Some Recent Applications of Reinforcement Learning

A. G. Barto, P. S. Thomas, and R. S. Sutton

Abstract— **Five relatively recent applications of reinforcement learning methods are described. These examples were chosen to illustrate a diversity of application types, the engineering needed to build applications, and most importantly, the impressive results that these methods are able to achieve. This paper is based on a case-study chapter of the forthcoming second edition of Sutton and Barto's 1998 book "Reinforcement Learning: An Introduction" [7].**

I. INTRODUCTION

Machine learning has come into its own as a key technology for a wide range of applications. Reinforcement learning is the branch of machine learning that allows systems to learn from the consequences of their own decisions instead of from the decisions of human experts. This makes reinforcement learning useful in problems where it is difficult, expensive, or impossible to obtain reliable expert training information. In this paper we describe five relatively recent applications of reinforcement learning. These applications were chosen to illustrate the diversity of problems to which reinforcement learning is being applied, a range of different reinforcement learning methods, including some that make use of deep neural networks, and the engineering needed to make them work. Most importantly, these applications illustrate the impressive results that are achievable with reinforcement learning, foreshadowing the promise of similarly-impressive results over the future in many other challenging problems.

II. WATSON'S DAILY-DOUBLE WAGERING

IBM WATSON[1] is the system developed by a team of IBM researchers to play the popular TV quiz show *Jeopardy!*.[2] It gained fame in 2011 by winning first prize in an exhibition match against human champions. Although the main technical achievement demonstrated by WATSON was its ability to quickly and accurately answer natural language questions over broad areas of general knowledge, its winning *Jeopardy!* performance also relied on sophisticated decision-making strategies for critical parts of the game. Tesauro, Gondek, Lechner, Fan, and Prager [9, 10] adapted Tesauro's TD–Gammon system [11, 12] to create the strategy used by WATSON in "Daily-Double" (DD) wagering in its celebrated winning performance against human champions. These authors report that the effectiveness of this wagering strategy went well beyond what human players are able to do in live game play, and that it, along with other advanced strategies, was an important contributor to WATSON's impressive winning performance.

Jeopardy! is played by three contestants who face a board showing 30 squares, each of which hides a clue and has a dollar value. One or two squares are special DD squares. A contestant who selects one of these gets an exclusive opportunity to respond to the square's clue and has to decide—before the clue is revealed—on how much to wager, or bet. If the contestant responds correctly to the DD clue, their score increases by the bet amount; otherwise it decreases by the bet amount. Winning or losing often depends on a contestant's DD wagering strategy.

Whenever WATSON selected a DD square, it chose its bet by comparing action values, $Q(s, \text{bet})$, that estimated the probability of a win from the current game state, s, for each round-dollar legal bet. Except for some risk-abatement measures, WATSON selected the bet with the maximum action value. Action values were computed whenever a betting decision was needed by combining a state value function, $V(\cdot, \theta)$, defined by parameters θ, estimating the probability of a win for WATSON from any game state, with an "in-category DD confidence" estimate of the likelihood that WATSON would respond correctly to the as-yet unrevealed DD clue.

$V(\cdot, \theta)$ was learned by the same reinforcement learning approach used by TD–Gammon: a combination of nonlinear TD(λ) using a multilayer neural network with weights θ trained by backpropagating TD errors during many simulated games. States were represented to the network by features that included the current scores of the three players, how many DDs remained, the total dollar value of the remaining clues, and other information related to the amount of play left in the game. Unlike TD–Gammon, which learned by self-play, WATSON's V was learned over millions of simulated games against models of human players based on statistics extracted from an extensive archive of game information that included information for nearly 300,000 clues. The in-category DD confidence estimates were based on the number of right and wrong responses that WATSON gave in previously-played clues in the current category over many thousands of historical categories.

Although its ability to quickly and accurately answer natural language questions stands out as WATSON's major achievement, all of its sophisticated decision strategies contributed to its impressive defeat of human champions. According to Tesauro et al. [9]:

> ... it is plainly evident that our strategy algorithms achieve a level of quantitative precision and real-

A.G. Barto and P.S. Thomas are with the College of Information and Computer Sciences, University of Massachusetts Amherst barto@cs.umass.edu, PThomasCS@gmail.com

R.S. Sutton is with the Department of Computing Science, University of Alberta, Edmonton, Alberta rsutton@ualberta.ca

[1]Registered trademark of IBM Corp.

[2]Registered trademark of Jeopardy Productions Inc.

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

III. OPTIMIZING MEMORY CONTROL

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

İpek, Mutlu, Martínez, and Caruana [1] (also Martínez and İpek [4]) designed a reinforcement learning DRAM controller and demonstrated that it can significantly improve the speed of program execution over what was possible with conventional controllers at the time of their research. They were motivated by limitations of existing state-of-the-art controllers that used policies that did not take advantage of past scheduling experience and did not account for long-term consequences of scheduling decisions. İpek et al.'s project was carried out by means of simulation, but they designed the controller at the detailed level of the hardware needed to implement it—including the learning algorithm—directly on a processor chip.

A DRAM controller maintains a *memory transaction queue* that stores memory-access requests from the processors sharing the memory system. The controller has to process requests by issuing commands to the memory system while adhering to a large number of timing constraints. To transfer a row of bits into or out of a row of a DRAM array, the row has to be "opened" by issuing an *activate* command, which moves the row's contents into the array's row buffer. With a row open, the controller can issue *read* and *write* commands to the cell array. Each read command transfers the contents of the row buffer to the external data bus, and each write command transfers a word in the external data bus to the row buffer. Before a different row can be opened, a *precharge* command must be issued which transfers the (possibly updated) data in the row buffer back into the addressed row of the cell array. After this, another activate command can open a new row to be accessed. Read and write commands to the currently-open row can be carried out more quickly than accessing a different row, which would involve additional precharge and activate commeands.

The simplest scheduling strategy handles access requests in the order in which they arrive by issuing all the commands required by the request before beginning to service the next one. But if the system is not ready for one of these commands, or executing a command would result in resources being underutilized (e.g., due to timing constraints arising from servicing that one command), it makes sense to begin servicing a newer request before finishing the older one. Policies can gain efficiency by reordering requests, for example, by giving priority to read requests over write requests, or by giving priority to read/write commands to already open rows.

İpek et al. modeled the DRAM access process as a Markov decision process (MDP) whose states are the contents of the transaction queue and whose actions are commands to the DRAM system: *precharge*, *activate*, *read*, *write*, and *NoOp*. The reward signal is 1 whenever the action is *read* or *write*, and otherwise it is 0. The reinforcement learning algorithm Sarsa was used to learn an action-value function via linear function approximation implemented by tile, or CMAC, coding. A relatively long list of potential state features was generated and then pared down to a handful using simulations guided by stepwise feature selection. States were finally represented by six integer-valued features: the number of read requests in the transaction queue, the number of write requests in the transaction queue, the number of write requests in the transaction queue waiting for their row to be opened, and the number of read requests in the transaction queue waiting for their row to be opened that are the oldest issued by their requesting processors. (The other features depended on how the DRAM interacts with cache memory, details we omit here.) The integrity of the DRAM system was assured by not allowing actions that would violate timing or resource constraints.

İpek et al. evaluated their learning controller in simulation by comparing it with several other controllers, including a First-Ready, First-Come-First-Serve (FR-FCFS) controller that produced the best on-average performance at the time of the project, and an unrealizable ideal controller, called the Optimistic controller, able to sustain 100% DRAM throughput if given enough demand by ignoring all timing and resource constraints. They simulated nine memory-intensive parallel workloads consisting of scientific and data-mining applications. Measuring performance as the inverse of execution time normalized to the performance of FR-FCFS, the learning controller showed an average improvement over the FR-FCFS controller of 19%, and closed the gap with the Optimistic's upper bound by 27%. The study also analyzed the impact of on-line learning compared to a previously-learned fixed policy. They trained their controller with data from nine benchmark applications and then held the resulting action values fixed throughout the simulated execution of the applications. They found that the average performance of the controller that learned on-line was 8% better than that of the controller using the fixed policy.

This learning memory controller was never committed to physical hardware because of the large cost of fabrication and the later development of double data rate DRAM. Nevertheless, İpek et al. argued that a memory controller that learns on-line via reinforcement learning has the potential to improve performance to levels that would otherwise

require more complex and more expensive memory systems, while removing from human designers some of the burden required to manually design efficient scheduling policies. The approach is especially promising for developing sophisticated power-aware DRAM interfaces.

IV. HUMAN-LEVEL VIDEO GAME PLAY

Multi-layer artificial neural networks (ANNs) have been used for function approximation in reinforcement learning ever since the 1986 popularization of the backpropagation algorithm, and some striking results have been obtained, such as TD–Gammon [11, 12] and WATSON's DD wagering discussed above. These and other applications benefited from the ability of multi-layer ANNs to learn task-relevant features. However, in all the examples of which we are aware, the most impressive demonstrations required the network's input to be represented in terms of specialized features handcrafted based on human knowledge and intuition about the specific problem to be tackled.

A team of researchers at Google DeepMind [5] developed a reinforcement learning agent called *deep Q-network* (DQN) to show how a single reinforcement learning agent can achieve high levels of performance in many different problems without relying on different problem-specific features. DQN couples a *deep convolutional* ANN [2] with a form of Q-learning modified to improve its speed and stability. The team let DQN learn to play 49 different Atari 2600 video games by interacting with a game emulator. For learning each game, DQN used the same raw input, the same network architecture, and the same parameter values (e.g., step-size, discount rate, exploration parameters, and many more specific to the implementation). DQN achieved levels of play at or beyond human level on a large fraction of these games. Although the games were alike in being played by watching streams of video images, they varied widely in other respects. Their actions had different effects, they had different state-transition dynamics, and they needed different policies for earning high scores. The deep convolutional ANN learned to transform the raw input common to all the games into features specialized for representing the action values required for playing at the high level DQN achieved for most of the games.

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

For an artificial learning system to achieve these levels of play would be impressive enough, but what makes these results remarkable—and what many at the time considered to be breakthrough results for artificial intelligence—is that the very same learning system achieved these levels of play on widely varying games without relying on any game-specific modifications. But as pointed out in [5], DQN is not a complete solution to the problem of task-independent learning. Although the skills needed to excel on the Atari games were markedly diverse, all the games were played by observing video images, which made a deep convolutional ANN a natural choice for this collection of tasks. In addition, DQN's performance on some of the Atari 2600 games fell considerably short of human skill levels on these games. The games most difficult for DQN likely require planning methods that DQN did not include.

V. MASTERING THE GAME OF GO

The ancient Chinese game of Go has challenged artificial intelligence researchers for many decades. Methods that achieve human-level skill, or even superhuman-level skill, in other games have not been successful in producing strong Go programs. Thanks to a very active community of Go programmers and international competitions, the level of Go program play has improved significantly over the years. Until recently, however, no Go program had been able to play anywhere near the level of a human Go master. A Google DeepMind team [6] developed a program called AlphaGo that broke this barrier by combining deep convolutional ANNs, supervised learning, Monte Carlo tree search (MCTS), and reinforcement learning. By the time of the 2016 publication [6], AlphaGo had been shown to be decisively stronger than other current Go programs, and it had defeated the human European Go champion 5 games to 0. These were the first victories of a Go program over a human professional Go player without handicap in full Go games. Shortly thereafter, AlphaGo went on to stunning victories over an 18-time world champion Go player, winning 4 out of a 5 games in a challenge match, making worldwide headline news. Artificial intelligence researchers thought that it would be many more years, perhaps decades, for a program to reach this level of play.

Go is a game between two players who alternately place black and white 'stones' on unoccupied intersections, or 'points,' on a board with a grid of 19 horizontal and 19

vertical lines. The game's goal is to capture an area of the board larger than that captured by the opponent. Stones are captured according to simple rules. Methods that produce strong play for other games, such as chess, have not worked as well for Go. The search space for Go is significantly larger than that of chess because Go has a larger number of legal moves per position than chess (≈ 250 versus ≈ 35) and Go games tend to involve more moves than chess games (≈ 150 versus ≈ 80). But the size of the search space is not the major factor that makes Go so difficult. Exhaustive search is infeasible for both chess and Go, and Go on smaller boards, e.g., 9×9, has proven to be exceedingly difficult as well. Experts agreed that the major stumbling block to creating stronger-than-amateur Go programs is the difficulty of defining an adequate position evaluation function. A good evaluation function allows search to be truncated at a feasible depth by providing relatively easy-to-compute predictions of what deeper search would likely yield.

AlphaGo used two deep convolutional ANNs: a policy network, and a value network. Each network had 13 layers. The final layer of the policy network had a unit for each point on the 19×19 board, and its output was a probability distribution over legal actions. The value network had a single output unit that produced state-value estimates, that is, estimates of the probability that AlphaGo would win the game from the game position currently represented by the network's input. Each network's input was a $19 \times 19 \times 48$ image stack in which each point on the Go board was represented by the values of 48 binary or integer-valued features. For example, for each point, one feature indicated if the point was occupied by one of AlphaGo's stones, one of its opponent's stones, or was unoccupied, thus providing the "raw" representation of the board configuration. Other features were based on the rules of Go, such as the number of adjacent points that were empty, the number of opponent stones that would be captured by placing a stone there, the number of turns since a stone was placed there, and other feature vectors that the design team considered to be important.

Each network was trained off-line before live play and remained fixed during actual games. The policy network was first trained by supervised learning to predict moves contained in a database of nearly 30 million expert moves. Training took approximately 3 weeks using a distributed implementation of stochastic gradient ascent on 50 processors. It achieved 57% accuracy, compared to best accuracy achieved by other groups at the time of publication of 44.4%. Reinforcement learning was then used to improve this policy. This was done with policy-gradient reinforcement learning on simulated games between the network's current policy and opponents using policies randomly selected from policies produced by earlier iterations of the learning algorithm. By simulating many games in parallel on 50 processors, the DeepMind team trained the network on a million games in a single day. In testing, the final policy won more than 80% of games played against the policy learned by supervised learning, and it won 85% of games played against a Go program using Monte Carlo search that simulated 100,000 games per move. The value network was trained by Monte Carlo policy evaluation on data obtained from a large number of simulated games in which each player used the policy learned by the policy network.

During live play, AlphaGo selected its moves by using the policy and value networks in a novel variant of MCTS, a recent and strikingly successful combination of tree search and Monte Carlo policy evaluation responsible for the impressive gains of the most successful preceding Go programs. MCTS is an enhanced *rollout algorithm* [8] that is executed after encountering each new game state to select the game program's move; it is executed again to select the move for the next state, and so on. Each execution is an iterative process that evaluates possible moves by simulating, or "rolling out", many complete games starting from the current board position. The results of the rolled out games are averaged to estimate the values of states in a search tree that grows as more rollouts are conducted. The core idea of MCTS is to successively focus its rollouts by extending the initial portions of rolled-out games that received high evaluations in earlier simulations.

In AlphaGo's version of MCTS, the policy network guides the initial part of each rollout to a leaf node, s_L, of its current search tree, with the rest of each rollout played with both sides using a "rollout policy" produced by a simpler and faster network that could be executed quickly enough to allow a large number of rollouts to be carried out during the available decision time. (This network was trained by supervised learning on a corpus of 8 million human moves and allowed approximately 1,000 complete game simulations per second to be run on each of the processing threads that AlphaGo used.) To decide if s_L is promising enough to expand the tree by adding some of its successor nodes to the tree, it is evaluated in two ways. In contrast to basic MCTS, which evaluates a node solely on the basis of the return of a rollout passing through it, AlphaGo's variant of MCTS, called "asynchronous policy and value MCTS," combined this rollout value estimate with the value produced by the value network previously trained by reinforcement learning:

$$V(s_L) = (1 - \lambda)v(s_L) + \lambda z_L,$$

where $v(s_L)$ is the output of the value network for the board state s_L, and z_L is the return of the rollout from leaf s_L. The parameter λ controls the mixing of the values resulting from these two evaluation methods.

The DeepMind team evaluated different versions of AlphaGo in order to asses the contributions made by these various components. With $\lambda = 0$, AlphaGo used just the value network without rollouts, and with $\lambda = 1$, evaluation relied just on rollouts. They found that AlphaGo using just the value network played better than the rollout-only AlphaGo, and in fact played better than the strongest of all other Go programs. The best play resulted from setting $\lambda = 0.5$, indicating that combining the value network with rollouts was particularly important to AlphaGo's success. These evaluation methods complemented one another: the

value network evaluated the high-performance policy that was too slow to be used in live play, while rollouts using the weaker but much faster rollout policy were able to add precision to the value network's evaluations for specific states that occurred during games.

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

VI. PERSONALIZED WEB SERVICES

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

A method long used in marketing, called *A/B testing*, is a simple type of reinforcement learning used to decide which of two versions, A or B, of a website users prefer. Because it is non-associative, like a two-armed bandit problem, this approach does not personalize content delivery. Adding context consisting of features describing individual users and the content to be delivered allows personalizing service. This has been formalized as a contextual bandit problem (or an associative reinforcement learning problem) with the objective of maximizing the total number of user clicks. Li, Chu, Langford, and Schapire [3] applied a contextual bandit algorithm to the problem of personalizing the Yahoo! Front Page Today webpage (one of the most visited pages on the internet at the time of their research) by selecting the news story to feature. Their objective was to maximize the *click-through rate* (CTR), which is the ratio of the total number of clicks all users make on a webpage to the total number of visits to the page. Their contextual bandit algorithm improved over a standard non-associative bandit algorithm by 12.5%.

Theocharous, Thomas, and Ghavamzadeh [13] argued that better results are possible by formulating personalized recommendation as an MDP with the objective of maximizing the total number of clicks users make over repeated visits to a website. Policies derived from the contextual bandit formulation do not take long-term effects of actions into account, effectively treating each visit to a website as if it were made by a new visitor uniformly sampled from the population of the website's visitors. By not using the fact that many users repeatedly visit the same websites, greedy policies do not take advantage of possibilities provided by long-term interactions with individual users.

Working at Adobe Systems Incorporated, Theocharous et al. conducted experiments to see if policies designed to maximize clicks over the long term could in fact improve over short-term greedy policies. The Adobe Marketing Cloud, a set of tools that many companies use to to run digital marketing campaigns, provides infrastructure for automating user-targeted advertising and fund-raising campaigns. Actually deploying novel policies using these tools entails significant risk because a new policy may end up performing poorly. For this reason, the research team needed to assess what a policy's performance would be if it were to be actually deployed, but to do so on the basis of data collected under the execution of other policies. A critical aspect of this research, then, was *off-policy* policy evaluation. Further, the team wanted to do this with high confidence to reduce the risk of deploying a new policy. High confidence off-policy evaluation was a central component of this research (see also [14, 15]).

Theocharous et al. [13] compared the results of two algorithms for learning ad recommendation policies. The first algorithm, which they called *greedy optimization*, had the goal of maximizing only the probability of immediate clicks. As in the standard contextual bandit formulation, this algorithm did not take the long-term effects of recommendations into account. The other algorithm, a reinforcement learning algorithm based on an MDP formulation, aimed at improving the number of clicks users made over multiple visits to a website. They called this latter algorithm *life-time value* (LTV) optimization. Both algorithms faced challenging problems because the reward signal in this domain is very sparse since users usually do not click on ads, and user clicking is very random so that returns have high variance.

Data sets from the banking industry were used for training and testing these algorithms. The data sets consisted of many complete trajectories of customer interaction with a bank's website that showed each customer one out of a collection of possible offers. If a customer clicked, the reward was 1, and otherwise it was 0. One data set contained approximately 200,000 interactions from a month of a bank's campaign that randomly offered one of 7 offers. The other data set from another bank's campaign contained 4,000,000 interactions involving 12 possible offers. All interactions included customer features such as the time since the customer's last visit to the website, the number of their visits so far, the last time the customer clicked, geographic location, one of a collection of interests, and features giving demographic information.

LTV optimization used a batch-mode reinforcement learning algorithm called *fitted Q iteration* (FQI). Batch mode means that the entire data set for learning is available from the start, as opposed to the on-line mode of the algorithms in which data are acquired sequentially while the learning algorithm executes. Batch-mode reinforcement learning algorithms are sometimes necessary when on-line learning is not practical, and they can use any batch-mode supervised learning regression algorithm, including algorithms known to scale well to high-dimensional spaces.

To measure the performance of the policies produced by

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

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

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

Because LTV is larger than CTR to the extent that individual users revisit the site, it is an indicator of how successful a policy is in encouraging users to engage in extended interactions with the site.

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

VII. CONCLUSION

The reinforcement learning applications described here illustrate how reinforcement learning has achieved impressive results over a diverse set of challenging problems. The daily-double wagering strategy developed for IBM's **WATSON** *Jeopardy!* player was learned through simulated games against models of human players. It went well beyond what human players are able to do in live game play, and it was an important contributor to WATSON's impressive winning performance. The reinforcement learning DRAM memory controller illustrates how on-line reinforcement learning implemented in hardware has the potential to improve computer performance to levels that would otherwise require more complex and more expensive systems. The DQN video game player illustrates how reinforcement learning coupled with deep neural networks can help alleviate the problem of having to handcraft specialized features based on human knowledge and intuition about the specific problem to be tackled. AlphaGo employed deep neural networks together with reinforcement learning and Monte Carlo tree search to achieve stunning victories over human Go masters, a feat that artificial intelligence researchers thought that would not be possible for many years. Batch-mode reinforcement learning applied to the problem of personalizing web services yielded a policy that encourages users to engage in extended interactions with a site. This policy is now a standard component of the Adobe Marketing Cloud.

Taken together, these examples serve as inspiration for applying reinforcement learning to a wide variety of challenging problems of real-world importance.

REFERENCES

[1] E. İpek, O. Mutlu, J. F. Martínez, and R. Caruana. Self-optimizing memory controllers: A reinforcement learning approach. In *35th International Symposium on Computer Architecture, ISCA'08*, pages 39–50. IEEE, 2008.

[2] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[3] L. Li, W. Chu, J. Langford, and R. E. Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th International Conference on World Wide Web*, pages 661–670. ACM, 2010.

[4] J. F. Martínez and E. İpek. Dynamic multicore resource management: A machine learning approach. *Micro, IEEE*, 29(5):8–17, 2009.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[6] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

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

[8] G. Tesauro and G. R. Galperin. On-line policy improvement using monte-carlo search. In *NIPS*, volume 96, pages 1068–1074, 1996.

[9] G. Tesauro, D. C. Gondek, J. Lechner, J. Fan, and J. M. Prager. Simulation, learning, and optimization techniques in watson's game strategies. *IBM Journal of Research and Development*, 56(3.4):16–1–16–11, 2012.

[10] G. Tesauro, D. C. Gondek, J. Lenchner, J. Fan, and J. M. Prager. Analysis of WATSON's strategies for playing Jeopardy! *Journal of Artificial Intelligence Research*, 21:205–251, 2013.

[11] G. J. Tesauro. Practical issues in temporal difference learning. *Machine Learning*, 8:257–277, 1992.

[12] G. J. Tesauro. TD–gammon, a self-teaching backgammon program, achieves master-level play. *Neural Computation*, 6(2):215–219, 1994.

[13] G. Theocharous, P. S. Thomas, and M. Ghavamzadeh. Personalized ad recommendation for life-time value optimization guarantees. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI-15)*, 2015.

[14] P. S. Thomas. *Safe Reinforcement Learning*. PhD thesis, University of Massachusetts Amherst, 2015.

[15] P. S. Thomas, G. Theocharous, and M. Ghavamzadeh. High-confidence off-policy evaluation. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, pages 3000–3006. The AAAI Press, Palo Alto, CA, 2015.