Klopf's hedonistic neurons. But near the end of their paper, Farley and Clark highlighted their interest in pattern classification and generalization, which can be studied without RL:

> It is to be hoped that, using a more complex modifier [that is, learning rule], this type of behavior can also be organized and controlled, leading to systems which effect classification and generalizations. (1954, 81)

Indeed, their second paper (Clark and Farley 1955) was entirely devoted to the pattern classification and generalization properties of their network. Later, but still early, artificial neural networks such as Rosenblatt's Perceptron (Rosenblatt 1958, 1962) and networks of Widrow and Hoff's ADALINEs (ADAptive LInear NEurons, and later ADAptive LINear Elements) (Widrow and Hoff 1960) all implemented error-correcting supervised learning to focus on pattern classification and generalization.

Confusing trial-and-error learning with error-correction learning is understandable because the word "error" of trial-and-error learning is not the right word: it should be *evaluation* or perhaps *test,* as I explained earlier, but history dictates otherwise. We can see this confusion in remarks by illustrious ML pioneers. For example, in describing their ADALINE supervised learning algorithm — now usually called the *least mean square* (LMS) algorithm, or sometimes the delta rule — Widrow and Hoff offered this analogy:

> The boss continually seeks a better worker by trial and error experimentation with the structure of the worker. Adaptation is a multidimensional *performance feedback* process. The "error" signal in the feedback control sense is the gradient of the mean square error with respect to the adjustment. ([1960] 1988)

Widrow and Hoff were clearly describing error correction but calling it trial and error. Similar confusion can still be found today. More than a decade after this 1960 remark, Widrow and colleagues did study genuine trial-and-error learning, which they called "learning with a critic" as opposed to "learning with a teacher," as supervised learning is sometimes called (Widrow, Gupta, and Maitra 1973). But this was an isolated foray into this more difficult type of learning. There were other early explorations of genuine trial-and-error learning in AI, but trial-and-error learning was largely viewed as a prototypical "weak method," as most research turned toward symbolic AI and away from learning.

Playing Checkers, TD Learning, and Dynamic Programming

A notable exception to the general neglect of trial-and-error learning was Arthur Samuel's checkers player. This program (in its several versions) was — and still is — recognized as a significant achievement in AI and ML. Trials were moves in simulated games of self-play, the results of which were used to adjust the program's rule for selecting moves. The goal was to improve the quality of play as measured by piece advantage, which is highly correlated with winning. Moves were selected by performing lookahead search from each current board position, with the board positions visited in each search evaluated by a "scoring polynomial," a function of features extracted from the board positions. Moves leading to the board positions with the largest scores were selected for play.

With its use of a scoring polynomial and lookahead search, Samuel's program moved beyond the simple law of effect, but it still learned by trial and error, that is, by RL. These enhancements to the simple law of effect are at the heart of some of the most important connections to emerge from the study of RL. In particular, Samuel's method for learning the scoring polynomial connects to modern RL's temporal difference (TD) algorithms. (Though much simpler, Samuel's inclusion of lookahead search is a precursor to the use of Monte Carlo tree search in modern systems using RL such as AlphaGo and

AlphaGo Zero [Silver, Huang, et al. 2016; Silver, Schrittwieser, et al. 2017].)

The idea of the scoring polynomial was to predict the reward (or penalty) expected to be received after each trial game. It was learned by working backward through the search tree from the scored terminal board positions, giving each position the score of the position that would result from the best move, assuming that the machine would always try to maximize the score, while its opponent would always try to minimize it. Samuel called this the "backed-up score" of the board position. When the minimax procedure reached the search tree's root — the current board position — it yielded the best move under the assumption that the opponent would be using the same evaluation criterion, shifted to its point of view:

> … we are attempting to make the score, calculated for the current board position, look like that calculated for the terminal board positions of the chain of moves which most probably occur during actual play. (Samuel 1959, 543)

By adding a predictor to trial-and-error learning, Samuel was addressing what Minsky called the "credit assignment problem" (Minsky 1961), which includes the problem of assigning credit (or blame) to actions within a sequence of actions when the reward (or penalty) for the whole sequence is not available until later in the sequence, or at its end as is the case in a game like checkers.

This backing-up process is a key feature of TD algorithms developed by Richard Sutton, the first PhD student I supervised (now a professor of computing science at the University of Alberta and a distinguished research scientist at DeepMind). TD algorithms are for learning to predict future values of some quantity, usually (though not necessarily) the amount of future reward (Sutton 1984, 1988). The simplest TD algorithms are unusual forms of supervised learning algorithms. Instead of the target outputs being labels of training examples, as in usual supervised learning, the targets are later observations of the quantity being predicted. But this aspect of the target is no different from what we see with conventional adaptive prediction algorithms. What makes TD algorithms unique is that in addition to later observations of the signal being predicted, their targets also include the predictor's own later predictions. This approach is like Samuel's method described in the quotation earlier that makes the score of a board position "look like" the score of a later position as given by the same scoring polynomial that is being updated.

TD algorithms are error correcting, where errors, called *TD errors*, are differences between predictions made at different times (hence the moniker "temporal difference"). The connection to Samuel's work was not fully appreciated at first, but it turned out that TD algorithms refined and improved upon Samuel's method for learning a scoring polynomial. I was surprised that Sutton had actually improved upon a central ingredient of Samuel's venerable checkers player. Before Sutton had finished his dissertation, he and I, along with Charles Anderson, my second graduate student (now Professor of Computer Science at Colorado State University), followed the earlier RL pole-balancing BOXES system of Michie and Chambers (1968) in experimenting with what is now called the *actor-critic architecture* (Barto, Sutton, and Anderson 1983). This architecture combined a trial-and-error learning component (the *actor)* with an adaptive critic implementing a TD algorithm to predict a delayed reward signal (which for this application was a penalty for the pole falling). The adaptive critic learned in much the same way that the scoring polynomial was learned in Samuel's checkers player, and changes in the critic's predictions provided immediate evaluations of the actions performed by the actor.

This connection between TD learning and Samuel's method for learning a scoring polynomial was a gratifying sign that we were not straying too far from hallowed ideas in AI. But the most interesting connection, and to me the most surprising at the time, is that both Samuel's method and TD algorithms are closely related to dynamic programming (DP), the term introduced in 1953 by Richard Bellman (Bellman 1953), an applied mathematician working in control theory and operations research. The idea of updating a prediction by moving it closer to a later prediction — the backing-up process — which is at the core of both TD and Samuel's method, is an instance of an operation that is basic to DP algorithms.

DP can be applied to different types of problems, but most relevant here is its application to sequential decision problems. These problems require finding optimal decisions for each stage in a sequence of stages when the best decision at each stage can depend on all the decisions that will be made at later stages of the sequence. The most common DP algorithm for problems like this is an iterative computation that proceeds backward from the end of the sequence, updating decisions based on predictions about decisions at later stages that have already been computed earlier in the backward iteration. This algorithm is often expressed in its recursive form. When the stages correspond to successive moves in a game like checkers, DP can find — at least in principle — winning moves.

Because this DP algorithm proceeds backward through time (assuming sequences unfold over time, with later stages occurring later in time than preceding stages), it would seem to be irrelevant for learning because an agent's experience unfolds forward in time. However, the same effect of this backward iteration can be achieved in the forward direction by means of backing-up operations, as in Samuel's method and TD algorithms. The same effect of the

backward-in-time DP algorithm can be produced by making multiple forward passes through sequences, backing up predictions at each step. Because DP has long been identified with this backward-in-time algorithm, its relevance to learning has been underappreciated. In fact, within AI, DP had long been largely dismissed as merely a recursive formulation of simple breadth-first search (Nilsson 1971). (Though outside of AI, Paul Werbos's [1977] work recognized connections between DP, prediction, and learning, leading to his development of a thread of RL parallel to what was developing in AI.) Missing from the view of DP as breadth-first search is that DP is actually memoized search. Results of each portion of the search are saved — memoized — to be accessed repeatedly as the search proceeds. This strategy is essential for the forward-going version of DP and underlies its relevance to learning.

The most surprising thing for me in all of this was that Samuel and Bellman, though both having made key contributions in the 1950s, apparently were unaware of each other's work. We have not found any reference to the other's work in any publication by either Samuel or Bellman. Maybe the connections were too obscure at the time in the absence of later developments, but a more plausible explanation is that there was just too little interaction between early AI and control engineering and operations research. Checkers was far removed from the kinds of problems in which Bellman was interested, and DP was not thought of as relevant to learning. Fortunately, this situation has changed by now.

It is fair to say that modern RL research has been a major contributor to bringing these fields closer together. RL is now regarded as a collection of algorithms for approximating solutions to stochastic optimal control problems. Most current RL theory has been developed using the mathematical framework of Markov decision processes (MDPs), one of the simplest stochastic optimal control formulations. This link from RL, with its roots in AI and psychology, to the highly developed field of stochastic optimal control is one of the most important outcomes of modern RL research.

The Power of Monte Carlo

Another surprise for me — and I think also for many others working with RL at the time — was the success of Gerald Tesauro's backgammon-playing program TD-Gammon (Tesauro 1992, 1994). TD-Gammon required little backgammon knowledge, yet learned to play extremely well, near the level of the world's strongest grandmasters. (Several versions of TD-Gammon differed in various ways and achieved different levels of success against human experts.) The learning algorithm in TD-Gammon was a combination of a TD algorithm and nonlinear function approximation using a multilayer neural network trained by the backpropagation algorithm (Rumelhart, Hinton, and Williams 1986). Like Samuel's checkers player, TD-Gammon learned over many games of self-play.

TD-Gammon was actually the source of two surprises. The first was simply its demonstration that an RL system was able to learn a complex skill rivaling that of human experts. (Samuel's checkers player learned to play better than Samuel could, but it did not learn to play at the level of expert human checkers players.) The second surprise was more subtle. Over the years, RL had received the reputation of being very slow. This reputation was well deserved since compared to supervised learning, an RL system does learn slowly. Learning from a scalar reward signal is more difficult than supervised learning, where gradient information is more directly accessible.

But Tesauro's results invited a different comparison: that between RL and conventional DP algorithms, which could — at least in principle — find an optimal backgammon-playing strategy. Applying DP to backgammon, however, immediately encounters what Bellman called the "curse of dimensionality": the size of a problem's state space grows exponentially with the number of the space's dimensions. In the case of backgammon, given the dimensions Tesauro chose to represent the game's states, there are approximately 10^{20} distinct states — a very large number! Since conventional DP algorithms require multiple exhaustive sweeps through the state space, a rough calculation based on the speed of the fastest computers of the day told us that it would take over 1,000 years to perform even a single sweep. Faster computers would help, but not enough to make conventional DP feasible for a problem with a state space this large. Compared to conventional DP, then, RL was not slow at all, merely taking upwards of a million games of self-play!

Of course, TD-Gammon only approximated an optimal playing strategy (and one focused on playing against itself), however it vividly demonstrated that RL can produce adequate approximations to optimal decision rules for problems with very large state spaces. One important component of TD-Gammon was its multilayer neural net trained by backpropagation to approximate its version of Samuel's scoring polynomial. The revelation for me, however, was that TD-Gammon avoided exhaustive sweeps of that very large state space by focusing its computational effort on the states visited in many simulated games, that is, on the states in sample trajectories through backgammon's state space. This is an example of a Monte Carlo method, a method that uses repeated random sampling to obtain results in problems where other approaches are difficult or impossible to apply. The power of Monte Carlo simulation has long been appreciated in other fields, such as physics and economic forecasting, but TD-Gammon first illustrated the utility of this approach for tackling challenging problems with RL.

To understand one of the advantages of Monte Carlo simulation compared to other methods, consider the problem of computing the expectation of the score that a given playing strategy will achieve over any complete backgammon game against a given opponent strategy. This prediction problem is not the entire optimization problem addressed by TD-Gammon, but it is a component of the full problem and simpler to analyze. To estimate this expectation via Monte Carlo simulation, one simply simulates many complete games between the given strategies by drawing (pseudo) random numbers to simulate the board state transitions that would be produced by rolling the dice in actual play. The desired estimate of the expected score is then simply the average of the scores achieved over the simulated games. (This is essentially an instance of the algorithm TD(1) [Sutton and Barto 1998, 2018].)

Alternatively, it is possible to compute the exact expectation by finding one component of the solution to a system of n simultaneous linear equations, where n is the number of game states. The component of interest corresponds to the initial game state, which is the state from which all of the simulations begin in the Monte Carlo method. To construct these equations, it is necessary to know all the $n \times n$ state-transition probabilities, which requires complete knowledge of the opponent's strategy, as well as the rules of the game and the reward associated with each state. One way to solve this system is an iterative method that conducts multiple exhaustive sweeps over the n states, updating expected scores for games that would begin in, or include, each state. Even though we are interested in the prediction for only one state, the initial state of the game, the predictions for all the other states are updated on each sweep because an exact solution requires consideration of all possible games and their probabilities of occurring. This iterative method is a simplified version of a conventional DP algorithm that does not involve the optimization steps. The memoizing nature of DP makes it an efficient way to perform this computation. Another method is the standard Gaussian elimination method for inverting the $n \times n$ matrix corresponding to this system of equations.

The left panel of figure 1 shows plots of the work (number of multiplications) versus the number of states, n, required by the iterative, Monte Carlo, and Gauss methods for reducing the initial prediction error by a factor of $\xi = .01$ (with a 95 percent confidence level, and all sample games assumed to have the same expected number of moves). First note that the work for the Monte Carlo method is independent of n. In contrast, the work for the iterative and Gaussian methods increases rapidly (though polynomially) with increasing n. The advantage of the Monte Carlo method over the iterative and Gauss methods grows rapidly beyond a rather small number of states. This advantage depends on the desired error reduction factor, ξ. Plotting ξ versus work, as shown in the right panel of figure 1 for the iterative and Monte Carlo methods, shows that the Monte Carlo method does not require a lot of work unless one demands high accuracy (very small ξ), a property clearly not enjoyed by the iterative (or the Gauss) method.

The prediction problem just analyzed is not the optimization problem addressed by TD-Gammon, and it is not the optimization problem that can be solved by the full version of DP: it is the problem of evaluating a given playing strategy, not the problem of finding, or approximating, the best strategy. Nevertheless, TD-Gammon's algorithm shares key features with the Monte Carlo prediction method and enjoys the same kinds of advantages over optimization by conventional sweep-based DP.

In a game like backgammon, as well as in many other problems with large state spaces, many states have a very low probability of being visited in any trajectory that might actually occur. Such states are essentially irrelevant, so it is not important to devote computational effort to finding good actions to take from them. Monte Carlo stochastic estimation automatically allocates computational effort to states according to their probabilities of occurring in actual trajectories. Computation is rarely devoted to finding good actions for states that would occur only rarely. The favorable scaling properties of Monte Carlo methods suggest why RL, though perceived to be slow, can actually be advantageous for problems with large numbers of states.

Two Kinds of Models

Models of the world with which an RL agent interacts, that is, models of the agent's environment, can play a variety of roles in acting and learning. Models can support planning, which enables agents to evaluate possible courses of action without actually performing them in their real environments. Conventional DP algorithms need models to compute all the expected values needed to find optimal decision policies. Simulation-based Monte Carlo methods need environment models to run the many simulations they require. For much of my experience with various kinds of RL algorithms, I did not distinguish between different kinds of models: models could be used in different ways, but models were models, simply structures that could act as surrogates for an agent's actual environment.

My casual view lasted until the late 1990s when my graduate student Robert Crites applied RL to the problem of elevator dispatching. The elevator dispatching problem is the problem of deciding how elevators should respond to passenger requests so that, for example, the amount of time any passenger is expected to wait until they get to their destination is minimized. Crites studied the application of RL to the four-elevator, ten-floor system shown in figure 2

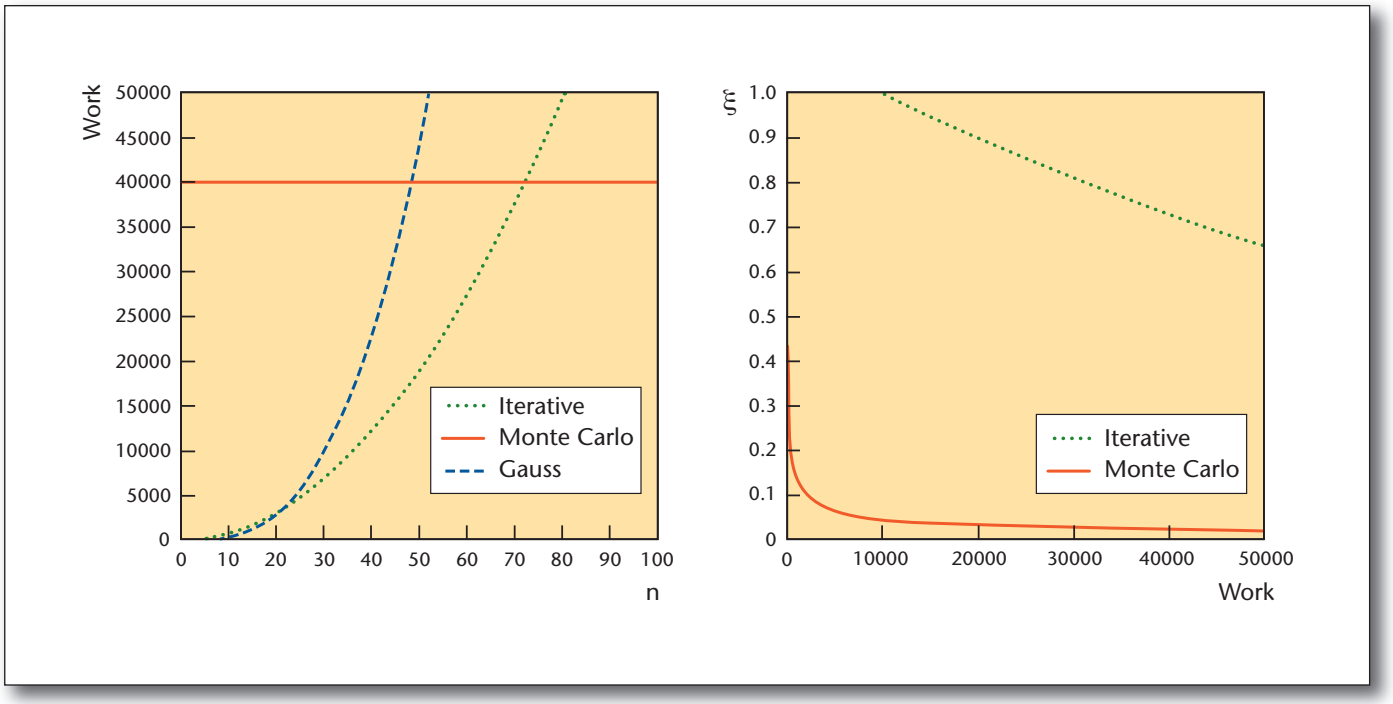

Figure 1. Comparison of Three Prediction Methods.

Left: Work versus number of states, *n,* for reducing the initial error by a factor of ξ = .01 *Right:* Error reduction versus work for *n* = 100. From Barto and Duff (1994).

(Crites 1996; Sutton and Barto 1998). Along the right-hand side are pickup request buttons and an indication of how long since each button was pressed. Each car has a position, direction, and speed, plus a set of buttons to indicate where passengers want to get off. Each car has a small set of primitive actions: if it is stopped at a floor, it must either move up or move down; if it is in motion between floors, it must either stop at the next floor or continue past the next floor. Roughly quantizing the continuous variables, Crites estimated that this elevator system has over 10^{22} states, making conventional sweep-based DP completely infeasible, but making the problem a good candidate for RL.

Inspired by TD-Gammon, Crites obtained a program that simulated the elevator system. (A research group studying more conventional methods was kind enough to give Crites its simulation code.) Periods of elevator operation were analogous to the simulated self-play games from which TD-Gammon learned. Each of several RL controllers were trained on 60,000 hours of simulated elevator time, which took four days on a workstation of the day. Crites's results showed that the dispatching policy learned by RL surpassed in simulation the best of the heuristic elevator control algorithms of which we were aware. (We never got so far as to work with an elevator company toward actually deploying a dispatching policy learned by RL in a real elevator installation because immediately upon receiving the PhD, Crites landed a job doing something completely different.)

My revelation came when I tried to write down the state-transition and reward probabilities that were the stock-in-trade of the MDP framework that had become standard for RL research. In the first place, the large number of states made it impossible to list all of these probabilities, but I was hoping to make the job simpler by considering aggregations of states that shared transition probabilities and/or rewards. But this was formidable too because the state transitions and rewards embodied in the simulation were the result of the interaction of many parts of the model. While not exactly an agent-based simulation in the modern sense (for example, Waldrop [2018]), the simulation shared with agent-based simulations the property that the state-transition probabilities emerged from many interacting components: passengers arriving, passengers making pickup and dropoff requests at various floors, elevators stopped or moving up or down at various locations, and the timing of all these events. The simulation generated behavior of the elevator system according to state-transition probabilities, but these probabilities were not explicit, and, in fact, were not needed at all for applying RL.

Now it is common in RL to distinguish between what are sometimes called *distribution models* and *sample models*. Distribution models consist of the probability distributions, either in tabular form or specified by sets of equations, that are needed by con-

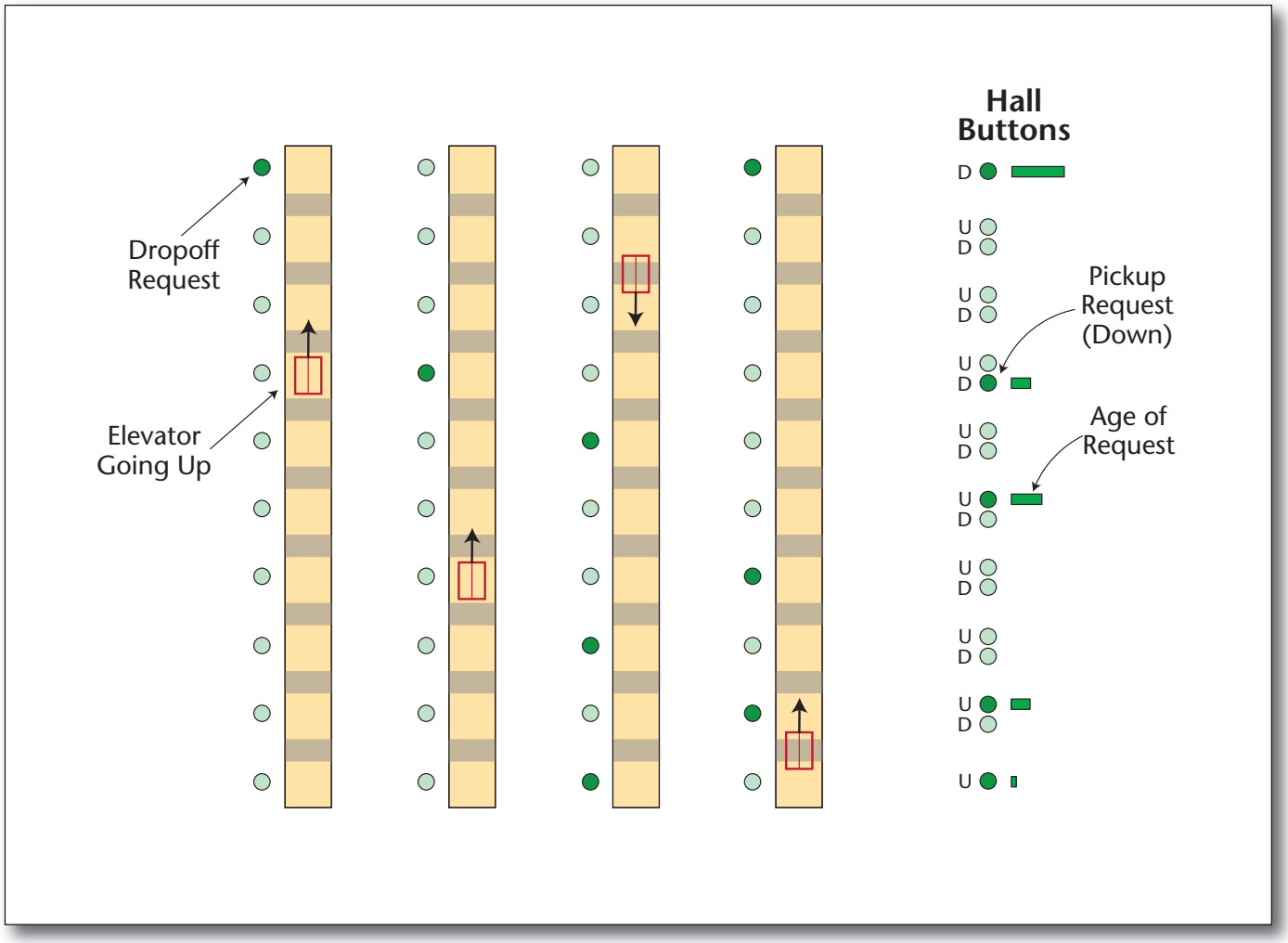

Figure 2. Four Elevators in a Ten-Story Building.

From Sutton and Barto (1998).

ventional DP algorithms. Sample models, on the other hand, produce state transitions and rewards that are sampled from these distributions. Clearly, if one has a distribution model, one can sample as needed from the distributions. But as the elevator task made clear, it is possible to generate samples by a simulation program that does not contain explicit representations of the underlying probability distributions. While theoretically possible to make these distributions explicit, it is not necessary. For this reason, a sample model is often much easier to create than the corresponding distribution model.

I realized, then, that another advantage of RL over optimization methods that depend on distribution models, such as conventional DP, is that RL can approximate optimal solutions through Monte Carlo optimization using only sample models. This advantage would not have been news to those in other disciplines who already understood the advantages of simulation-based optimization, but for me it was an important realization.

Dopamine

One of the most exciting connections between RL and another discipline is the result of what neuroscientists are learning about the brain's reward system. There is mounting evidence from neuroscience that the nervous systems of humans and many other animals implement algorithms that correspond in striking ways to RL algorithms. The most remarkable point of contact involves dopamine, a chemical fundamentally involved in reward processing in the brains of mammals.

Experiments conducted in the late 1980s and the 1990s in the laboratory of neuroscientist Wolfram Schultz (reviewed in Schultz [1998]) showed that neurons that produce dopamine as a neurotransmitter respond to rewarding events with substantial bursts of activity only if the animal does not expect those events. This finding suggests that dopamine-producing neurons are signaling reward prediction errors instead of reward itself. Further, these experi-

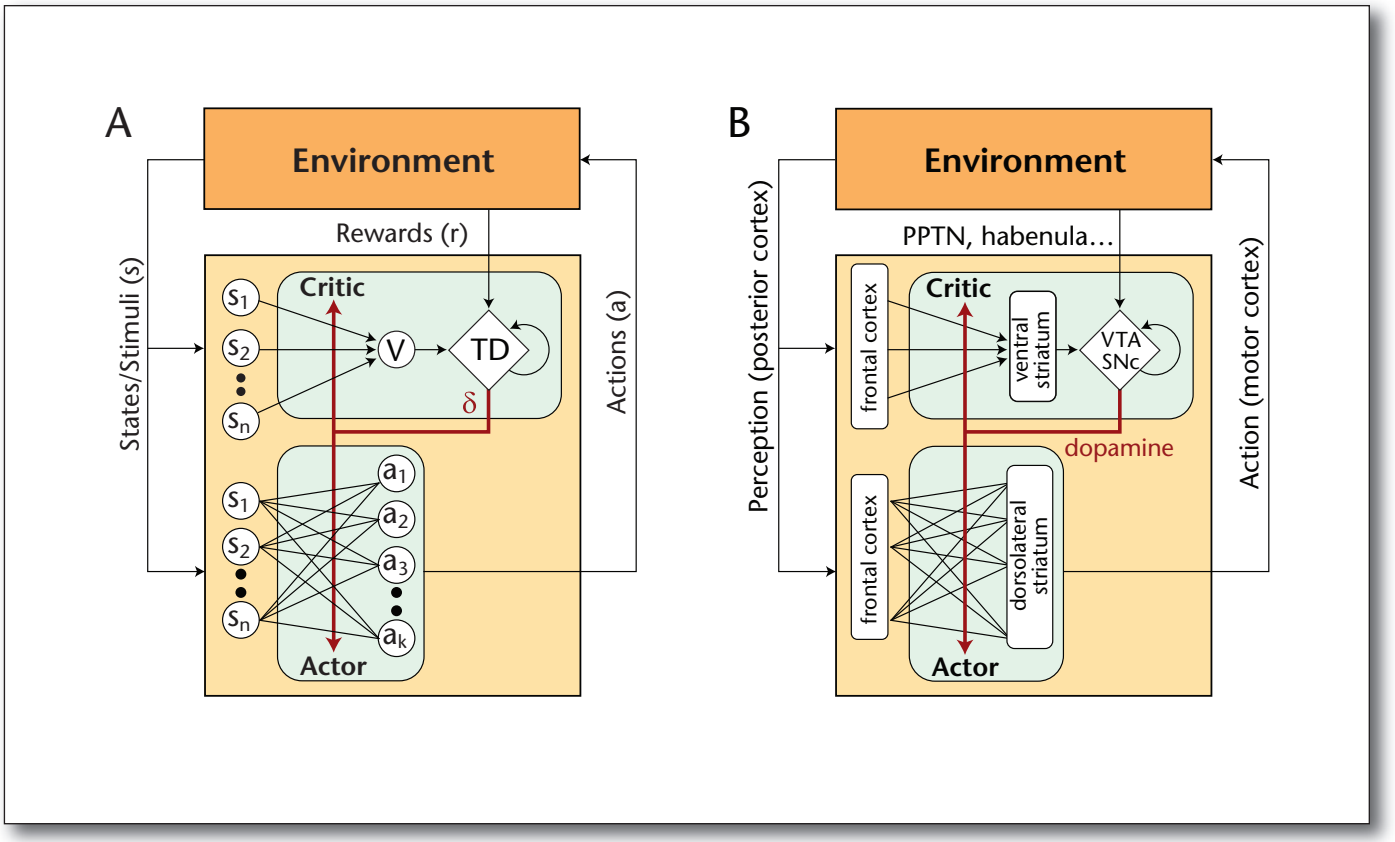

Figure 3: Actor-Critic as an Artificial Neural Network and a Hypothetical Neural Implementation.

Adapted from Takahashi, Schoenbaum, and Niv (2008).

ments showed that as an animal learns to predict a rewarding event on the basis of preceding sensory cues, the bursting activity of dopamine-producing neurons shifts to earlier predictive cues while decreasing to later predictive cues.

Researchers familiar with RL quickly recognized that these results are strikingly similar to how the TD error behaves as an RL agent learns to predict reward (for example, Barto [1995]; Schultz, Daylan, and Montague [1997]). It is not an exaggeration to say that the results of the experiments of Schultz and colleagues, together with their correspondence to RL algorithms, have revolutionized the neuroscience of reward processing in the brain. It is now almost universally accepted that bursts of dopamine neuron activity convey reward prediction errors to brain structures where learning and decision-making take place, and evidence supports the idea that the prediction errors might be TD errors.

RL theory provides a model for understanding the functional significance of reward prediction errors. In addition to driving the learning of reward predictions, reward prediction errors are ideal signals implementing trial-and-error learning. Actions followed by greater-than-expected reward (a positive reward prediction error) are selected for; actions followed by less-than-expected reward (a negative reward prediction error) are selected against. This observation suggests that the brain might implement something like an actor-critic algorithm in which TD errors are both error signals to train the critic's predictions and signals for encouraging or discouraging the actor's choice of actions.

Figure 3 illustrates a hypothesis about how the brain might implement an actor-critic algorithm. Panel (a) shows the actor-critic algorithm as an artificial neural network. The actor adjusts a policy based on the TD error δ it receives from the critic; the critic adjusts reward predictions using the same δ. The critic produces a TD error from the reward signal, *r,* and its current reward predictions. Panel (b) shows a hypothetical neural implementation of an actor-critic algorithm. The actor and the critic are respectively placed in particular parts of the brain. The TD error is transmitted by dopamine-producing neurons to modulate changes in synaptic weights of input from cortical areas.

While these developments do not directly support Klopf's hypothesis that individual neurons implement a kind of law of effect, a recent study by Athalye et al. (2018), entitled "Evidence for a Neural Law of Effect," adds to the plausibility of Klopf's idea.

These authors devised a way to monitor the activities of groups of neurons in the brains of mice and directly trigger dopamine neuron activity when the monitored neurons produced desired target activity patterns. They found that the target activity patterns — those reinforced by dopamine — occur with increasing frequency over time, and that bursts of dopamine neuron activity shape the activity patterns to more closely resemble the target patterns. There are other ways to explain these results besides locating the law of effect within individual neurons, but Klopf's hypothesis is a leading possibility.

A remarkable aspect of these developments is that the RL algorithms and theory that connect so well with properties of the dopamine system were developed from a computational perspective in total absence of any knowledge about how dopamine-producing neurons behave and the role dopamine plays in learning. TD learning and its connections to optimal control and DP were developed years before the key neuroscience experiments were conducted. This unplanned correspondence suggests that the TD error/dopamine parallel and other aspects of RL algorithms capture something significant about brain reward processing. The brain's reward system is undoubtedly much more complicated than these algorithms, and the story is still unfolding as more is being learned about the brain's reward system, but we can surely expect that continued interaction between neuroscience and RL will lead to fruitful advances on both sides.

Challenges

Many challenges have to be faced as RL moves out into the real world. One is the challenge of extending the capabilities of RL systems so that they can help address pressing real-world problems, which will mean also making RL methods more robust and easier to apply. Another challenge is to develop ways to ensure that RL applications make positive contributions to our lives that outweigh any negative consequences.

Many researchers around the world are working to extend the capabilities of basic RL systems. Some of the most dramatic results have been achieved by combining RL with other methods, such as deep neural networks and Monte Carlo tree search, as in DeepMind's impressive Go-playing programs. Combining RL with other methods, such as Bayesian methods, symbolic methods as seen in developments of relational RL, and evolutionary methods, also extends the capabilities of basic RL systems in important directions. Other extensions are perhaps more correctly viewed as enhancements of the RL framework itself, as seen in the development of hierarchical RL, RL for partially observable MDPs, and ways of handling continuous state and action spaces. Wiering and van Otterlo (2012) provide good introductions to research in many of these directions.

Other efforts have been devoted to making RL methods more robust and easier to apply to real problems. Many design decisions are involved in applying RL, including selecting state and action representations and setting hyperparameters that control such things as learning rate, exploration, and eligibility trace characteristics. Some approaches eliminate hyperparameters altogether, or adapt them during learning. Of particular interest to me is the problem of designing the reward function for an application. This is the function that assigns a numerical reward amount to states, actions, state-action pairs, and perhaps other aspects of the RL system: it defines the goal of the learning agent. The success of an RL application strongly depends on how well the reward function frames the goal of the application's designer and how well it assesses progress in reaching that goal. The reward function is another hyperparameter that has to be set at the start.

A critical challenge in designing a reward function is that any method, like RL, that is based on optimization can produce unexpected results. This possibility has long been recognized in literature and engineering. In the ancient myth of King Midas, for example, joy with his golden touch turned to fear when his food and even his daughter turned into gold. Norbert Wiener, the founder of cybernetics, warned of this problem more than half a century ago by relating the supernatural story of "The Monkey's Paw": "... it grants what you ask for, not what you should have asked for or what you intend" (Wiener 1964, 59). The problem is featured as "perverse instantiation" in Bostrom's (2014) broadside about the dangers of AI. RL agents can discover unexpected ways to make their environments deliver reward, some of which might be undesirable or even dangerous.

This perverse literalness is not so much of a problem if RL takes place in simulated environments, as is the case for the most notable applications of RL to date. But if RL operates online while an agent is interacting with a real physical environment, it is critical to make sure that what is learned conforms to the intentions of the application's designer and that the agent does no harm to itself or to its environment, including any people in it, both during and after learning. It is critical too when what is learned in simulation is then, after learning, deployed in the real world. Unless RL is restricted to always operate in benign environments, like game playing where one can tolerate the worst that can happen, ensuring the safety of RL applications is a critical challenge that needs careful attention.

We can take some solace from the fact that optimization has been used for hundreds of years by engineers, architects, and others whose designs have positively impacted the world. Approaches have been developed to mitigate and manage optimization's risks, and we owe much that is good in our

environment to optimization methods. Mitigating and managing risk for an RL system while it is learning in the real world is not completely novel or unique to RL. Control engineers have had to confront similar problems from the beginning of using automatic control in situations where a controller's behavior can have unacceptable, possibly catastrophic, consequences. As RL moves out into the real world, developers have an obligation to adapt and extend best practices that have guided applications of more established technologies that have improved the quality, efficiency, and cost-effectiveness of processes upon which we have come to rely.

Conclusion

I delivered brief opening remarks at the First Multidisciplinary Conference on Reinforcement Learning and Decision Making held at Princeton University in 2013. After recounting my early anxiety that I was doing nothing but reinventing the wheel, I urged the mostly young audience to not let this sort of anxiety inhibit their research. But, I went on to say: if you do reinvent the wheel, please call it a wheel, or perhaps an improved wheel, instead of giving it a new name unconnected from the fabric of history. Effort to do this by me and others in studying RL — which contains a lot of wheel-like parts — has resulted in the multidisciplinary fabric that has sustained my interest in the subject.

My intention in this article has been to convey a sense of this multidisciplinary ground that RL covers by describing some of the connections, surprises, and challenges that have impressed me over the years during which my students and I focused on RL. Exploration of Klopf's idea of hedonistic neurons led to excursions through some of the early history of AI, to psychology's theories of learning, to appreciation of DP and the power of Monte Carlo methods. Then the striking parallels between TD algorithms and the brain's dopamine system revealed strong connections between RL algorithms and reward processing in the brain. It is fair to say that the scientific merit of Klopf's hypothesis of the hedonistic neuron — the exploration of which started me out upon this journey — has been amply demonstrated, and as neuroscience reveals more about how reward processing works in the brain, we might see more detailed support for the idea that individual neurons implement the law of effect. Finally, witnessing the potency of deep neural networks coupled with RL and Monte Carlo tree search in DeepMind's Go-playing programs opened a vista onto possibilities for RL to help improve the quality, fairness, and sustainability of life on our planet, provided its risks can be successfully managed.

Acknowledgments

The author thanks his many talented and creative students who made the journey described in this article possible, and the Air Force Office of Scientific Research and the National Science Foundation for their financial support.

References

Athalye, V. R.; Santos, E. J.; Carmena, J. M.; and Costa, R. M. 2018. Evidence for a Neural Law of Effect. *Science* 359(6379): 1024–29. doi.org/10.1126/science.aao6058.

Barto, A. G. 1995. Adaptive Critics and the Basal Ganglia. In *Models of Information Processing in the Basal Ganglia,* edited by J. C. Houk, J. L. Davis, and D. G. Beiser, 215–32. Cambridge, MA: The MIT Press.

Barto, A. G., and Duff, M. 1994. Monte Carlo Matrix Inversion and Reinforcement Learning. In *Advances in Neural Information Processing Systems: Proceedings of the 1993 Conference,* edited by J. D. Cohen, G. Tesauro, and J. Alspector, 687–94. San Francisco, CA: Morgan Kaufmann.

Barto, A. G.; Sutton, R. S.; and Anderson, C. W. 1983. Neuronlike Elements That Can Solve Difficult Learning Control Problems. *IEEE Transactions on Systems, Man, and Cybernetics* 13(5): 835–46. Reprinted in *Neurocomputing: Foundations of Research,* 1988, edited by J. A. Anderson and E. Rosenfeld, 535–49. Cambridge, MA: The MIT Press.

Bellman, R. 1953. *An Introduction to the Theory of Dynamic Programming*. RAND monograph R-245. Santa Monica, CA: The Rand Corporation.

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

Clark, W. A., and Farley, B. G. 1955. Generalization of Pattern Recognition in a Self-Organizing System. In *Proceedings of the 1955 Western Joint Computer Conference,* 86–91. doi.org/10.1145/1455292.1455309.

Crites, R. H. 1996. Large-Scale Dynamic Optimization Using Teams of Reinforcement Learning Agents. PhD dissertation, Department of Computer and Information Science, University of Massachusetts, Amherst, MA.

Farley, B. G., and Clark, W. A. 1954. Simulation of Self-Organizing Systems by Digital Computer. *IRE Transactions on Information Theory* 4(4): 76–84. doi.org/10.1109/TIT.1954.1057468.

Klopf, A. H. 1972. *Brain Function and Adaptive Systems — A Heterostatic Theory*. Technical Report AFCRL-72-0164. Bedford, MA: Air Force Cambridge Research Laboratories. (A summary appears in *Proceedings of the International Conference on Systems, Man, and Cybernetics*. 1974. New York: Institute of Electrical and Eloectronics Engineers.)

Klopf, A. H. 1982. *The Hedonistic Neuron: A Theory of Memory, Learning, and Intelligence*. Washington, DC: Hemisphere.

Michie, D. 1967. *Memo Functions: A Language Feature with "Rote-Learning" Properties*. Research Memorandum MIP-R-29. Edinburgh, UK: University of Edinburgh, Department of Machine Intelligence and Perception.

Michie, D. 1968. "Memo" Functions and Machine Learning. *Nature* 218(5136): 19–22. doi.org/10.1038/218019a0.

Michie, D., and Chambers, R. A. 1968. BOXES: An Experiment in Adaptive Control. In *Machine Intelligence 2*, edited by E. Dale and D. Michie, 137–52. Edinburgh, UK: Oliver and Boyd.

Minsky, M. L. 1954. Theory of Neural-Analog Reinforcement Systems and Its Application to the Brain-Model Prob-

lem. PhD Dissertation, Princeton University, Princeton, NJ.

Minsky, M. L. 1961 Steps Toward Artificial Intelligence. *Proceedings of the Institute of Radio Engineers* 49(1): 8–30. Reprinted in *Computers and Thought,* 1963, edited by E. A. Feigenbaum and J. Feldman, 406–50. New York: McGraw-Hill.

Narendra, K. S., and Thathachar, M. A. L. 1989. *Learning Automata: An Introduction*. Englewood Cliffs, NJ: Prentice Hall.

Nilsson, N. J. 1971. *Problem-Solving Methods in Artificial Intelligence*. New York: McGraw-Hill.

Popplestone, R. J. 1967. *Memo Functions and the Pop-2 Language*. Research Memorandum MIP-R-30. Edinburgh, UK: University of Edinburgh, Department of Machine Intelligence and Perception.

Rosenblatt, F. 1958 The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. *Psychological Review* 65(6): 386–408. doi.org/10.1037/h0042519.

Rosenblatt, F. 1962. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Washington, DC: Spartan Books.

Rumelhart, D. E.; Hinton, G. E.; Williams, R. J. 1986. Learning Internal Representations by Error Propagation. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition,* edited by D. E. Rumelhart and J. L. McClelland, Vol. I, Foundations. Cambridge, MA: The MIT Press.

Samuel, A. L. 1959. Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal on Research and Development* 3(3): 211–29. Reprinted in *Computers and Thought,* 1963, edited by E. A. Feigenbaum and J. Feldman, 71–105. New York: McGraw-Hill.

Samuel, A. L. 1959. Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal on Research and Development* 3(3), 210–229.

Schultz, W.; Dayan, P.; and Montague, P. R. 1997. A Neural Substrate of Prediction and Reward. *Science* 275(5306): 1593–98. doi.org/10.1126/science.275.5306.1593.

Schultz, W. 1998. Predictive Reward Signal of Dopamine Neurons. *Journal of Neurophysiology* 80(1): 1–27. doi.org/10.1152/jn.1998.80.1.1.

Shannon, C. E. 1951. Presentation of a Maze-Solving Machine. In *Cybernetics. Transactions of the Eighth Conference,* edited by H. V. Forester, 173–180. New York: Josiah Macy Jr. Foundation.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the Game of Go with Deep Neural Networks and Tree Search. *Nature* 529(7587): 484–89. doi.org/10.1038/nature16961.

Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; et al. 2017. Mastering the Game of Go without Human Knowledge. *Nature* 550(7676): 354–59. doi.org/10.1038/nature24270.

Sutton, R. S. 1984. Temporal Credit Assignment in Reinforcement Learning. PhD dissertation, Department of Computer and Information Science, University of Massachusetts, Amherst, MA.

Sutton, R. S. 1988. Learning to Predict by the Method of Temporal Differences. *Machine Learning* 3(1): 9–44. doi.org/0.1007/BF00115009.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement Learning: An Introduction*. Cambridge, MA: The MIT Press.

Sutton, R. S., and Barto, A. G. 2018. *Reinforcement Learning: An Introduction,* 2nd ed. Cambridge, MA: The MIT Press.

Takahashi, Y.; Schoenbaum, G.; and Niv, Y. 2008. Silencing the Critics: Understanding the Effects of Cocaine Sensitization on Dorsolateral and Ventral Striatum in the Context of an Actor/Critic Model. *Frontiers in Neuroscience* 2(1): 86–99. doi.org/10.3389/neuro.01.014.2008.

Tesauro, G. J. 1992. Practical Issues in Temporal Difference Learning. *Machine Learning* 8(3–4): 257–77. doi.org/10.1007/BF00992697.

Tesauro, G. J. 1994. TD–Gammon, a Self-Teaching Backgammon Program, Achieves Master-Level Play. *Neural Computation* 6(2): 215–19. doi.org/10.1162/neco.1994.6.2.215.

Thorndike, E. L. 1911. *Animal Intelligence*. Darien, CT: Hafner Publishing.

Tsetlin, M. L. 1973. *Automaton Theory and Modeling of Biological Systems*. New York: Academic Press.

Turing, A. M. 1948. Intelligent Machinery. In *The Essential Turing,* 2004, edited by B. J. Copeland, 410–32. Oxford: Oxford University Press. Citations refer to the reprinted text.

Waldrop, M. M. 2018. Free Agents: Monumentally Complex Models Are Gaming Out Disaster Scenarios with Millions of Simulated People. *Science* 360(6385): 144–47. doi.org/10.1126/science.360.6385.144.

Werbos, P. J. 1977. Advanced Forecasting Methods for Global Crisis Warning and Models of Intelligence. *General Systems Yearbook* 22(1): 25–38.

Widrow, B., and Hoff, M. E. 1960. Adaptive Switching Circuits. In *1960 WESCON Convention Record Part IV*, 96–104. New York: Institute of Radio Engineers. Reprinted in *Neurocomputing: Foundations of Research,* 1988, edited by J. A. Anderson and E. Rosenfeld, 126–34. Cambridge, MA: The MIT Press.

Widrow, B.; Gupta, N. K.; and Maitra, S. 1973. Punish / Reward: Learning with a Critic in Adaptive Threshold Systems. *IEEE Transactions on Systems, Man, and Cybernetics* 3(5): 455–65. doi.org/10.1109/TSMC.1973.4309272.

Wiener, N. 1964. *God and Golem, Inc*. Cambridge, MA: The MIT Press.

Wiering, M., and Otterlo, M. V, eds. 2012. *Reinforcement Learning: State-of-the-Art*. Adaptation, Learning, and Optimization 12. Berlin: Springer-Verlag. doi.org/10.1007/978-3-642-27645-3.

Andrew Barto is a professor emeritus in the College of Information and Computer Sciences at the University of Massachusetts Amherst. He received a BS in mathematics with distinction in 1970 and a PhD in computer science in 1975, both from the University of Michigan. He is recipient of the 2004 IEEE Neural Network Society Pioneer Award and the IJCAI-17 Award for Research Excellence. He is coauthor of the book *Reinforcement Learning: An Introduction,* MIT Press 1998, the second edition of which was recently published.