

Reinforcement Learning

An Introduction
second edition

Richard S. Sutton and Andrew G. Barto



Adaptive Computation and Machine Learning

Francis Bach

A complete list of books published in the Adaptive Computation and Machine Learning series appears at the back of this book.

The cover design is based on the trajectories of a simulated bicycle controlled by a reinforcement learning system developed by Jette Randløv.

Reinforcement Learning:

An Introduction

second edition

Richard S. Sutton and Andrew G. Barto

The MIT Press

Cambridge, Massachusetts

London, England

©2018 Richard S. Sutton and Andrew G. Barto

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivs 2.0 Generic License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc-nd/2.0/> or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.

This book was set in 10/12, CMR by Westchester Publishing Services. Printed and bound in the United States of America.

Library of Congress Cataloging-in-Publication Data

Names: Sutton, Richard S., author. | Barto, Andrew G., author.

Title: Reinforcement learning: an introduction / Richard S. Sutton and Andrew G. Barto.

Description: Second edition. | Cambridge, MA : The MIT Press, [2018] | Series: Adaptive computation and machine learning series | Includes bibliographical references and index.

Identifiers: LCCN 2018023826 | ISBN 9780262039246 (hardcover : alk. paper)

Subjects: LCSH: Reinforcement learning

Classification: LCC Q325.6 .R45 2018 | DDC 006.3/1--dc23 LC record available
at <https://lccn.loc.gov/2018023826>

10 9 8 7 6 5 4 3 2 1

In memory of A. Harry Klopf

Contents

Preface to the Second Edition	xiii
Preface to the First Edition	xvii
Summary of Notation	xix
1 Introduction	1
1.1 Reinforcement Learning	1
1.2 Examples	4
1.3 Elements of Reinforcement Learning	6
1.4 Limitations and Scope	7
1.5 An Extended Example: Tic-Tac-Toe	8
1.6 Summary	13
1.7 Early History of Reinforcement Learning	13
I Tabular Solution Methods	23
2 Multi-armed Bandits	25
2.1 A k -armed Bandit Problem	25
2.2 Action-value Methods	27
2.3 The 10-armed Testbed	28
2.4 Incremental Implementation	30
2.5 Tracking a Nonstationary Problem	32
2.6 Optimistic Initial Values	34
2.7 Upper-Confidence-Bound Action Selection	35
2.8 Gradient Bandit Algorithms	37
2.9 Associative Search (Contextual Bandits)	41
2.10 Summary	42

3 Finite Markov Decision Processes	47
3.1 The Agent–Environment Interface	47
3.2 Goals and Rewards	53
3.3 Returns and Episodes	54
3.4 Unified Notation for Episodic and Continuing Tasks	57
3.5 Policies and Value Functions	58
3.6 Optimal Policies and Optimal Value Functions	62
3.7 Optimality and Approximation	67
3.8 Summary	68
4 Dynamic Programming	73
4.1 Policy Evaluation (Prediction)	74
4.2 Policy Improvement	76
4.3 Policy Iteration	80
4.4 Value Iteration	82
4.5 Asynchronous Dynamic Programming	85
4.6 Generalized Policy Iteration	86
4.7 Efficiency of Dynamic Programming	87
4.8 Summary	88
5 Monte Carlo Methods	91
5.1 Monte Carlo Prediction	92
5.2 Monte Carlo Estimation of Action Values	96
5.3 Monte Carlo Control	97
5.4 Monte Carlo Control without Exploring Starts	100
5.5 Off-policy Prediction via Importance Sampling	103
5.6 Incremental Implementation	109
5.7 Off-policy Monte Carlo Control	110
5.8 *Discounting-aware Importance Sampling	112
5.9 *Per-decision Importance Sampling	114
5.10 Summary	115
6 Temporal-Difference Learning	119
6.1 TD Prediction	119
6.2 Advantages of TD Prediction Methods	124
6.3 Optimality of TD(0)	126
6.4 Sarsa: On-policy TD Control	129
6.5 Q-learning: Off-policy TD Control	131
6.6 Expected Sarsa	133
6.7 Maximization Bias and Double Learning	134
6.8 Games, Afterstates, and Other Special Cases	136
6.9 Summary	138

7 n-step Bootstrapping	141
7.1 n-step TD Prediction	142
7.2 n-step Sarsa	145
7.3 n-step Off-policy Learning	148
7.4 *Per-decision Methods with Control Variates	150
7.5 Off-policy Learning Without Importance Sampling: The n -step Tree Backup Algorithm	152
7.6 *A Unifying Algorithm: n -step $Q(\sigma)$	154
7.7 Summary	157
8 Planning and Learning with Tabular Methods	159
8.1 Models and Planning	159
8.2 Dyna: Integrated Planning, Acting, and Learning	161
8.3 When the Model Is Wrong	166
8.4 Prioritized Sweeping	168
8.5 Expected vs. Sample Updates	172
8.6 Trajectory Sampling	174
8.7 Real-time Dynamic Programming	177
8.8 Planning at Decision Time	180
8.9 Heuristic Search	181
8.10 Rollout Algorithms	183
8.11 Monte Carlo Tree Search	185
8.12 Summary of the Chapter	188
8.13 Summary of Part I: Dimensions	189
II Approximate Solution Methods	195
9 On-policy Prediction with Approximation	197
9.1 Value-function Approximation	198
9.2 The Prediction Objective (VE)	199
9.3 Stochastic-gradient and Semi-gradient Methods	200
9.4 Linear Methods	204
9.5 Feature Construction for Linear Methods	210
9.5.1 Polynomials	210
9.5.2 Fourier Basis	211
9.5.3 Coarse Coding	215
9.5.4 Tile Coding	217
9.5.5 Radial Basis Functions	221
9.6 Selecting Step-Size Parameters Manually	222
9.7 Nonlinear Function Approximation: Artificial Neural Networks	223
9.8 Least-Squares TD	228

9.9	Memory-based Function Approximation	230
9.10	Kernel-based Function Approximation	232
9.11	Looking Deeper at On-policy Learning: Interest and Emphasis	234
9.12	Summary	236
10	On-policy Control with Approximation	243
10.1	Episodic Semi-gradient Control	243
10.2	Semi-gradient n -step Sarsa	247
10.3	Average Reward: A New Problem Setting for Continuing Tasks	249
10.4	Deprecating the Discounted Setting	253
10.5	Differential Semi-gradient n -step Sarsa	255
10.6	Summary	256
11	*Off-policy Methods with Approximation	257
11.1	Semi-gradient Methods	258
11.2	Examples of Off-policy Divergence	260
11.3	The Deadly Triad	264
11.4	Linear Value-function Geometry	266
11.5	Gradient Descent in the Bellman Error	269
11.6	The Bellman Error is Not Learnable	274
11.7	Gradient-TD Methods	278
11.8	Emphatic-TD Methods	281
11.9	Reducing Variance	283
11.10	Summary	284
12	Eligibility Traces	287
12.1	The λ -return	288
12.2	$TD(\lambda)$	292
12.3	n -step Truncated λ -return Methods	295
12.4	Redoing Updates: Online λ -return Algorithm	297
12.5	True Online $TD(\lambda)$	299
12.6	*Dutch Traces in Monte Carlo Learning	301
12.7	$Sarsa(\lambda)$	303
12.8	Variable λ and γ	307
12.9	*Off-policy Traces with Control Variates	309
12.10	Watkins's $Q(\lambda)$ to Tree-Backup(λ)	312
12.11	Stable Off-policy Methods with Traces	314
12.12	Implementation Issues	316
12.13	Conclusions	317

13 Policy Gradient Methods	321
13.1 Policy Approximation and its Advantages	322
13.2 The Policy Gradient Theorem	324
13.3 REINFORCE: Monte Carlo Policy Gradient	326
13.4 REINFORCE with Baseline	329
13.5 Actor–Critic Methods	331
13.6 Policy Gradient for Continuing Problems	333
13.7 Policy Parameterization for Continuous Actions	335
13.8 Summary	337
III Looking Deeper	339
14 Psychology	341
14.1 Prediction and Control	342
14.2 Classical Conditioning	343
14.2.1 Blocking and Higher-order Conditioning	345
14.2.2 The Rescorla–Wagner Model	346
14.2.3 The TD Model	349
14.2.4 TD Model Simulations	350
14.3 Instrumental Conditioning	357
14.4 Delayed Reinforcement	361
14.5 Cognitive Maps	363
14.6 Habitual and Goal-directed Behavior	364
14.7 Summary	368
15 Neuroscience	377
15.1 Neuroscience Basics	378
15.2 Reward Signals, Reinforcement Signals, Values, and Prediction Errors	380
15.3 The Reward Prediction Error Hypothesis	381
15.4 Dopamine	383
15.5 Experimental Support for the Reward Prediction Error Hypothesis	387
15.6 TD Error/Dopamine Correspondence	390
15.7 Neural Actor–Critic	395
15.8 Actor and Critic Learning Rules	398
15.9 Hedonistic Neurons	402
15.10 Collective Reinforcement Learning	404
15.11 Model-based Methods in the Brain	407
15.12 Addiction	409
15.13 Summary	410

16 Applications and Case Studies	421
16.1 TD-Gammon	421
16.2 Samuel’s Checkers Player	426
16.3 Watson’s Daily-Double Wagering	429
16.4 Optimizing Memory Control	432
16.5 Human-level Video Game Play	436
16.6 Mastering the Game of Go	441
16.6.1 AlphaGo	444
16.6.2 AlphaGo Zero	447
16.7 Personalized Web Services	450
16.8 Thermal Soaring	453
17 Frontiers	459
17.1 General Value Functions and Auxiliary Tasks	459
17.2 Temporal Abstraction via Options	461
17.3 Observations and State	464
17.4 Designing Reward Signals	469
17.5 Remaining Issues	472
17.6 The Future of Artificial Intelligence	475
References	481
Index	519

Preface to the Second Edition

The twenty years since the publication of the first edition of this book have seen tremendous progress in artificial intelligence, propelled in large part by advances in machine learning, including advances in reinforcement learning. Although the impressive computational power that became available is responsible for some of these advances, new developments in theory and algorithms have been driving forces as well. In the face of this progress, a second edition of our 1998 book was long overdue, and we finally began the project in 2012. Our goal for the second edition was the same as our goal for the first: to provide a clear and simple account of the key ideas and algorithms of reinforcement learning that is accessible to readers in all the related disciplines. The edition remains an introduction, and we retain a focus on core, online learning algorithms. This edition includes some new topics that rose to importance over the intervening years, and we expanded coverage of topics that we now understand better. But we made no attempt to provide comprehensive coverage of the field, which has exploded in many different directions. We apologize for having to leave out all but a handful of these contributions.

As in the first edition, we chose not to produce a rigorous formal treatment of reinforcement learning, or to formulate it in the most general terms. However, our deeper understanding of some topics since the first edition required a bit more mathematics to explain; we have set off the more mathematical parts in shaded boxes that the non-mathematically-inclined may choose to skip. We also use a slightly different notation than was used in the first edition. In teaching, we have found that the new notation helps to address some common points of confusion. It emphasizes the difference between random variables, denoted with capital letters, and their instantiations, denoted in lower case. For example, the state, action, and reward at time step t are denoted S_t , A_t , and R_t , while their possible values might be denoted s , a , and r . Along with this, it is natural to use lower case for value functions (e.g., v_π) and restrict capitals to their tabular estimates (e.g., $Q_t(s, a)$). Approximate value functions are deterministic functions of random parameters and are thus also in lower case (e.g., $\hat{v}(s, \mathbf{w}_t) \approx v_\pi(s)$). Vectors, such as the weight vector \mathbf{w}_t (formerly $\boldsymbol{\theta}_t$) and the feature vector \mathbf{x}_t (formerly ϕ_t), are bold and written in lowercase even if they are random variables. Uppercase bold is reserved for matrices. In the first edition we used special notations, $\mathcal{P}_{ss'}^a$ and $\mathcal{R}_{ss'}^a$, for the transition probabilities and expected rewards. One weakness of that notation is that it still did not fully characterize the dynamics of the rewards, giving only their expectations, which is sufficient for dynamic programming but not for reinforcement learning. Another weakness

is the excess of subscripts and superscripts. In this edition we use the explicit notation of $p(s', r | s, a)$ for the joint probability for the next state and reward given the current state and action. All the changes in notation are summarized in a table on page xix.

The second edition is significantly expanded, and its top-level organization has been changed. After the introductory first chapter, the second edition is divided into three new parts. The first part (Chapters 2–8) treats as much of reinforcement learning as possible without going beyond the tabular case for which exact solutions can be found. We cover both learning and planning methods for the tabular case, as well as their unification in n -step methods and in Dyna. Many algorithms presented in this part are new to the second edition, including UCB, Expected Sarsa, Double learning, tree-backup, $Q(\sigma)$, RTDP, and MCTS. Doing the tabular case first, and thoroughly, enables core ideas to be developed in the simplest possible setting. The second part of the book (Chapters 9–13) is then devoted to extending the ideas to function approximation. It has new sections on artificial neural networks, the fourier basis, LSTD, kernel-based methods, Gradient-TD and Emphatic-TD methods, average-reward methods, true online TD(λ), and policy-gradient methods. The second edition significantly expands the treatment of off-policy learning, first for the tabular case in Chapters 5–7, then with function approximation in Chapters 11 and 12. Another change is that the second edition separates the forward-view idea of n -step bootstrapping (now treated more fully in Chapter 7) from the backward-view idea of eligibility traces (now treated independently in Chapter 12). The third part of the book has large new chapters on reinforcement learning’s relationships to psychology (Chapter 14) and neuroscience (Chapter 15), as well as an updated case-studies chapter including Atari game playing, Watson’s wagering strategy, and the Go playing programs AlphaGo and AlphaGo Zero (Chapter 16). Still, out of necessity we have included only a small subset of all that has been done in the field. Our choices reflect our long-standing interests in inexpensive model-free methods that should scale well to large applications. The final chapter now includes a discussion of the future societal impacts of reinforcement learning. For better or worse, the second edition is about twice as large as the first.

This book is designed to be used as the primary text for a one- or two-semester course on reinforcement learning. For a one-semester course, the first ten chapters should be covered in order and form a good core, to which can be added material from the other chapters, from other books such as Bertsekas and Tsitsiklis (1996), Wiering and van Otterlo (2012), and Szepesvári (2010), or from the literature, according to taste. Depending on the students’ background, some additional material on online supervised learning may be helpful. The ideas of options and option models are a natural addition (Sutton, Precup and Singh, 1999). A two-semester course can cover all the chapters as well as supplementary material. The book can also be used as part of broader courses on machine learning, artificial intelligence, or neural networks. In this case, it may be desirable to cover only a subset of the material. We recommend covering Chapter 1 for a brief overview, Chapter 2 through Section 2.4, Chapter 3, and then selecting sections from the remaining chapters according to time and interests. Chapter 6 is the most important for the subject and for the rest of the book. A course focusing on machine learning or neural networks should cover Chapters 9 and 10, and a course focusing on artificial intelligence or planning should cover Chapter 8. Throughout the book, sections and chapters that are more difficult and not essential to the rest of the book are marked

with a *. These can be omitted on first reading without creating problems later on. Some exercises are also marked with a * to indicate that they are more advanced and not essential to understanding the basic material of the chapter.

Most chapters end with a section entitled “Bibliographical and Historical Remarks,” wherein we credit the sources of the ideas presented in that chapter, provide pointers to further reading and ongoing research, and describe relevant historical background. Despite our attempts to make these sections authoritative and complete, we have undoubtedly left out some important prior work. For that we again apologize, and we welcome corrections and extensions for incorporation into the electronic version of the book.

Like the first edition, this edition of the book is dedicated to the memory of A. Harry Klopf. It was Harry who introduced us to each other, and it was his ideas about the brain and artificial intelligence that launched our long excursion into reinforcement learning. Trained in neurophysiology and long interested in machine intelligence, Harry was a senior scientist affiliated with the Avionics Directorate of the Air Force Office of Scientific Research (AFOSR) at Wright-Patterson Air Force Base, Ohio. He was dissatisfied with the great importance attributed to equilibrium-seeking processes, including homeostasis and error-correcting pattern classification methods, in explaining natural intelligence and in providing a basis for machine intelligence. He noted that systems that try to maximize something (whatever that might be) are qualitatively different from equilibrium-seeking systems, and he argued that maximizing systems hold the key to understanding important aspects of natural intelligence and for building artificial intelligences. Harry was instrumental in obtaining funding from AFOSR for a project to assess the scientific merit of these and related ideas. This project was conducted in the late 1970s at the University of Massachusetts Amherst (UMass Amherst), initially under the direction of Michael Arbib, William Kilmer, and Nico Spinelli, professors in the Department of Computer and Information Science at UMass Amherst, and founding members of the Cybernetics Center for Systems Neuroscience at the University, a farsighted group focusing on the intersection of neuroscience and artificial intelligence. Barto, a recent Ph.D. from the University of Michigan, was hired as post doctoral researcher on the project. Meanwhile, Sutton, an undergraduate studying computer science and psychology at Stanford, had been corresponding with Harry regarding their mutual interest in the role of stimulus timing in classical conditioning. Harry suggested to the UMass group that Sutton would be a great addition to the project. Thus, Sutton became a UMass graduate student, whose Ph.D. was directed by Barto, who had become an Associate Professor. The study of reinforcement learning as presented in this book is rightfully an outcome of that project instigated by Harry and inspired by his ideas. Further, Harry was responsible for bringing us, the authors, together in what has been a long and enjoyable interaction. By dedicating this book to Harry we honor his essential contributions, not only to the field of reinforcement learning, but also to our collaboration. We also thank Professors Arbib, Kilmer, and Spinelli for the opportunity they provided to us to begin exploring these ideas. Finally, we thank AFOSR for generous support over the early years of our research, and the NSF for its generous support over many of the following years.

We have very many people to thank for their inspiration and help with this second edition. Everyone we acknowledged for their inspiration and help with the first edition

deserve our deepest gratitude for this edition as well, which would not exist were it not for their contributions to edition number one. To that long list we must add many others who contributed specifically to the second edition. Our students over the many years that we have taught this material contributed in countless ways: exposing errors, offering fixes, and—not the least—being confused in places where we could have explained things better. We especially thank Martha Steenstrup for reading and providing detailed comments throughout. The chapters on psychology and neuroscience could not have been written without the help of many experts in those fields. We thank John Moore for his patient tutoring over many many years on animal learning experiments, theory, and neuroscience, and for his careful reading of multiple drafts of Chapters 14 and 15. We also thank Matt Botvinick, Nathaniel Daw, Peter Dayan, and Yael Niv for their penetrating comments on drafts of these chapter, their essential guidance through the massive literature, and their interception of many of our errors in early drafts. Of course, the remaining errors in these chapters—and there must still be some—are totally our own. We thank Phil Thomas for helping us make these chapters accessible to non-psychologists and non-neuroscientists, and we thank Peter Sterling for helping us improve the exposition. We are grateful to Jim Houk for introducing us to the subject of information processing in the basal ganglia and for alerting us to other relevant aspects of neuroscience. José Martínez, Terry Sejnowski, David Silver, Gerry Tesrauro, Georgios Theoccharous, and Phil Thomas generously helped us understand details of their reinforcement learning applications for inclusion in the case-studies chapter, and they provided helpful comments on drafts of these sections. Special thanks are owed to David Silver for helping us better understand Monte Carlo Tree Search and the DeepMind Go-playing programs. We thank George Konidaris for his help with the section on the Fourier basis. Emilio Cartoni, Thomas Cederborg, Stefan Dernbach, Clemens Rosenbaum, Patrick Taylor, Thomas Colin, and Pierre-Luc Bacon helped us in a number important ways for which we are most grateful.

Sutton would also like to thank the members of the Reinforcement Learning and Artificial Intelligence laboratory at the University of Alberta for contributions to the second edition. He owes a particular debt to Rupam Mahmood for essential contributions to the treatment of off-policy Monte Carlo methods in Chapter 5, to Hamid Maei for helping develop the perspective on off-policy learning presented in Chapter 11, to Eric Graves for conducting the experiments in Chapter 13, to Shangtong Zhang for replicating and thus verifying almost all the experimental results, to Kris De Asis for improving the new technical content of Chapters 7 and 12, and to Harm van Seijen for insights that led to the separation of n -step methods from eligibility traces and (along with Hado van Hasselt) for the ideas involving exact equivalence of forward and backward views of eligibility traces presented in Chapter 12. Sutton also gratefully acknowledges the support and freedom he was granted by the Government of Alberta and the National Science and Engineering Research Council of Canada throughout the period during which the second edition was conceived and written. In particular, he would like to thank Randy Goebel for creating a supportive and far-sighted environment for research in Alberta. He would also like to thank DeepMind their support in the last six months of writing the book.

Finally, we owe thanks to the many careful readers of drafts of the second edition that we posted on the internet. They found many errors that we had missed and alerted us to potential points of confusion.

Preface to the First Edition

We first came to focus on what is now known as reinforcement learning in late 1979. We were both at the University of Massachusetts, working on one of the earliest projects to revive the idea that networks of neuronlike adaptive elements might prove to be a promising approach to artificial adaptive intelligence. The project explored the “heterostatic theory of adaptive systems” developed by A. Harry Klopf. Harry’s work was a rich source of ideas, and we were permitted to explore them critically and compare them with the long history of prior work in adaptive systems. Our task became one of teasing the ideas apart and understanding their relationships and relative importance. This continues today, but in 1979 we came to realize that perhaps the simplest of the ideas, which had long been taken for granted, had received surprisingly little attention from a computational perspective. This was simply the idea of a learning system that *wants* something, that adapts its behavior in order to maximize a special signal from its environment. This was the idea of a “hedonistic” learning system, or, as we would say now, the idea of reinforcement learning.

Like others, we had a sense that reinforcement learning had been thoroughly explored in the early days of cybernetics and artificial intelligence. On closer inspection, though, we found that it had been explored only slightly. While reinforcement learning had clearly motivated some of the earliest computational studies of learning, most of these researchers had gone on to other things, such as pattern classification, supervised learning, and adaptive control, or they had abandoned the study of learning altogether. As a result, the special issues involved in learning how to get something from the environment received relatively little attention. In retrospect, focusing on this idea was the critical step that set this branch of research in motion. Little progress could be made in the computational study of reinforcement learning until it was recognized that such a fundamental idea had not yet been thoroughly explored.

The field has come a long way since then, evolving and maturing in several directions. Reinforcement learning has gradually become one of the most active research areas in machine learning, artificial intelligence, and neural network research. The field has developed strong mathematical foundations and impressive applications. The computational study of reinforcement learning is now a large field, with hundreds of active researchers around the world in diverse disciplines such as psychology, control theory, artificial intelligence, and neuroscience. Particularly important have been the contributions establishing and developing the relationships to the theory of optimal control and dynamic programming.

The overall problem of learning from interaction to achieve goals is still far from being solved, but our understanding of it has improved significantly. We can now place component ideas, such as temporal-difference learning, dynamic programming, and function approximation, within a coherent perspective with respect to the overall problem.

Our goal in writing this book was to provide a clear and simple account of the key ideas and algorithms of reinforcement learning. We wanted our treatment to be accessible to readers in all of the related disciplines, but we could not cover all of these perspectives in detail. For the most part, our treatment takes the point of view of artificial intelligence and engineering. Coverage of connections to other fields we leave to others or to another time. We also chose not to produce a rigorous formal treatment of reinforcement learning. We did not reach for the highest possible level of mathematical abstraction and did not rely on a theorem–proof format. We tried to choose a level of mathematical detail that points the mathematically inclined in the right directions without distracting from the simplicity and potential generality of the underlying ideas.

...

In some sense we have been working toward this book for thirty years, and we have lots of people to thank. First, we thank those who have personally helped us develop the overall view presented in this book: Harry Klopf, for helping us recognize that reinforcement learning needed to be revived; Chris Watkins, Dimitri Bertsekas, John Tsitsiklis, and Paul Werbos, for helping us see the value of the relationships to dynamic programming; John Moore and Jim Kehoe, for insights and inspirations from animal learning theory; Oliver Selfridge, for emphasizing the breadth and importance of adaptation; and, more generally, our colleagues and students who have contributed in countless ways: Ron Williams, Charles Anderson, Satinder Singh, Sridhar Mahadevan, Steve Bradtke, Bob Crites, Peter Dayan, and Leemon Baird. Our view of reinforcement learning has been significantly enriched by discussions with Paul Cohen, Paul Utgoff, Martha Steenstrup, Gerry Tesauro, Mike Jordan, Leslie Kaelbling, Andrew Moore, Chris Atkeson, Tom Mitchell, Nils Nilsson, Stuart Russell, Tom Dietterich, Tom Dean, and Bob Narendra. We thank Michael Littman, Gerry Tesauro, Bob Crites, Satinder Singh, and Wei Zhang for providing specifics of Sections 4.7, 15.1, 15.4, 15.4, and 15.6 respectively. We thank the Air Force Office of Scientific Research, the National Science Foundation, and GTE Laboratories for their long and farsighted support.

We also wish to thank the many people who have read drafts of this book and provided valuable comments, including Tom Kalt, John Tsitsiklis, Paweł Cichosz, Olle Gällmo, Chuck Anderson, Stuart Russell, Ben Van Roy, Paul Steenstrup, Paul Cohen, Sridhar Mahadevan, Jette Randlov, Brian Sheppard, Thomas O’Connell, Richard Coggins, Cristina Versino, John H. Hiett, Andreas Badelt, Jay Ponte, Joe Beck, Justus Piater, Martha Steenstrup, Satinder Singh, Tommi Jaakkola, Dimitri Bertsekas, Torbjörn Ekman, Christina Björkman, Jakob Carlström, and Olle Palmgren. Finally, we thank Gwyn Mitchell for helping in many ways, and Harry Stanton and Bob Prior for being our champions at MIT Press.

Summary of Notation

Capital letters are used for random variables, whereas lower case letters are used for the values of random variables and for scalar functions. Quantities that are required to be real-valued vectors are written in bold and in lower case (even if random variables). Matrices are bold capitals.

\doteq	equality relationship that is true by definition
\approx	approximately equal
\propto	proportional to
$\Pr\{X=x\}$	probability that a random variable X takes on the value x
$X \sim p$	random variable X selected from distribution $p(x) \doteq \Pr\{X=x\}$
$\mathbb{E}[X]$	expectation of a random variable X , i.e., $\mathbb{E}[X] \doteq \sum_x p(x)x$
$\arg\max_a f(a)$	a value of a at which $f(a)$ takes its maximal value
$\ln x$	natural logarithm of x
e^x	the base of the natural logarithm, $e \approx 2.71828$, carried to power x ; $e^{\ln x} = x$
\mathbb{R}	set of real numbers
$f : \mathcal{X} \rightarrow \mathcal{Y}$	function f from elements of set \mathcal{X} to elements of set \mathcal{Y}
\leftarrow	assignment
$(a, b]$	the real interval between a and b including b but not including a
ε	probability of taking a random action in an ε -greedy policy
α, β	step-size parameters
γ	discount-rate parameter
λ	decay-rate parameter for eligibility traces
$\mathbb{1}_{predicate}$	indicator function ($\mathbb{1}_{predicate} \doteq 1$ if the <i>predicate</i> is true, else 0)

In a multi-arm bandit problem:

k	number of actions (arms)
t	discrete time step or play number
$q_*(a)$	true value (expected reward) of action a
$Q_t(a)$	estimate at time t of $q_*(a)$
$N_t(a)$	number of times action a has been selected up prior to time t
$H_t(a)$	learned preference for selecting action a at time t
$\pi_t(a)$	probability of selecting action a at time t
\bar{R}_t	estimate at time t of the expected reward given π_t

In a Markov Decision Process:

s, s'	states
a	an action
r	a reward
\mathcal{S}	set of all nonterminal states
\mathcal{S}^+	set of all states, including the terminal state
$\mathcal{A}(s)$	set of all actions available in state s
\mathcal{R}	set of all possible rewards, a finite subset of \mathbb{R}
\subset	subset of; e.g., $\mathcal{R} \subset \mathbb{R}$
\in	is an element of; e.g., $s \in \mathcal{S}, r \in \mathcal{R}$
$ \mathcal{S} $	number of elements in set \mathcal{S}
t	discrete time step
$T, T(t)$	final time step of an episode, or of the episode including time step t
A_t	action at time t
S_t	state at time t , typically due, stochastically, to S_{t-1} and A_{t-1}
R_t	reward at time t , typically due, stochastically, to S_{t-1} and A_{t-1}
π	policy (decision-making rule)
$\pi(s)$	action taken in state s under <i>deterministic</i> policy π
$\pi(a s)$	probability of taking action a in state s under <i>stochastic</i> policy π
G_t	return following time t
h	horizon, the time step one looks up to in a forward view
$G_{t:t+n}, G_{t:h}$	n -step return from $t+1$ to $t+n$, or to h (discounted and corrected)
$\bar{G}_{t:h}$	flat return (undiscounted and uncorrected) from $t+1$ to h (Section 5.8)
G_t^λ	λ -return (Section 12.1)
$G_{t:h}^\lambda$	truncated, corrected λ -return (Section 12.3)
$G_t^{\lambda_s}, G_t^{\lambda_a}$	λ -return, corrected by estimated state, or action, values (Section 12.8)
$p(s', r s, a)$	probability of transition to state s' with reward r , from state s and action a
$p(s' s, a)$	probability of transition to state s' , from state s taking action a
$r(s, a)$	expected immediate reward from state s after action a
$r(s, a, s')$	expected immediate reward on transition from s to s' under action a
$v_\pi(s)$	value of state s under policy π (expected return)
$v_*(s)$	value of state s under the optimal policy
$q_\pi(s, a)$	value of taking action a in state s under policy π
$q_*(s, a)$	value of taking action a in state s under the optimal policy
V, V_t	array estimates of state-value function v_π or v_*
Q, Q_t	array estimates of action-value function q_π or q_*
$\bar{V}_t(s)$	expected approximate action value, e.g., $\bar{V}_t(s) \doteq \sum_a \pi(a s) Q_t(s, a)$
U_t	target for estimate at time t

δ_t	temporal-difference (TD) error at t (a random variable) (Section 6.1)
δ_t^s, δ_t^a	state- and action-specific forms of the TD error (Section 12.9)
n	in n -step methods, n is the number of steps of bootstrapping
d	dimensionality—the number of components of \mathbf{w}
d'	alternate dimensionality—the number of components of $\boldsymbol{\theta}$
\mathbf{w}, \mathbf{w}_t	d -vector of weights underlying an approximate value function
$w_i, w_{t,i}$	i th component of learnable weight vector
$\hat{v}(s, \mathbf{w})$	approximate value of state s given weight vector \mathbf{w}
$v_{\mathbf{w}}(s)$	alternate notation for $\hat{v}(s, \mathbf{w})$
$\hat{q}(s, a, \mathbf{w})$	approximate value of state-action pair s, a given weight vector \mathbf{w}
$\nabla \hat{v}(s, \mathbf{w})$	column vector of partial derivatives of $\hat{v}(s, \mathbf{w})$ with respect to \mathbf{w}
$\nabla \hat{q}(s, a, \mathbf{w})$	column vector of partial derivatives of $\hat{q}(s, a, \mathbf{w})$ with respect to \mathbf{w}
$\mathbf{x}(s)$	vector of features visible when in state s
$\mathbf{x}(s, a)$	vector of features visible when in state s taking action a
$x_i(s), x_i(s, a)$	i th component of vector $\mathbf{x}(s)$ or $\mathbf{x}(s, a)$
\mathbf{x}_t	shorthand for $\mathbf{x}(S_t)$ or $\mathbf{x}(S_t, A_t)$
$\mathbf{w}^\top \mathbf{x}$	inner product of vectors, $\mathbf{w}^\top \mathbf{x} \doteq \sum_i w_i x_i$; e.g., $\hat{v}(s, \mathbf{w}) \doteq \mathbf{w}^\top \mathbf{x}(s)$
\mathbf{v}, \mathbf{v}_t	secondary d -vector of weights, used to learn \mathbf{w} (Chapter 11)
\mathbf{z}_t	d -vector of eligibility traces at time t (Chapter 12)
$\boldsymbol{\theta}, \boldsymbol{\theta}_t$	parameter vector of target policy (Chapter 13)
$\pi(a s, \boldsymbol{\theta})$	probability of taking action a in state s given parameter vector $\boldsymbol{\theta}$
$\pi_{\boldsymbol{\theta}}$	policy corresponding to parameter $\boldsymbol{\theta}$
$\nabla \pi(a s, \boldsymbol{\theta})$	column vector of partial derivatives of $\pi(a s, \boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$
$J(\boldsymbol{\theta})$	performance measure for the policy $\pi_{\boldsymbol{\theta}}$
$\nabla J(\boldsymbol{\theta})$	column vector of partial derivatives of $J(\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$
$h(s, a, \boldsymbol{\theta})$	preference for selecting action a in state s based on $\boldsymbol{\theta}$
$b(a s)$	behavior policy used to select actions while learning about target policy π
$b(s)$	a baseline function $b : \mathcal{S} \mapsto \mathbb{R}$ for policy-gradient methods
b	branching factor for an MDP or search tree
$\rho_{t:h}$	importance sampling ratio for time t through time h (Section 5.5)
ρ_t	importance sampling ratio for time t alone, $\rho_t \doteq \rho_{t:t}$
$r(\pi)$	average reward (reward rate) for policy π (Section 10.3)
\bar{R}_t	estimate of $r(\pi)$ at time t
$\mu(s)$	on-policy distribution over states (Section 9.2)
$\boldsymbol{\mu}$	$ \mathcal{S} $ -vector of the $\mu(s)$ for all $s \in \mathcal{S}$
$\ v\ _{\mu}^2$	μ -weighted squared norm of value function v , i.e., $\ v\ _{\mu}^2 \doteq \sum_{s \in \mathcal{S}} \mu(s) v(s)^2$
$\eta(s)$	expected number of visits to state s per episode (page 199)
Π	projection operator for value functions (page 268)
B_{π}	Bellman operator for value functions (Section 11.4)

A	$d \times d$ matrix $\mathbf{A} \doteq \mathbb{E} \left[\mathbf{x}_t (\mathbf{x}_t - \gamma \mathbf{x}_{t+1})^\top \right]$
b	d -dimensional vector $\mathbf{b} \doteq \mathbb{E}[R_{t+1} \mathbf{x}_t]$
\mathbf{w}_{TD}	TD fixed point $\mathbf{w}_{\text{TD}} \doteq \mathbf{A}^{-1} \mathbf{b}$ (a d -vector, Section 9.4)
I	identity matrix
P	$ \mathcal{S} \times \mathcal{S} $ matrix of state-transition probabilities under π
D	$ \mathcal{S} \times \mathcal{S} $ diagonal matrix with μ on its diagonal
X	$ \mathcal{S} \times d$ matrix with the $\mathbf{x}(s)$ as its rows
$\bar{\delta}_{\mathbf{w}}(s)$	Bellman error (expected TD error) for $v_{\mathbf{w}}$ at state s (Section 11.4)
$\bar{\delta}_{\mathbf{w}}$, BE	Bellman error vector, with components $\bar{\delta}_{\mathbf{w}}(s)$
$\overline{\text{VE}}(\mathbf{w})$	mean square value error $\overline{\text{VE}}(\mathbf{w}) \doteq \ v_{\mathbf{w}} - v_{\pi}\ _{\mu}^2$ (Section 9.2)
$\overline{\text{BE}}(\mathbf{w})$	mean square Bellman error $\overline{\text{BE}}(\mathbf{w}) \doteq \ \bar{\delta}_{\mathbf{w}}\ _{\mu}^2$
$\overline{\text{PBE}}(\mathbf{w})$	mean square projected Bellman error $\overline{\text{PBE}}(\mathbf{w}) \doteq \ \Pi \bar{\delta}_{\mathbf{w}}\ _{\mu}^2$
$\overline{\text{TDE}}(\mathbf{w})$	mean square temporal-difference error $\overline{\text{TDE}}(\mathbf{w}) \doteq \mathbb{E}_b[\rho_t \delta_t^2]$ (Section 11.5)
$\overline{\text{RE}}(\mathbf{w})$	mean square return error (Section 11.6)

Chapter 1

Introduction

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

In this book we explore a *computational* approach to learning from interaction. Rather than directly theorizing about how people or animals learn, we primarily explore idealized learning situations and evaluate the effectiveness of various learning methods.¹ That is, we adopt the perspective of an artificial intelligence researcher or engineer. We explore designs for machines that are effective in solving learning problems of scientific or economic interest, evaluating the designs through mathematical analysis or computational experiments. The approach we explore, called *reinforcement learning*, is much more focused on goal-directed learning from interaction than are other approaches to machine learning.

1.1 Reinforcement Learning

Reinforcement learning is learning what to do—how to map situations to actions—so as to maximize a numerical reward signal. The learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them. In the most interesting and challenging cases, actions may affect not only the immediate

¹The relationships to psychology and neuroscience are summarized in Chapters 14 and 15.

reward but also the next situation and, through that, all subsequent rewards. These two characteristics—trial-and-error search and delayed reward—are the two most important distinguishing features of reinforcement learning.

Reinforcement learning, like many topics whose names end with “ing,” such as machine learning and mountaineering, is simultaneously a problem, a class of solution methods that work well on the problem, and the field that studies this problem and its solution methods. It is convenient to use a single name for all three things, but at the same time essential to keep the three conceptually separate. In particular, the distinction between problems and solution methods is very important in reinforcement learning; failing to make this distinction is the source of many confusions.

We formalize the problem of reinforcement learning using ideas from dynamical systems theory, specifically, as the optimal control of incompletely-known Markov decision processes. The details of this formalization must wait until Chapter 3, but the basic idea is simply to capture the most important aspects of the real problem facing a learning agent interacting over time with its environment to achieve a goal. A learning agent must be able to sense the state of its environment to some extent and must be able to take actions that affect the state. The agent also must have a goal or goals relating to the state of the environment. Markov decision processes are intended to include just these three aspects—sensation, action, and goal—in their simplest possible forms without trivializing any of them. Any method that is well suited to solving such problems we consider to be a reinforcement learning method.

Reinforcement learning is different from *supervised learning*, the kind of learning studied in most current research in the field of machine learning. Supervised learning is learning from a training set of labeled examples provided by a knowledgeable external supervisor. Each example is a description of a situation together with a specification—the label—of the correct action the system should take to that situation, which is often to identify a category to which the situation belongs. The object of this kind of learning is for the system to extrapolate, or generalize, its responses so that it acts correctly in situations not present in the training set. This is an important kind of learning, but alone it is not adequate for learning from interaction. In interactive problems it is often impractical to obtain examples of desired behavior that are both correct and representative of all the situations in which the agent has to act. In uncharted territory—where one would expect learning to be most beneficial—an agent must be able to learn from its own experience.

Reinforcement learning is also different from what machine learning researchers call *unsupervised learning*, which is typically about finding structure hidden in collections of unlabeled data. The terms supervised learning and unsupervised learning would seem to exhaustively classify machine learning paradigms, but they do not. Although one might be tempted to think of reinforcement learning as a kind of unsupervised learning because it does not rely on examples of correct behavior, reinforcement learning is trying to maximize a reward signal instead of trying to find hidden structure. Uncovering structure in an agent’s experience can certainly be useful in reinforcement learning, but by itself does not address the reinforcement learning problem of maximizing a reward signal. We therefore consider reinforcement learning to be a third machine learning paradigm, alongside supervised learning and unsupervised learning and perhaps other paradigms.

One of the challenges that arise in reinforcement learning, and not in other kinds of learning, is the trade-off between exploration and exploitation. To obtain a lot of reward, a reinforcement learning agent must prefer actions that it has tried in the past and found to be effective in producing reward. But to discover such actions, it has to try actions that it has not selected before. The agent has to *exploit* what it has already experienced in order to obtain reward, but it also has to *explore* in order to make better action selections in the future. The dilemma is that neither exploration nor exploitation can be pursued exclusively without failing at the task. The agent must try a variety of actions *and* progressively favor those that appear to be best. On a stochastic task, each action must be tried many times to gain a reliable estimate of its expected reward. The exploration-exploitation dilemma has been intensively studied by mathematicians for many decades, yet remains unresolved. For now, we simply note that the entire issue of balancing exploration and exploitation does not even arise in supervised and unsupervised learning, at least in the purest forms of these paradigms.

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

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

By a complete, interactive, goal-seeking agent we do not always mean something like a complete organism or robot. These are clearly examples, but a complete, interactive, goal-seeking agent can also be a component of a larger behaving system. In this case, the agent directly interacts with the rest of the larger system and indirectly interacts with the larger system's environment. A simple example is an agent that monitors the charge level of robot's battery and sends commands to the robot's control architecture. This agent's environment is the rest of the robot together with the robot's environment. One must look beyond the most obvious examples of agents and their environments to

appreciate the generality of the reinforcement learning framework.

One of the most exciting aspects of modern reinforcement learning is its substantive and fruitful interactions with other engineering and scientific disciplines. Reinforcement learning is part of a decades-long trend within artificial intelligence and machine learning toward greater integration with statistics, optimization, and other mathematical subjects. For example, the ability of some reinforcement learning methods to learn with parameterized approximators addresses the classical “curse of dimensionality” in operations research and control theory. More distinctively, reinforcement learning has also interacted strongly with psychology and neuroscience, with substantial benefits going both ways. Of all the forms of machine learning, reinforcement learning is the closest to the kind of learning that humans and other animals do, and many of the core algorithms of reinforcement learning were originally inspired by biological learning systems. Reinforcement learning has also given back, both through a psychological model of animal learning that better matches some of the empirical data, and through an influential model of parts of the brain’s reward system. The body of this book develops the ideas of reinforcement learning that pertain to engineering and artificial intelligence, with connections to psychology and neuroscience summarized in Chapters 14 and 15.

Finally, reinforcement learning is also part of a larger trend in artificial intelligence back toward simple general principles. Since the late 1960’s, many artificial intelligence researchers presumed that there are no general principles to be discovered, that intelligence is instead due to the possession of a vast number of special purpose tricks, procedures, and heuristics. It was sometimes said that if we could just get enough relevant facts into a machine, say one million, or one billion, then it would become intelligent. Methods based on general principles, such as search or learning, were characterized as “weak methods,” whereas those based on specific knowledge were called “strong methods.” This view is still common today, but not dominant. From our point of view, it was simply premature: too little effort had been put into the search for general principles to conclude that there were none. Modern artificial intelligence now includes much research looking for general principles of learning, search, and decision making. It is not clear how far back the pendulum will swing, but reinforcement learning research is certainly part of the swing back toward simpler and fewer general principles of artificial intelligence.

1.2 Examples

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

- A master chess player makes a move. The choice is informed both by planning—anticipating possible replies and counterreplies—and by immediate, intuitive judgments of the desirability of particular positions and moves.
- An adaptive controller adjusts parameters of a petroleum refinery’s operation in real time. The controller optimizes the yield/cost/quality trade-off on the basis of specified marginal costs without sticking strictly to the set points originally suggested by engineers.

- A gazelle calf struggles to its feet minutes after being born. Half an hour later it is running at 20 miles per hour.
- A mobile robot decides whether it should enter a new room in search of more trash to collect or start trying to find its way back to its battery recharging station. It makes its decision based on the current charge level of its battery and how quickly and easily it has been able to find the recharger in the past.
- Phil prepares his breakfast. Closely examined, even this apparently mundane activity reveals a complex web of conditional behavior and interlocking goal–subgoal relationships: walking to the cupboard, opening it, selecting a cereal box, then reaching for, grasping, and retrieving the box. Other complex, tuned, interactive sequences of behavior are required to obtain a bowl, spoon, and milk carton. Each step involves a series of eye movements to obtain information and to guide reaching and locomotion. Rapid judgments are continually made about how to carry the objects or whether it is better to ferry some of them to the dining table before obtaining others. Each step is guided by goals, such as grasping a spoon or getting to the refrigerator, and is in service of other goals, such as having the spoon to eat with once the cereal is prepared and ultimately obtaining nourishment. Whether he is aware of it or not, Phil is accessing information about the state of his body that determines his nutritional needs, level of hunger, and food preferences.

These examples share features that are so basic that they are easy to overlook. All involve *interaction* between an active decision-making agent and its environment, within which the agent seeks to achieve a *goal* despite *uncertainty* about its environment. The agent’s actions are permitted to affect the future state of the environment (e.g., the next chess position, the level of reservoirs of the refinery, the robot’s next location and the future charge level of its battery), thereby affecting the actions and opportunities available to the agent at later times. Correct choice requires taking into account indirect, delayed consequences of actions, and thus may require foresight or planning.

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

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

1.3 Elements of Reinforcement Learning

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

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

A *reward signal* defines the goal of a reinforcement learning problem. On each time step, the environment sends to the reinforcement learning agent a single number called the *reward*. The agent's sole objective is to maximize the total reward it receives over the long run. The reward signal thus defines what are the good and bad events for the agent. In a biological system, we might think of rewards as analogous to the experiences of pleasure or pain. They are the immediate and defining features of the problem faced by the agent. The reward signal is the primary basis for altering the policy; if an action selected by the policy is followed by low reward, then the policy may be changed to select some other action in that situation in the future. In general, reward signals may be stochastic functions of the state of the environment and the actions taken.

Whereas the reward signal indicates what is good in an immediate sense, a *value function* specifies what is good in the long run. Roughly speaking, the *value* of a state is the total amount of reward an agent can expect to accumulate over the future, starting from that state. Whereas rewards determine the immediate, intrinsic desirability of environmental states, values indicate the *long-term* desirability of states after taking into account the states that are likely to follow and the rewards available in those states. For example, a state might always yield a low immediate reward but still have a high value because it is regularly followed by other states that yield high rewards. Or the reverse could be true. To make a human analogy, rewards are somewhat like pleasure (if high) and pain (if low), whereas values correspond to a more refined and farsighted judgment of how pleased or displeased we are that our environment is in a particular state.

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

method for efficiently estimating values. The central role of value estimation is arguably the most important thing that has been learned about reinforcement learning over the last six decades.

The fourth and final element of some reinforcement learning systems is a *model* of the environment. This is something that mimics the behavior of the environment, or more generally, that allows inferences to be made about how the environment will behave. For example, given a state and action, the model might predict the resultant next state and next reward. Models are used for *planning*, by which we mean any way of deciding on a course of action by considering possible future situations before they are actually experienced. Methods for solving reinforcement learning problems that use models and planning are called *model-based* methods, as opposed to simpler *model-free* methods that are explicitly trial-and-error learners—viewed as almost the *opposite* of planning. In Chapter 8 we explore reinforcement learning systems that simultaneously learn by trial and error, learn a model of the environment, and use the model for planning. Modern reinforcement learning spans the spectrum from low-level, trial-and-error learning to high-level, deliberative planning.

1.4 Limitations and Scope

Reinforcement learning relies heavily on the concept of state—as input to the policy and value function, and as both input to and output from the model. Informally, we can think of the state as a signal conveying to the agent some sense of “how the environment is” at a particular time. The formal definition of state as we use it here is given by the framework of Markov decision processes presented in Chapter 3. More generally, however, we encourage the reader to follow the informal meaning and think of the state as whatever information is available to the agent about its environment. In effect, we assume that the state signal is produced by some preprocessing system that is nominally part of the agent’s environment. We do not address the issues of constructing, changing, or learning the state signal in this book (other than briefly in Section 17.3). We take this approach not because we consider state representation to be unimportant, but in order to focus fully on the decision-making issues. In other words, our concern in this book is not with designing the state signal, but with deciding what action to take as a function of whatever state signal is available.

Most of the reinforcement learning methods we consider in this book are structured around estimating value functions, but it is not strictly necessary to do this to solve reinforcement learning problems. For example, solution methods such as genetic algorithms, genetic programming, simulated annealing, and other optimization methods never estimate value functions. These methods apply multiple static policies each interacting over an extended period of time with a separate instance of the environment. The policies that obtain the most reward, and random variations of them, are carried over to the next generation of policies, and the process repeats. We call these *evolutionary* methods because their operation is analogous to the way biological evolution produces organisms with skilled behavior even if they do not learn during their individual lifetimes. If the space of policies is sufficiently small, or can be structured so that good policies are

common or easy to find—or if a lot of time is available for the search—then evolutionary methods can be effective. In addition, evolutionary methods have advantages on problems in which the learning agent cannot sense the complete state of its environment.

Our focus is on reinforcement learning methods that learn while interacting with the environment, which evolutionary methods do not do. Methods able to take advantage of the details of individual behavioral interactions can be much more efficient than evolutionary methods in many cases. Evolutionary methods ignore much of the useful structure of the reinforcement learning problem: they do not use the fact that the policy they are searching for is a function from states to actions; they do not notice which states an individual passes through during its lifetime, or which actions it selects. In some cases this information can be misleading (e.g., when states are misperceived), but more often it should enable more efficient search. Although evolution and learning share many features and naturally work together, we do not consider evolutionary methods by themselves to be especially well suited to reinforcement learning problems and, accordingly, we do not cover them in this book.

1.5 An Extended Example: Tic-Tac-Toe

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

Consider the familiar child’s game of tic-tac-toe. Two players take turns playing on a three-by-three board. One player plays Xs and the other Os until one player wins by placing three marks in a row, horizontally, vertically, or diagonally, as the X player has in the game shown to the right. If the board fills up with neither player getting three in a row, then the game is a draw. Because a skilled player can play so as never to lose, let us assume that we are playing against an imperfect player, one whose play is sometimes incorrect and allows us to win. For the moment, in fact, let us consider draws and losses to be equally bad for us. How might we construct a player that will find the imperfections in its opponent’s play and learn to maximize its chances of winning?

X	O	O
O	X	X
		X

Although this is a simple problem, it cannot readily be solved in a satisfactory way through classical techniques. For example, the classical “minimax” solution from game theory is not correct here because it assumes a particular way of playing by the opponent. For example, a minimax player would never reach a game state from which it could lose, even if in fact it always won from that state because of incorrect play by the opponent. Classical optimization methods for sequential decision problems, such as dynamic programming, can *compute* an optimal solution for any opponent, but require as input a complete specification of that opponent, including the probabilities with which the opponent makes each move in each board state. Let us assume that this information is not available *a priori* for this problem, as it is not for the vast majority of problems of practical interest. On the other hand, such information can be estimated from experience, in this case by playing many games against the opponent. About the best one can do

on this problem is first to learn a model of the opponent’s behavior, up to some level of confidence, and then apply dynamic programming to compute an optimal solution given the approximate opponent model. In the end, this is not that different from some of the reinforcement learning methods we examine later in this book.

An evolutionary method applied to this problem would directly search the space of possible policies for one with a high probability of winning against the opponent. Here, a policy is a rule that tells the player what move to make for every state of the game—every possible configuration of Xs and Os on the three-by-three board. For each policy considered, an estimate of its winning probability would be obtained by playing some number of games against the opponent. This evaluation would then direct which policy or policies were considered next. A typical evolutionary method would hill-climb in policy space, successively generating and evaluating policies in an attempt to obtain incremental improvements. Or, perhaps, a genetic-style algorithm could be used that would maintain and evaluate a population of policies. Literally hundreds of different optimization methods could be applied.

Here is how the tic-tac-toe problem would be approached with a method making use of a value function. First we would set up a table of numbers, one for each possible state of the game. Each number will be the latest estimate of the probability of our winning from that state. We treat this estimate as the state’s *value*, and the whole table is the learned value function. State A has higher value than state B, or is considered “better” than state B, if the current estimate of the probability of our winning from A is higher than it is from B. Assuming we always play Xs, then for all states with three Xs in a row the probability of winning is 1, because we have already won. Similarly, for all states with three Os in a row, or that are filled up, the correct probability is 0, as we cannot win from them. We set the initial values of all the other states to 0.5, representing a guess that we have a 50% chance of winning.

We then play many games against the opponent. To select our moves we examine the states that would result from each of our possible moves (one for each blank space on the board) and look up their current values in the table. Most of the time we move *greedily*, selecting the move that leads to the state with greatest value, that is, with the highest estimated probability of winning. Occasionally, however, we select randomly from among the other moves instead. These are called *exploratory* moves because they cause us to experience states that we might otherwise never see. A sequence of moves made and considered during a game can be diagrammed as in Figure 1.1.

While we are playing, we change the values of the states in which we find ourselves during the game. We attempt to make them more accurate estimates of the probabilities of winning. To do this, we “back up” the value of the state after each greedy move to the state before the move, as suggested by the arrows in Figure 1.1. More precisely, the current value of the earlier state is updated to be closer to the value of the later state. This can be done by moving the earlier state’s value a fraction of the way toward the value of the later state. If we let S_t denote the state before the greedy move, and S_{t+1} the state after the move, then the update to the estimated value of S_t , denoted $V(S_t)$, can be written as

$$V(S_t) \leftarrow V(S_t) + \alpha [V(S_{t+1}) - V(S_t)],$$



Figure 1.1: A sequence of tic-tac-toe moves. The solid black lines represent the moves taken during a game; the dashed lines represent moves that we (our reinforcement learning player) considered but did not make. Our second move was an exploratory move, meaning that it was taken even though another sibling move, the one leading to e^* , was ranked higher. Exploratory moves do not result in any learning, but each of our other moves does, causing updates as suggested by the red arrows in which estimated values are moved up the tree from later nodes to earlier nodes as detailed in the text.

where α is a small positive fraction called the *step-size parameter*, which influences the rate of learning. This update rule is an example of a *temporal-difference* learning method, so called because its changes are based on a difference, $V(S_{t+1}) - V(S_t)$, between estimates at two successive times.

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

This example illustrates the differences between evolutionary methods and methods that learn value functions. To evaluate a policy an evolutionary method holds the policy fixed and plays many games against the opponent, or simulates many games using a model of the opponent. The frequency of wins gives an unbiased estimate of the probability

of winning with that policy, and can be used to direct the next policy selection. But each policy change is made only after many games, and only the final outcome of each game is used: what happens *during* the games is ignored. For example, if the player wins, then *all* of its behavior in the game is given credit, independently of how specific moves might have been critical to the win. Credit is even given to moves that never occurred! Value function methods, in contrast, allow individual states to be evaluated. In the end, evolutionary and value function methods both search the space of policies, but learning a value function takes advantage of information available during the course of play.

This simple example illustrates some of the key features of reinforcement learning methods. First, there is the emphasis on learning while interacting with an environment, in this case with an opponent player. Second, there is a clear goal, and correct behavior requires planning or foresight that takes into account delayed effects of one's choices. For example, the simple reinforcement learning player would learn to set up multi-move traps for a shortsighted opponent. It is a striking feature of the reinforcement learning solution that it can achieve the effects of planning and lookahead without using a model of the opponent and without conducting an explicit search over possible sequences of future states and actions.

While this example illustrates some of the key features of reinforcement learning, it is so simple that it might give the impression that reinforcement learning is more limited than it really is. Although tic-tac-toe is a two-person game, reinforcement learning also applies in the case in which there is no external adversary, that is, in the case of a “game against nature.” Reinforcement learning also is not restricted to problems in which behavior breaks down into separate episodes, like the separate games of tic-tac-toe, with reward only at the end of each episode. It is just as applicable when behavior continues indefinitely and when rewards of various magnitudes can be received at any time. Reinforcement learning is also applicable to problems that do not even break down into discrete time steps like the plays of tic-tac-toe. The general principles apply to continuous-time problems as well, although the theory gets more complicated and we omit it from this introductory treatment.

Tic-tac-toe has a relatively small, finite state set, whereas reinforcement learning can be used when the state set is very large, or even infinite. For example, Gerry Tesauro (1992, 1995) combined the algorithm described above with an artificial neural network to learn to play backgammon, which has approximately 10^{20} states. With this many states it is impossible ever to experience more than a small fraction of them. Tesauro's program learned to play far better than any previous program and eventually better than the world's best human players (Section 16.1). The artificial neural network provides the program with the ability to generalize from its experience, so that in new states it selects moves based on information saved from similar states faced in the past, as determined by its network. How well a reinforcement learning system can work in problems with such large state sets is intimately tied to how appropriately it can generalize from past experience. It is in this role that we have the greatest need for supervised learning methods with reinforcement learning. Artificial neural networks and deep learning (Section 9.6) are not the only, or necessarily the best, way to do this.

In this tic-tac-toe example, learning started with no prior knowledge beyond the

rules of the game, but reinforcement learning by no means entails a tabula rasa view of learning and intelligence. On the contrary, prior information can be incorporated into reinforcement learning in a variety of ways that can be critical for efficient learning (e.g., see Sections 9.5, 17.4, and 13.1). We also have access to the true state in the tic-tac-toe example, whereas reinforcement learning can also be applied when part of the state is hidden, or when different states appear to the learner to be the same.

Finally, the tic-tac-toe player was able to look ahead and know the states that would result from each of its possible moves. To do this, it had to have a model of the game that allowed it to foresee how its environment would change in response to moves that it might never make. Many problems are like this, but in others even a short-term model of the effects of actions is lacking. Reinforcement learning can be applied in either case. A model is not required, but models can easily be used if they are available or can be learned (Chapter 8).

On the other hand, there are reinforcement learning methods that do not need any kind of environment model at all. Model-free systems cannot even think about how their environments will change in response to a single action. The tic-tac-toe player is model-free in this sense with respect to its opponent: it has no model of its opponent of any kind. Because models have to be reasonably accurate to be useful, model-free methods can have advantages over more complex methods when the real bottleneck in solving a problem is the difficulty of constructing a sufficiently accurate environment model. Model-free methods are also important building blocks for model-based methods. In this book we devote several chapters to model-free methods before we discuss how they can be used as components of more complex model-based methods.

Reinforcement learning can be used at both high and low levels in a system. Although the tic-tac-toe player learned only about the basic moves of the game, nothing prevents reinforcement learning from working at higher levels where each of the “actions” may itself be the application of a possibly elaborate problem-solving method. In hierarchical learning systems, reinforcement learning can work simultaneously on several levels.

Exercise 1.1: Self-Play Suppose, instead of playing against a random opponent, the reinforcement learning algorithm described above played against itself, with both sides learning. What do you think would happen in this case? Would it learn a different policy for selecting moves? □

Exercise 1.2: Symmetries Many tic-tac-toe positions appear different but are really the same because of symmetries. How might we amend the learning process described above to take advantage of this? In what ways would this change improve the learning process? Now think again. Suppose the opponent did not take advantage of symmetries. In that case, should we? Is it true, then, that symmetrically equivalent positions should necessarily have the same value? □

Exercise 1.3: Greedy Play Suppose the reinforcement learning player was *greedy*, that is, it always played the move that brought it to the position that it rated the best. Might it learn to play better, or worse, than a nongreedy player? What problems might occur? □

Exercise 1.4: Learning from Exploration Suppose learning updates occurred after *all* moves, including exploratory moves. If the step-size parameter is appropriately reduced

over time (but not the tendency to explore), then the state values would converge to a different set of probabilities. What (conceptually) are the two sets of probabilities computed when we do, and when we do not, learn from exploratory moves? Assuming that we do continue to make exploratory moves, which set of probabilities might be better to learn? Which would result in more wins? \square

Exercise 1.5: Other Improvements Can you think of other ways to improve the reinforcement learning player? Can you think of any better way to solve the tic-tac-toe problem as posed? \square

1.6 Summary

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

Reinforcement learning uses the formal framework of Markov decision processes to define the interaction between a learning agent and its environment in terms of states, actions, and rewards. This framework is intended to be a simple way of representing essential features of the artificial intelligence problem. These features include a sense of cause and effect, a sense of uncertainty and nondeterminism, and the existence of explicit goals.

The concepts of value and value function are key to most of the reinforcement learning methods that we consider in this book. We take the position that value functions are important for efficient search in the space of policies. The use of value functions distinguishes reinforcement learning methods from evolutionary methods that search directly in policy space guided by evaluations of entire policies.

1.7 Early History of Reinforcement Learning

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

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

The term “optimal control” came into use in the late 1950s to describe the problem of designing a controller to minimize or maximize a measure of a dynamical system’s behavior over time. One of the approaches to this problem was developed in the mid-1950s by Richard Bellman and others through extending a nineteenth century theory of Hamilton and Jacobi. This approach uses the concepts of a dynamical system’s state and of a value function, or “optimal return function,” to define a functional equation, now often called the Bellman equation. The class of methods for solving optimal control problems by solving this equation came to be known as dynamic programming (Bellman, 1957a). Bellman (1957b) also introduced the discrete stochastic version of the optimal control problem known as Markov decision processes (MDPs). Ronald Howard (1960) devised the policy iteration method for MDPs. All of these are essential elements underlying the theory and algorithms of modern reinforcement learning.

Dynamic programming is widely considered the only feasible way of solving general stochastic optimal control problems. It suffers from what Bellman called “the curse of dimensionality,” meaning that its computational requirements grow exponentially with the number of state variables, but it is still far more efficient and more widely applicable than any other general method. Dynamic programming has been extensively developed since the late 1950s, including extensions to partially observable MDPs (surveyed by Lovejoy, 1991), many applications (surveyed by White, 1985, 1988, 1993), approximation methods (surveyed by Rust, 1996), and asynchronous methods (Bertsekas, 1982, 1983). Many excellent modern treatments of dynamic programming are available (e.g., Bertsekas, 2005, 2012; Puterman, 1994; Ross, 1983; and Whittle, 1982, 1983). Bryson (1996) provides an authoritative history of optimal control.

Connections between optimal control and dynamic programming, on the one hand, and learning, on the other, were slow to be recognized. We cannot be sure about what accounted for this separation, but its main cause was likely the separation between the disciplines involved and their different goals. Also contributing may have been the prevalent view of dynamic programming as an offline computation depending essentially on accurate system models and analytic solutions to the Bellman equation. Further, the simplest form of dynamic programming is a computation that proceeds backwards in time, making it difficult to see how it could be involved in a learning process that must proceed in a forward direction. Some of the earliest work in dynamic programming, such as that by Bellman and Dreyfus (1959), might now be classified as following a learning approach. Witten’s (1977) work (discussed below) certainly qualifies as a combination of learning and dynamic-programming ideas. Werbos (1987) argued explicitly for greater interrelation of dynamic programming and learning methods and for dynamic programming’s relevance to understanding neural and cognitive mechanisms. For us the full integration of dynamic programming methods with online learning did not occur until the work of Chris Watkins in 1989, whose treatment of reinforcement learning using the MDP formalism has been widely adopted. Since then these relationships have been extensively developed by many researchers, most particularly by Dimitri Bertsekas

and John Tsitsiklis (1996), who coined the term “neurodynamic programming” to refer to the combination of dynamic programming and artificial neural networks. Another term currently in use is “approximate dynamic programming.” These various approaches emphasize different aspects of the subject, but they all share with reinforcement learning an interest in circumventing the classical shortcomings of dynamic programming.

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

Let us return now to the other major thread leading to the modern field of reinforcement learning, the thread centered on the idea of trial-and-error learning. We only touch on the major points of contact here, taking up this topic in more detail in Section 14.3. According to American psychologist R. S. Woodworth (1938) the idea of trial-and-error learning goes as far back as the 1850s to Alexander Bain’s discussion of learning by “groping and experiment” and more explicitly to the British ethologist and psychologist Conway Lloyd Morgan’s 1894 use of the term to describe his observations of animal behavior. Perhaps the first to succinctly express the essence of trial-and-error learning as a principle of learning was Edward Thorndike:

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

Thorndike called this the “Law of Effect” because it describes the effect of reinforcing events on the tendency to select actions. Thorndike later modified the law to better account for subsequent data on animal learning (such as differences between the effects of reward and punishment), and the law in its various forms has generated considerable controversy among learning theorists (e.g., see Gallistel, 2005; Herrnstein, 1970; Kimble, 1961, 1967; Mazur, 1994). Despite this, the Law of Effect—in one form or another—is widely regarded as a basic principle underlying much behavior (e.g., Hilgard and Bower, 1975; Dennett, 1978; Campbell, 1960; Cziko, 1995). It is the basis of the influential

learning theories of Clark Hull (1943, 1952) and the influential experimental methods of B. F. Skinner (1938).

The term “reinforcement” in the context of animal learning came into use well after Thorndike’s expression of the Law of Effect, first appearing in this context (to the best of our knowledge) in the 1927 English translation of Pavlov’s monograph on conditioned reflexes. Pavlov described reinforcement as the strengthening of a pattern of behavior due to an animal receiving a stimulus—a reinforcer—in an appropriate temporal relationship with another stimulus or with a response. Some psychologists extended the idea of reinforcement to include weakening as well as strengthening of behavior, and extended the idea of a reinforcer to include possibly the omission or termination of stimulus. To be considered reinforcer, the strengthening or weakening must persist after the reinforcer is withdrawn; a stimulus that merely attracts an animal’s attention or that energizes its behavior without producing lasting changes would not be considered a reinforcer.

The idea of implementing trial-and-error learning in a computer appeared among the earliest thoughts about the possibility of artificial intelligence. In a 1948 report, Alan Turing described a design for a “pleasure-pain system” that worked along the lines of the Law of Effect:

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)

Many ingenious electro-mechanical machines were constructed that demonstrated trial-and-error learning. The earliest may have been a machine built by Thomas Ross (1933) that was able to find its way through a simple maze and remember the path through the settings of switches. In 1951 W. Grey Walter built a version of his “mechanical tortoise” (Walter, 1950) capable of a simple form of learning. In 1952 Claude Shannon demonstrated a maze-running mouse named Theseus that used trial and error to find its way through a maze, with the maze itself remembering the successful directions via magnets and relays under its floor (see also Shannon, 1951). J. A. Deutsch (1954) described a maze-solving machine based on his behavior theory (Deutsch, 1953) that has some properties in common with model-based reinforcement learning (Chapter 8). In his Ph.D. dissertation, Marvin Minsky (1954) discussed computational models of reinforcement learning and described his construction of an analog machine composed of components he called SNARCs (Stochastic Neural-Analog Reinforcement Calculators) meant to resemble modifiable synaptic connections in the brain (Chapter 15). The web site cyberneticzoo.com contains a wealth of information on these and many other electro-mechanical learning machines.

Building electro-mechanical learning machines gave way to programming digital computers to perform various types of learning, some of which implemented trial-and-error learning. Farley and Clark (1954) described a digital simulation of a neural-network learning machine that learned by trial and error. But their interests soon shifted from trial-and-error learning to generalization and pattern recognition, that is, from reinforcement learning to supervised learning (Clark and Farley, 1955). This began a pattern

of confusion about the relationship between these types of learning. Many researchers seemed to believe that they were studying reinforcement learning when they were actually studying supervised learning. For example, artificial neural network pioneers such as Rosenblatt (1962) and Widrow and Hoff (1960) were clearly motivated by reinforcement learning—they used the language of rewards and punishments—but the systems they studied were supervised learning systems suitable for pattern recognition and perceptual learning. Even today, some researchers and textbooks minimize or blur the distinction between these types of learning. For example, some artificial neural network textbooks have used the term “trial-and-error” to describe networks that learn from training examples. This is an understandable confusion because these networks use error information to update connection weights, but this misses the essential character of trial-and-error learning as selecting actions on the basis of evaluative feedback that does not rely on knowledge of what the correct action should be.

Partly as a result of these confusions, research into genuine trial-and-error learning became rare in the 1960s and 1970s, although there were notable exceptions. In the 1960s the terms “reinforcement” and “reinforcement learning” were used in the engineering literature for the first time to describe engineering uses of trial-and-error learning (e.g., Waltz and Fu, 1965; Mendel, 1966; Fu, 1970; Mendel and McLaren, 1970). Particularly influential was Minsky’s paper “Steps Toward Artificial Intelligence” (Minsky, 1961), which discussed several issues relevant to trial-and-error learning, including prediction, expectation, and what he called the *basic credit-assignment problem for complex reinforcement learning systems*: How do you distribute credit for success among the many decisions that may have been involved in producing it? All of the methods we discuss in this book are, in a sense, directed toward solving this problem. Minsky’s paper is well worth reading today.

In the next few paragraphs we discuss some of the other exceptions and partial exceptions to the relative neglect of computational and theoretical study of genuine trial-and-error learning in the 1960s and 1970s.

One exception was the work of the New Zealand researcher John Andreae, who developed a system called STeLLA that learned by trial and error in interaction with its environment. This system included an internal model of the world and, later, an “internal monologue” to deal with problems of hidden state (Andreae, 1963, 1969a,b). Andreae’s later work (1977) placed more emphasis on learning from a teacher, but still included learning by trial and error, with the generation of novel events being one of the system’s goals. A feature of this work was a “leakback process,” elaborated more fully in Andreae (1998), that implemented a credit-assignment mechanism similar to the backing-up update operations that we describe. Unfortunately, his pioneering research was not well known and did not greatly impact subsequent reinforcement learning research. Recent summaries are available (Andreae, 2017a,b).

More influential was the work of Donald Michie. In 1961 and 1963 he described a simple trial-and-error learning system for learning how to play tic-tac-toe (or naughts and crosses) called MENACE (for Matchbox Educable Naughts and Crosses Engine). It consisted of a matchbox for each possible game position, each matchbox containing a number of colored beads, a different color for each possible move from that position. By

drawing a bead at random from the matchbox corresponding to the current game position, one could determine MENACE’s move. When a game was over, beads were added to or removed from the boxes used during play to reward or punish MENACE’s decisions. Michie and Chambers (1968) described another tic-tac-toe reinforcement learner called GLEE (Game Learning Expectimaxing Engine) and a reinforcement learning controller called BOXES. They applied BOXES to the task of learning to balance a pole hinged to a movable cart on the basis of a failure signal occurring only when the pole fell or the cart reached the end of a track. This task was adapted from the earlier work of Widrow and Smith (1964), who used supervised learning methods, assuming instruction from a teacher already able to balance the pole. Michie and Chambers’s version of pole-balancing is one of the best early examples of a reinforcement learning task under conditions of incomplete knowledge. It influenced much later work in reinforcement learning, beginning with some of our own studies (Barto, Sutton, and Anderson, 1983; Sutton, 1984). Michie consistently emphasized the role of trial and error and learning as essential aspects of artificial intelligence (Michie, 1974).

Widrow, Gupta, and Maitra (1973) modified the Least-Mean-Square (LMS) algorithm of Widrow and Hoff (1960) to produce a reinforcement learning rule that could learn from success and failure signals instead of from training examples. They called this form of learning “selective bootstrap adaptation” and described it as “learning with a critic” instead of “learning with a teacher.” They analyzed this rule and showed how it could learn to play blackjack. This was an isolated foray into reinforcement learning by Widrow, whose contributions to supervised learning were much more influential. Our use of the term “critic” is derived from Widrow, Gupta, and Maitra’s paper. Buchanan, Mitchell, Smith, and Johnson (1978) independently used the term critic in the context of machine learning (see also Dietterich and Buchanan, 1984), but for them a critic is an expert system able to do more than evaluate performance.

Research on *learning automata* had a more direct influence on the trial-and-error thread leading to modern reinforcement learning research. These are methods for solving a nonassociative, purely selectional learning problem known as the *k*-armed bandit by analogy to a slot machine, or “one-armed bandit,” except with *k* levers (see Chapter 2). Learning automata are simple, low-memory machines for improving the probability of reward in these problems. Learning automata originated with work in the 1960s of the Russian mathematician and physicist M. L. Tsetlin and colleagues (published posthumously in Tsetlin, 1973) and has been extensively developed since then within engineering (see Narendra and Thathachar, 1974, 1989). These developments included the study of *stochastic learning automata*, which are methods for updating action probabilities on the basis of reward signals. Although not developed in the tradition of stochastic learning automata, Harth and Tzanakou’s (1974) Alopex algorithm (for *Algorithm of pattern extraction*) is a stochastic method for detecting correlations between actions and reinforcement that influenced some of our early research (Barto, Sutton, and Brouwer, 1981). Stochastic learning automata were foreshadowed by earlier work in psychology, beginning with William Estes’ (1950) effort toward a statistical theory of learning and further developed by others (e.g., Bush and Mosteller, 1955; Sternberg, 1963).

The statistical learning theories developed in psychology were adopted by researchers in

economics, leading to a thread of research in that field devoted to reinforcement learning. This work began in 1973 with the application of Bush and Mosteller’s learning theory to a collection of classical economic models (Cross, 1973). One goal of this research was to study artificial agents that act more like real people than do traditional idealized economic agents (Arthur, 1991). This approach expanded to the study of reinforcement learning in the context of game theory. Reinforcement learning in economics developed largely independently of the early work in reinforcement learning in artificial intelligence, though game theory remains a topic of interest in both fields (beyond the scope of this book). Camerer (2011) discusses the reinforcement learning tradition in economics, and Nowé, Vrancx, and De Hauwere (2012) provide an overview of the subject from the point of view of multi-agent extensions to the approach that we introduce in this book. Reinforcement in the context of game theory is a much different subject than reinforcement learning used in programs to play tic-tac-toe, checkers, and other recreational games. See, for example, Szita (2012) for an overview of this aspect of reinforcement learning and games.

John Holland (1975) outlined a general theory of adaptive systems based on selectional principles. His early work concerned trial and error primarily in its nonassociative form, as in evolutionary methods and the *k*-armed bandit. In 1976 and more fully in 1986, he introduced *classifier systems*, true reinforcement learning systems including association and value functions. A key component of Holland’s classifier systems was the “bucket-brigade algorithm” for credit assignment, which is closely related to the temporal difference algorithm used in our tic-tac-toe example and discussed in Chapter 6. Another key component was a *genetic algorithm*, an evolutionary method whose role was to evolve useful representations. Classifier systems have been extensively developed by many researchers to form a major branch of reinforcement learning research (reviewed by Urbanowicz and Moore, 2009), but genetic algorithms—which we do not consider to be reinforcement learning systems by themselves—have received much more attention, as have other approaches to evolutionary computation (e.g., Fogel, Owens and Walsh, 1966, and Koza, 1992).

The individual most responsible for reviving the trial-and-error thread to reinforcement learning within artificial intelligence was Harry Klopf (1972, 1975, 1982). Klopf recognized that essential aspects of adaptive behavior were being lost as learning researchers came to focus almost exclusively on supervised learning. What was missing, according to Klopf, were the hedonic aspects of behavior, the drive to achieve some result from the environment, to control the environment toward desired ends and away from undesired ends (see Section 15.9). This is the essential idea of trial-and-error learning. Klopf’s ideas were especially influential on the authors because our assessment of them (Barto and Sutton, 1981a) led to our appreciation of the distinction between supervised and reinforcement learning, and to our eventual focus on reinforcement learning. Much of the early work that we and colleagues accomplished was directed toward showing that reinforcement learning and supervised learning were indeed different (Barto, Sutton, and Brouwer, 1981; Barto and Sutton, 1981b; Barto and Anandan, 1985). Other studies showed how reinforcement learning could address important problems in artificial neural network learning, in particular, how it could produce learning algorithms for multilayer networks (Barto, Anderson, and Sutton, 1982; Barto and Anderson, 1985; Barto, 1985, 1986; Barto and Jordan, 1987; see Section 15.10).

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

The origins of temporal-difference learning are in part in animal learning psychology, in particular, in the notion of *secondary reinforcers*. A secondary reinforcer is a stimulus that has been paired with a primary reinforcer such as food or pain and, as a result, has come to take on similar reinforcing properties. Minsky (1954) may have been the first to realize that this psychological principle could be important for artificial learning systems. Arthur Samuel (1959) was the first to propose and implement a learning method that included temporal-difference ideas, as part of his celebrated checkers-playing program (Section 16.2).

Samuel made no reference to Minsky's work or to possible connections to animal learning. His inspiration apparently came from Claude Shannon's (1950) suggestion that a computer could be programmed to use an evaluation function to play chess, and that it might be able to improve its play by modifying this function online. (It is possible that these ideas of Shannon's also influenced Bellman, but we know of no evidence for this.) Minsky (1961) extensively discussed Samuel's work in his "Steps" paper, suggesting the connection to secondary reinforcement theories, both natural and artificial.

As we have discussed, in the decade following the work of Minsky and Samuel, little computational work was done on trial-and-error learning, and apparently no computational work at all was done on temporal-difference learning. In 1972, Klopf brought trial-and-error learning together with an important component of temporal-difference learning. Klopf was interested in principles that would scale to learning in large systems, and thus was intrigued by notions of local reinforcement, whereby subcomponents of an overall learning system could reinforce one another. He developed the idea of "generalized reinforcement," whereby every component (nominally, every neuron) views all of its inputs in reinforcement terms: excitatory inputs as rewards and inhibitory inputs as punishments. This is not the same idea as what we now know as temporal-difference learning, and in retrospect it is farther from it than was Samuel's work. On the other hand, Klopf linked the idea with trial-and-error learning and related it to the massive empirical database of animal learning psychology.

Sutton (1978a,b,c) developed Klopf's ideas further, particularly the links to animal learning theories, describing learning rules driven by changes in temporally successive predictions. He and Barto refined these ideas and developed a psychological model of classical conditioning based on temporal-difference learning (Sutton and Barto, 1981a; Barto and Sutton, 1982). There followed several other influential psychological models of classical conditioning based on temporal-difference learning (e.g., Klopf, 1988; Moore et al., 1986; Sutton and Barto, 1987, 1990). Some neuroscience models developed at this time are well interpreted in terms of temporal-difference learning (Hawkins and Kandel, 1984; Byrne, Gingrich, and Baxter, 1990; Gelperin, Hopfield, and Tank, 1985; Tesauro,

1986; Friston et al., 1994), although in most cases there was no historical connection.

Our early work on temporal-difference learning was strongly influenced by animal learning theories and by Klopf’s work. Relationships to Minsky’s “Steps” paper and to Samuel’s checkers players were recognized only afterward. By 1981, however, we were fully aware of all the prior work mentioned above as part of the temporal-difference and trial-and-error threads. At this time we developed a method for using temporal-difference learning combined with trial-and-error learning, known as the *actor–critic architecture*, and applied this method to Michie and Chambers’s pole-balancing problem (Barto, Sutton, and Anderson, 1983). This method was extensively studied in Sutton’s (1984) Ph.D. dissertation and extended to use backpropagation neural networks in Anderson’s (1986) Ph.D. dissertation. Around this time, Holland (1986) incorporated temporal-difference ideas explicitly into his classifier systems in the form of his bucket-brigade algorithm. A key step was taken by Sutton (1988) by separating temporal-difference learning from control, treating it as a general prediction method. That paper also introduced the $\text{TD}(\lambda)$ algorithm and proved some of its convergence properties.

As we were finalizing our work on the actor–critic architecture in 1981, we discovered a paper by Ian Witten (1977, 1976a) which appears to be the earliest publication of a temporal-difference learning rule. He proposed the method that we now call tabular $\text{TD}(0)$ for use as part of an adaptive controller for solving MDPs. This work was first submitted for journal publication in 1974 and also appeared in Witten’s 1976 PhD dissertation. Witten’s work was a descendant of Andreae’s early experiments with STELLA and other trial-and-error learning systems. Thus, Witten’s 1977 paper spanned both major threads of reinforcement learning research—trial-and-error learning and optimal control—while making a distinct early contribution to temporal-difference learning.

The temporal-difference and optimal control threads were fully brought together in 1989 with Chris Watkins’s development of Q-learning. This work extended and integrated prior work in all three threads of reinforcement learning research. Paul Werbos (1987) contributed to this integration by arguing for the convergence of trial-and-error learning and dynamic programming since 1977. By the time of Watkins’s work there had been tremendous growth in reinforcement learning research, primarily in the machine learning subfield of artificial intelligence, but also in artificial neural networks and artificial intelligence more broadly. In 1992, the remarkable success of Gerry Tesauro’s backgammon playing program, TD-Gammon, brought additional attention to the field.

In the time since publication of the first edition of this book, a flourishing subfield of neuroscience developed that focuses on the relationship between reinforcement learning algorithms and reinforcement learning in the nervous system. Most responsible for this is an uncanny similarity between the behavior of temporal-difference algorithms and the activity of dopamine producing neurons in the brain, as pointed out by a number of researchers (Friston et al., 1994; Barto, 1995a; Houk, Adams, and Barto, 1995; Montague, Dayan, and Sejnowski, 1996; and Schultz, Dayan, and Montague, 1997). Chapter 15 provides an introduction to this exciting aspect of reinforcement learning. Other important contributions made in the recent history of reinforcement learning are too numerous to mention in this brief account; we cite many of these at the end of the individual chapters in which they arise.

Bibliographical Remarks

For additional general coverage of reinforcement learning, we refer the reader to the books by Szepesvári (2010), Bertsekas and Tsitsiklis (1996), Kaelbling (1993a), and Sugiyama, Hachiya, and Morimura (2013). Books that take a control or operations research perspective include those of Si, Barto, Powell, and Wunsch (2004), Powell (2011), Lewis and Liu (2012), and Bertsekas (2012). Cao's (2009) review places reinforcement learning in the context of other approaches to learning and optimization of stochastic dynamic systems. Three special issues of the journal *Machine Learning* focus on reinforcement learning: Sutton (1992a), Kaelbling (1996), and Singh (2002). Useful surveys are provided by Barto (1995b); Kaelbling, Littman, and Moore (1996); and Keerthi and Ravindran (1997). The volume edited by Weiring and van Otterlo (2012) provides an excellent overview of recent developments.

- 1.2** The example of Phil's breakfast in this chapter was inspired by Agre (1988).
- 1.5** The temporal-difference method used in the tic-tac-toe example is developed in Chapter 6.

Part III: Looking Deeper

In this last part of the book we look beyond the standard reinforcement learning ideas presented in the first two parts of the book to briefly survey their relationships with psychology and neuroscience, a sampling of reinforcement learning applications, and some of the active frontiers for future reinforcement learning research.

Chapter 14

Psychology

In previous chapters we developed ideas for algorithms based on computational considerations alone. In this chapter we look at some of these algorithms from another perspective: the perspective of psychology and its study of how animals learn. The goals of this chapter are, first, to discuss ways that reinforcement learning ideas and algorithms correspond to what psychologists have discovered about animal learning, and second, to explain the influence reinforcement learning is having on the study of animal learning. The clear formalism provided by reinforcement learning that systemizes tasks, returns, and algorithms is proving to be enormously useful in making sense of experimental data, in suggesting new kinds of experiments, and in pointing to factors that may be critical to manipulate and to measure. The idea of optimizing return over the long term that is at the core of reinforcement learning is contributing to our understanding of otherwise puzzling features of animal learning and behavior.

Some of the correspondences between reinforcement learning and psychological theories are not surprising because the development of reinforcement learning drew inspiration from psychological learning theories. However, as developed in this book, reinforcement learning explores idealized situations from the perspective of an artificial intelligence researcher or engineer, with the goal of solving computational problems with efficient algorithms, rather than to replicate or explain in detail how animals learn. As a result, some of the correspondences we describe connect ideas that arose independently in their respective fields. We believe these points of contact are specially meaningful because they expose computational principles important to learning, whether it is learning by artificial or by natural systems.

For the most part, we describe correspondences between reinforcement learning and learning theories developed to explain how animals like rats, pigeons, and rabbits learn in controlled laboratory experiments. Thousands of these experiments were conducted throughout the 20th century, and many are still being conducted today. Although sometimes dismissed as irrelevant to wider issues in psychology, these experiments probe subtle properties of animal learning, often motivated by precise theoretical questions. As psychology shifted its focus to more cognitive aspects of behavior, that is, to mental processes such as thought and reasoning, animal learning experiments came to play

less of a role in psychology than they once did. But this experimentation led to the discovery of learning principles that are elemental and widespread throughout the animal kingdom, principles that should not be neglected in designing artificial learning systems. In addition, as we shall see, some aspects of cognitive processing connect naturally to the computational perspective provided by reinforcement learning.

This chapter's final section includes references relevant to the connections we discuss as well as to connections we neglect. We hope this chapter encourages readers to probe all of these connections more deeply. Also included in this final section is a discussion of how the terminology used in reinforcement learning relates to that of psychology. Many of the terms and phrases used in reinforcement learning are borrowed from animal learning theories, but the computational/engineering meanings of these terms and phrases do not always coincide with their meanings in psychology.

14.1 Prediction and Control

The algorithms we describe in this book fall into two broad categories: algorithms for *prediction* and algorithms for *control*.¹ These categories arise naturally in solution methods for the reinforcement learning problem presented in Chapter 3. In many ways these categories respectively correspond to categories of learning extensively studied by psychologists: *classical*, or *Pavlovian conditioning* and *instrumental*, or *operant conditioning*. These correspondences are not completely accidental because of psychology's influence on reinforcement learning, but they are nevertheless striking because they connect ideas arising from different objectives.

The prediction algorithms presented in this book estimate quantities that depend on how features of an agent's environment are expected to unfold over the future. We specifically focus on estimating the amount of reward an agent can expect to receive over the future while it interacts with its environment. In this role, prediction algorithms are *policy evaluation algorithms*, which are integral components of algorithms for improving policies. But prediction algorithms are not limited to predicting future reward; they can predict any feature of the environment (see, for example, Modayil, White, and Sutton, 2014). The correspondence between prediction algorithms and classical conditioning rests on their common property of predicting upcoming stimuli, whether or not those stimuli are rewarding (or punishing).

The situation in an instrumental, or operant, conditioning experiment is different. Here, the experimental apparatus is set up so that an animal is given something it likes (a reward) or something it dislikes (a penalty) depending on what the animal did. The animal learns to increase its tendency to produce rewarded behavior and to decrease its tendency to produce penalized behavior. The reinforcing stimulus is said to be *contingent* on the animal's behavior, whereas in classical conditioning it is not (although it is difficult to remove all behavior contingencies in a classical conditioning experiment). Instrumental conditioning experiments are like those that inspired Thorndike's Law of Effect that

¹What control means for us is different from what it typically means in animal learning theories; there the environment controls the agent instead of the other way around. See our comments on terminology at the end of this chapter.

we briefly discuss in Chapter 1. *Control* is at the core of this form of learning, which corresponds to the operation of reinforcement learning’s policy-improvement algorithms.

Thinking of classical conditioning in terms of prediction, and instrumental conditioning in terms of control, is a starting point for connecting our computational view of reinforcement learning to animal learning, but in reality, the situation is more complicated than this. There is more to classical conditioning than prediction; it also involves action, and so is a mode of control, sometimes called *Pavlovian control*. Further, classical and instrumental conditioning interact in interesting ways, with both sorts of learning likely being engaged in most experimental situations. Despite these complications, aligning the classical/instrumental distinction with the prediction/control distinction is a convenient first approximation in connecting reinforcement learning to animal learning.

In psychology, the term reinforcement is used to describe learning in both classical and instrumental conditioning. Originally referring only to the strengthening of a pattern of behavior, it is frequently also used for the weakening of a pattern of behavior. A stimulus considered to be the cause of the change in behavior is called a reinforcer, whether or not it is contingent on the animal’s previous behavior. At the end of this chapter we discuss this terminology in more detail and how it relates to terminology used in machine learning.

14.2 Classical Conditioning

While studying the activity of the digestive system, the celebrated Russian physiologist Ivan Pavlov found that an animal’s innate responses to certain triggering stimuli can come to be triggered by other stimuli that are quite unrelated to the inborn triggers. His experimental subjects were dogs that had undergone minor surgery to allow the intensity of their salivary reflex to be accurately measured. In one case he describes, the dog did not salivate under most circumstances, but about 5 seconds after being presented with food it produced about six drops of saliva over the next several seconds. After several repetitions of presenting another stimulus, one not related to food, in this case the sound of a metronome, shortly before the introduction of food, the dog salivated in response to the sound of the metronome in the same way it did to the food. “The activity of the salivary gland has thus been called into play by impulses of sound—a stimulus quite alien to food” (Pavlov, 1927, p. 22). Summarizing the significance of this finding, Pavlov wrote:

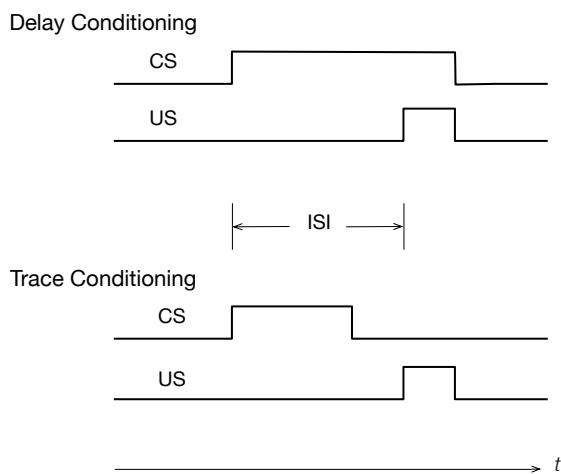
It is pretty evident that under natural conditions the normal animal must respond not only to stimuli which themselves bring immediate benefit or harm, but also to other physical or chemical agencies—waves of sound, light, and the like—which in themselves only *signal* the approach of these stimuli; though it is not the sight and sound of the beast of prey which is in itself harmful to the smaller animal, but its teeth and claws. (Pavlov, 1927, p. 14)

Connecting new stimuli to innate reflexes in this way is now called classical, or Pavlovian, conditioning. Pavlov (or more exactly, his translators) called inborn responses (e.g., salivation in his demonstration described above) “unconditioned responses” (URs), their

natural triggering stimuli (e.g., food) “unconditioned stimuli” (USs), and new responses triggered by predictive stimuli (e.g., here also salivation) “conditioned responses” (CRs). A stimulus that is initially neutral, meaning that it does not normally elicit strong responses (e.g., the metronome sound), becomes a “conditioned stimulus” (CS) as the animal learns that it predicts the US and so comes to produce a CR in response to the CS. These terms are still used in describing classical conditioning experiments (though better translations would have been “conditional” and “unconditional” instead of conditioned and unconditioned). The US is called a reinforcer because it reinforces producing a CR in response to the CS.

The arrangement of stimuli in two common types of classical conditioning experiments is shown to the right. In *delay conditioning*, the CS extends throughout the interstimulus interval, or ISI, which is the time interval between the CS onset and the US onset (with the CS ending when the US ends in a common version shown here). In *trace conditioning*, the US begins after the CS ends, and the time interval between CS offset and US onset is called the trace interval.

The salivation of Pavlov’s dogs to the sound of a metronome is just one example of classical conditioning, which has been intensively studied across many response systems of many species of animals. URs are often preparatory in some way, like the salivation of Pavlov’s dog, or protective in some way, like an eye blink in response to something irritating to the eye, or freezing in response to seeing a predator. Experiencing the CS-US predictive relationship over a series of trials causes the animal to learn that the CS predicts the US so that the animal can respond to the CS with a CR that prepares the animal for, or protects it from, the predicted US. Some CRs are similar to the UR but begin earlier and differ in ways that increase their effectiveness. In one intensively studied type of experiment, for example, a tone CS reliably predicts a puff of air (the US) to a rabbit’s eye, triggering a UR consisting of the closure of a protective inner eyelid called the nictitating membrane. After one or more trials, the tone comes to trigger a CR consisting of membrane closure that begins before the air puff and eventually becomes timed so that peak closure occurs just when the air puff is likely to occur. This CR, being initiated in anticipation of the air puff and appropriately timed, offers better protection than simply initiating closure as a reaction to the irritating US. The ability to act in anticipation of important events by learning about predictive relationships among stimuli is so beneficial that it is widely present across the animal kingdom.



14.2.1 Blocking and Higher-order Conditioning

Many interesting properties of classical conditioning have been observed in experiments. Beyond the anticipatory nature of CRs, two widely observed properties figured prominently in the development of classical conditioning models: *blocking* and *higher-order conditioning*. Blocking occurs when an animal fails to learn a CR when a potential CS is presented along with another CS that had been used previously to condition the animal to produce that CR. For example, in the first stage of a blocking experiment involving rabbit nictitating membrane conditioning, a rabbit is first conditioned with a tone CS and an air puff US to produce the CR of closing its nictitating membrane in anticipation of the air puff. The experiment's second stage consists of additional trials in which a second stimulus, say a light, is added to the tone to form a compound tone/light CS followed by the same air puff US. In the experiment's third phase, the second stimulus alone—the light—is presented to the rabbit to see if the rabbit has learned to respond to it with a CR. It turns out that the rabbit produces very few, or no, CRs in response to the light: learning to the light had been *blocked* by the previous learning to the tone.² Blocking results like this challenged the idea that conditioning depends only on simple temporal contiguity, that is, that a necessary and sufficient condition for conditioning is that a US frequently follows a CS closely in time. In the next section we describe the *Rescorla–Wagner model* (Rescorla and Wagner, 1972) that offered an influential explanation for blocking.

Higher-order conditioning occurs when a previously-conditioned CS acts as a US in conditioning another initially neutral stimulus. Pavlov described an experiment in which his assistant first conditioned a dog to salivate to the sound of a metronome that predicted a food US, as described above. After this stage of conditioning, a number of trials were conducted in which a black square, to which the dog was initially indifferent, was placed in the dog's line of vision followed by the sound of the metronome—and this was *not* followed by food. In just ten trials, the dog began to salivate merely upon seeing the black square, despite the fact that the sight of it had never been followed by food. The sound of the metronome itself acted as a US in conditioning a salivation CR to the black square CS. This was second-order conditioning. If the black square had been used as a US to establish salivation CRs to another otherwise neutral CS, it would have been third-order conditioning, and so on. Higher-order conditioning is difficult to demonstrate, especially above the second order, in part because a higher-order reinforcer loses its reinforcing value due to not being repeatedly followed by the original US during higher-order conditioning trials. But under the right conditions, such as intermixing first-order trials with higher-order trials or by providing a general energizing stimulus, higher-order conditioning beyond the second order can be demonstrated. As we describe below, the *TD model of classical conditioning* uses the bootstrapping idea that is central to our approach to extend the Rescorla–Wagner model's account of blocking to include both the anticipatory nature of CRs and higher-order conditioning.

Higher-order instrumental conditioning occurs as well. In this case, a stimulus that

²Comparison with a control group is necessary to show that the previous conditioning to the tone is responsible for blocking learning to the light. This is done by trials with the tone/light CS but with no prior conditioning to the tone. Learning to the light in this case is unimpaired. Moore and Schmajuk (2008) give a full account of this procedure.

consistently predicts primary reinforcement becomes a reinforcer itself, where reinforcement is primary if its rewarding or penalizing quality has been built into the animal by evolution. The predicting stimulus becomes a *secondary reinforcer*, or more generally, a *higher-order* or *conditioned reinforcer*—the latter being a better term when the predicted reinforcing stimulus is itself a secondary, or an even higher-order, reinforcer. A conditioned reinforcer delivers *conditioned reinforcement*: conditioned reward or conditioned penalty. Conditioned reinforcement acts like primary reinforcement in increasing an animal’s tendency to produce behavior that leads to conditioned reward, and to decrease an animal’s tendency to produce behavior that leads to conditioned penalty. (See our comments at the end of this chapter that explain how our terminology sometimes differs, as it does here, from terminology used in psychology.)

Conditioned reinforcement is a key phenomenon that explains, for instance, why we work for the conditioned reinforcer money, whose worth derives solely from what is predicted by having it. In actor–critic methods described in Section 13.5 (and discussed in the context of neuroscience in Sections 15.7 and 15.8), the critic uses a TD method to evaluate the actor’s policy, and its value estimates provide conditioned reinforcement to the actor, allowing the actor to improve its policy. This analog of higher-order instrumental conditioning helps address the credit-assignment problem mentioned in Section 1.7 because the critic gives moment-by-moment reinforcement to the actor when the primary reward signal is delayed. We discuss this more below in Section 14.4.

14.2.2 The Rescorla–Wagner Model

Rescorla and Wagner created their model mainly to account for blocking. The core idea of the Rescorla–Wagner model is that an animal only learns when events violate its expectations, in other words, only when the animal is surprised (although without necessarily implying any *conscious* expectation or emotion). We first present Rescorla and Wagner’s model using their terminology and notation before shifting to the terminology and notation we use to describe the TD model.

Here is how Rescorla and Wagner described their model. The model adjusts the “associative strength” of each component stimulus of a compound CS, which is a number representing how strongly or reliably that component is predictive of a US. When a compound CS consisting of several component stimuli is presented in a classical conditioning trial, the associative strength of each component stimulus changes in a way that depends on an associative strength associated with the entire stimulus compound, called the “aggregate associative strength,” and not just on the associative strength of each component itself.

Rescorla and Wagner considered a compound CS AX, consisting of component stimuli A and X, where the animal may have already experienced stimulus A, and stimulus X might be new to the animal. Let V_A , V_X , and V_{AX} respectively denote the associative strengths of stimuli A, X, and the compound AX. Suppose that on a trial the compound CS AX is followed by a US, which we label stimulus Y. Then the associative strengths of

the stimulus components change according to these expressions:

$$\begin{aligned}\Delta V_A &= \alpha_A \beta_Y (R_Y - V_{AX}) \\ \Delta V_X &= \alpha_X \beta_Y (R_Y - V_{AX}),\end{aligned}$$

where $\alpha_A \beta_Y$ and $\alpha_X \beta_Y$ are the step-size parameters, which depend on the identities of the CS components and the US, and R_Y is the asymptotic level of associative strength that the US Y can support. (Rescorla and Wagner used λ here instead of R , but we use R to avoid confusion with our use of λ and because we usually think of this as the magnitude of a reward signal, with the caveat that the US in classical conditioning is not necessarily rewarding or penalizing.) A key assumption of the model is that the aggregate associative strength V_{AX} is equal to $V_A + V_X$. The associative strengths as changed by these Δ s become the associative strengths at the beginning of the next trial.

To be complete, the model needs a response-generation mechanism, which is a way of mapping values of V s to CRs. Because this mapping would depend on details of the experimental situation, Rescorla and Wagner did not specify a mapping but simply assumed that larger V s would produce stronger or more likely CRs, and that negative V s would mean that there would be no CRs.

The Rescorla–Wagner model accounts for the acquisition of CRs in a way that explains blocking. As long as the aggregate associative strength, V_{AX} , of the stimulus compound is below the asymptotic level of associative strength, R_Y , that the US Y can support, the prediction error $R_Y - V_{AX}$ is positive. This means that over successive trials the associative strengths V_A and V_X of the component stimuli increase until the aggregate associative strength V_{AX} equals R_Y , at which point the associative strengths stop changing (unless the US changes). When a new component is added to a compound CS to which the animal has already been conditioned, further conditioning with the augmented compound produces little or no increase in the associative strength of the added CS component because the error has already been reduced to zero, or to a low value. The occurrence of the US is already predicted nearly perfectly, so little or no error—or surprise—is introduced by the new CS component. Prior learning blocks learning to the new component.

To transition from Rescorla and Wagner’s model to the TD model of classical conditioning (which we just call the TD model), we first recast their model in terms of the concepts that we are using throughout this book. Specifically, we match the notation we use for learning with linear function approximation (Section 9.4), and we think of the conditioning process as one of learning to predict the “magnitude of the US” on a trial on the basis of the compound CS presented on that trial, where the magnitude of a US Y is the R_Y of the Rescorla–Wagner model as given above. We also introduce states. Because the Rescorla–Wagner model is a *trial-level* model, meaning that it deals with how associative strengths change from trial to trial without considering any details about what happens within and between trials, we do not have to consider how states change during a trial until we present the full TD model in the following section. Instead, here we simply think of a state as a way of labeling a trial in terms of the collection of component CSs that are present on the trial.

Therefore, assume that trial-type, or state, s is described by a real-valued vector of features $\mathbf{x}(s) = (x_1(s), x_2(s), \dots, x_d(s))^\top$ where $x_i(s) = 1$ if CS_i , the i^{th} component of a

compound CS, is present on the trial and 0 otherwise. Then if the d -dimensional vector of associative strengths is \mathbf{w} , the aggregate associative strength for trial-type s is

$$\hat{v}(s, \mathbf{w}) = \mathbf{w}^\top \mathbf{x}(s). \quad (14.1)$$

This corresponds to a *value estimate* in reinforcement learning, and we think of it as the *US prediction*.

Now temporally let t denote the number of a complete trial and not its usual meaning as a time step (we revert to t 's usual meaning when we extend this to the TD model below), and assume that S_t is the state corresponding to trial t . Conditioning trial t updates the associative strength vector \mathbf{w}_t to \mathbf{w}_{t+1} as follows:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \delta_t \mathbf{x}(S_t), \quad (14.2)$$

where α is the step-size parameter, and—because here we are describing the Rescorla–Wagner model— δ_t is the *prediction error*

$$\delta_t = R_t - \hat{v}(S_t, \mathbf{w}_t). \quad (14.3)$$

R_t is the target of the prediction on trial t , that is, the magnitude of the US, or in Rescorla and Wagner's terms, the associative strength that the US on the trial can support. Note that because of the factor $\mathbf{x}(S_t)$ in (14.2), only the associative strengths of CS components present on a trial are adjusted as a result of that trial. You can think of the prediction error as a measure of surprise, and the aggregate associative strength as the animal's expectation that is violated when it does not match the target US magnitude.

From the perspective of machine learning, the Rescorla–Wagner model is an error-correction supervised learning rule. It is essentially the same as the Least Mean Square (LMS), or Widrow-Hoff, learning rule (Widrow and Hoff, 1960) that finds the weights—here the associative strengths—that make the average of the squares of all the errors as close to zero as possible. It is a “curve-fitting,” or regression, algorithm that is widely used in engineering and scientific applications (see Section 9.4).³

The Rescorla–Wagner model was very influential in the history of animal learning theory because it showed that a “mechanistic” theory could account for the main facts about blocking without resorting to more complex cognitive theories involving, for example, an animal's explicit recognition that another stimulus component had been added and then scanning its short-term memory backward to reassess the predictive relationships involving the US. The Rescorla–Wagner model showed how traditional contiguity theories of conditioning—that temporal contiguity of stimuli was a necessary and sufficient condition for learning—could be adjusted in a simple way to account for blocking (Moore and Schmajuk, 2008).

The Rescorla–Wagner model provides a simple account of blocking and some other features of classical conditioning, but it is not a complete or perfect model of classical

³The only differences between the LMS rule and the Rescorla–Wagner model are that for LMS the input vectors \mathbf{x}_t can have any real numbers as components, and—at least in the simplest version of the LMS rule—the step-size parameter α does not depend on the input vector or the identity of the stimulus setting the prediction target.

conditioning. Different ideas account for a variety of other observed effects, and progress is still being made toward understanding the many subtleties of classical conditioning. The TD model, which we describe next, though also not a complete or perfect model model of classical conditioning, extends the Rescorla–Wagner model to address how within-trial and between-trial timing relationships among stimuli can influence learning and how higher-order conditioning might arise.

14.2.3 The TD Model

The TD model is a *real-time* model, as opposed to a trial-level model like the Rescorla–Wagner model. A single step t in the Rescorla–Wagner model represents an entire conditioning trial. The model does not apply to details about what happens during the time a trial is taking place, or what might happen between trials. Within each trial an animal might experience various stimuli whose onsets occur at particular times and that have particular durations. These timing relationships strongly influence learning. The Rescorla–Wagner model also does not include a mechanism for higher-order conditioning, whereas for the TD model, higher-order conditioning is a natural consequence of the bootstrapping idea that is at the base of TD algorithms.

To describe the TD model we begin with the formulation of the Rescorla–Wagner model above, but t now labels time steps within or between trials instead of complete trials. Think of the time between t and $t + 1$ as a small time interval, say .01 second, and think of a trial as a sequences of states, one associated with each time step, where the state at step t now represents details of how stimuli are represented at t instead of just a label for the CS components present on a trial. In fact, we can completely abandon the idea of trials. From the point of view of the animal, a trial is just a fragment of its continuing experience interacting with its world. Following our usual view of an agent interacting with its environment, imagine that the animal is experiencing an endless sequence of states s , each represented by a feature vector $\mathbf{x}(s)$. That said, it is still often convenient to refer to trials as fragments of time during which patterns of stimuli repeat in an experiment.

State features are not restricted to describing the external stimuli that an animal experiences; they can describe neural activity patterns that external stimuli produce in an animal’s brain, and these patterns can be history-dependent, meaning that they can be persistent patterns produced by sequences of external stimuli. Of course, we do not know exactly what these neural activity patterns are, but a real-time model like the TD model allows one to explore the consequences on learning of different hypotheses about the internal representations of external stimuli. For these reasons, the TD model does not commit to any particular state representation. In addition, because the TD model includes discounting and eligibility traces that span time intervals between stimuli, the model also makes it possible to explore how discounting and eligibility traces interact with stimulus representations in making predictions about the results of classical conditioning experiments.

Below we describe some of the state representations that have been used with the TD model and some of their implications, but for the moment we stay agnostic about

the representation and just assume that each state s is represented by a feature vector $\mathbf{x}(s) = (x_1(s), x_2(s), \dots, x_n(s))^\top$. Then the aggregate associative strength corresponding to a state s is given by (14.1), the same as for the Rescorla–Wagner model, but the TD model updates the associative strength vector, \mathbf{w} , differently. With t now labeling a time step instead of a complete trial, the TD model governs learning according to this update:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \delta_t \mathbf{z}_t, \quad (14.4)$$

which replaces $\mathbf{x}_t(S_t)$ in the Rescorla–Wagner update (14.2) with \mathbf{z}_t , a vector of eligibility traces, and instead of the δ_t of (14.3), here δ_t is a TD error:

$$\delta_t = R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}_t) - \hat{v}(S_t, \mathbf{w}_t), \quad (14.5)$$

where γ is a discount factor (between 0 and 1), R_t is the prediction target at time t , and $\hat{v}(S_{t+1}, \mathbf{w}_t)$ and $\hat{v}(S_t, \mathbf{w}_t)$ are aggregate associative strengths at $t+1$ and t as defined by (14.1).

Each component i of the eligibility-trace vector \mathbf{z}_t increments or decrements according to the component $x_i(S_t)$ of the feature vector $\mathbf{x}(S_t)$, and otherwise decays with a rate determined by $\gamma\lambda$:

$$\mathbf{z}_{t+1} = \gamma\lambda \mathbf{z}_t + \mathbf{x}(S_t). \quad (14.6)$$

Here λ is the usual eligibility trace decay parameter.

Note that if $\gamma = 0$, the TD model reduces to the Rescorla–Wagner model with the exceptions that: the meaning of t is different in each case (a trial number for the Rescorla–Wagner model and a time step for the TD model), and in the TD model there is a one-time-step lead in the prediction target R . The TD model is equivalent to the backward view of the semi-gradient TD(λ) algorithm with linear function approximation (Chapter 12), except that R_t in the model does not have to be a reward signal as it does when the TD algorithm is used to learn a value function for policy-improvement.

14.2.4 TD Model Simulations

Real-time conditioning models like the TD model are interesting primarily because they make predictions for a wide range of situations that cannot be represented by trial-level models. These situations involve the timing and durations of conditionable stimuli, the timing of these stimuli in relation to the timing of the US, and the timing and shapes of CRs. For example, the US generally must begin after the onset of a neutral stimulus for conditioning to occur, with the rate and effectiveness of learning depending on the inter-stimulus interval, or ISI, the interval between the onsets of the CS and the US. When CRs appear, they generally begin before the appearance of the US and their temporal profiles change during learning. In conditioning with compound CSs, the component stimuli of the compound CSs may not all begin and end at the same time, sometimes forming what is called a *serial compound* in which the component stimuli occur in a sequence over time. Timing considerations like these make it important to consider how stimuli are represented, how these representations unfold over time during and between trials, and how they interact with discounting and eligibility traces.

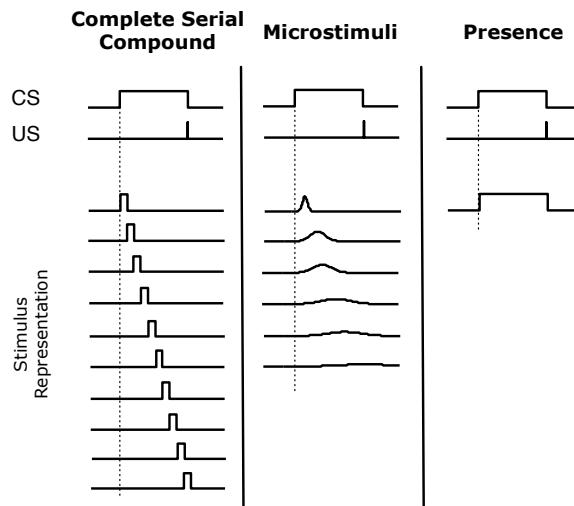


Figure 14.1: Three stimulus representations (in columns) sometimes used with the TD model. Each row represents one element of the stimulus representation. The three representations vary along a temporal generalization gradient, with no generalization between nearby time points in the complete serial compound (left column) and complete generalization between nearby time points in the presence representation (right column). The microstimulus representation occupies a middle ground. The degree of temporal generalization determines the temporal granularity with which US predictions are learned. Adapted with minor changes from *Learning & Behavior*, Evaluating the TD Model of Classical Conditioning, volume 40, 2012, p. 311, E. A. Ludvig, R. S. Sutton, E. J. Kehoe. With permission of Springer.

Figure 14.1 shows three of the stimulus representations that have been used in exploring the behavior of the TD model: the *complete serial compound* (CSC), the *microstimulus* (MS), and the *presence* representations (Ludvig, Sutton, and Kehoe, 2012). These representations differ in the degree to which they force generalization among nearby time points during which a stimulus is present.

The simplest of the representations shown in Figure 14.1 is the presence representation in the figure's right column. This representation has a single feature for each component CS present on a trial, where the feature has value 1 whenever that component is present, and 0 otherwise.⁴ The presence representation is not a realistic hypothesis about how stimuli are represented in an animal's brain, but as we describe below, the TD model with this representation can produce many of the timing phenomena seen in classical conditioning.

For the CSC representation (left column of Figure 14.1), the onset of each external stimulus initiates a sequence of precisely-timed short-duration internal signals that

⁴In our formalism, there is a different state, S_t , for each time step t during a trial, and for a trial in which a compound CS consists of n component CSs of various durations occurring at various times throughout the trial, there is a feature, x_i , for each component CS_i, $i = 1, \dots, n$, where $x_i(S_t) = 1$ for all times t when the CS_i is present, and equals zero otherwise.

continues until the external stimulus ends.⁵ This is like assuming the animal's nervous system has a clock that keeps precise track of time during stimulus presentations; it is what engineers call a "tapped delay line." Like the presence representation, the CSC representation is unrealistic as a hypothesis about how the brain internally represents stimuli, but Ludvig et al. (2012) call it a "useful fiction" because it can reveal details of how the TD model works when relatively unconstrained by the stimulus representation. The CSC representation is also used in most TD models of dopamine-producing neurons in the brain, a topic we take up in Chapter 15. The CSC representation is often viewed as an essential part of the TD model, although this view is mistaken.

The MS representation (center column of Figure 14.1) is like the CSC representation in that each external stimulus initiates a cascade of internal stimuli, but in this case the internal stimuli—the microstimuli—are not of such limited and non-overlapping form; they are extended over time and overlap. As time elapses from stimulus onset, different sets of microstimuli become more or less active, and each subsequent microstimulus becomes progressively wider in time and reaches a lower maximal level. Of course, there are many MS representations depending on the nature of the microstimuli, and a number of examples of MS representations have been studied in the literature, in some cases along with proposals for how an animal's brain might generate them (see the Bibliographic and Historical Comments at the end of this chapter). MS representations are more realistic than the presence or CSC representations as hypotheses about neural representations of stimuli, and they allow the behavior of the TD model to be related to a broader collection of phenomena observed in animal experiments. In particular, by assuming that cascades of microstimuli are initiated by USs as well as by CSs, and by studying the significant effects on learning of interactions between microstimuli, eligibility traces, and discounting, the TD model is helping to frame hypotheses to account for many of the subtle phenomena of classical conditioning and how an animal's brain might produce them. We say more about this below, particularly in Chapter 15 where we discuss reinforcement learning and neuroscience.

Even with the simple presence representation, however, the TD model produces all the basic properties of classical conditioning that are accounted for by the Rescorla–Wagner model, plus features of conditioning that are beyond the scope of trial-level models. For example, as we have already mentioned, a conspicuous feature of classical conditioning is that the US generally must begin *after* the onset of a neutral stimulus for conditioning to occur, and that after conditioning, the CR begins *before* the appearance of the US. In other words, conditioning generally requires a positive ISI, and the CR generally anticipates the US. How the strength of conditioning (e.g., the percentage of CRs elicited by a CS) depends on the ISI varies substantially across species and response systems, but it typically has the following properties: it is negligible for a zero or negative ISI, i.e., when the US onset occurs simultaneously with, or earlier than, the CS onset (although research has found that associative strengths sometimes increase slightly or become negative with

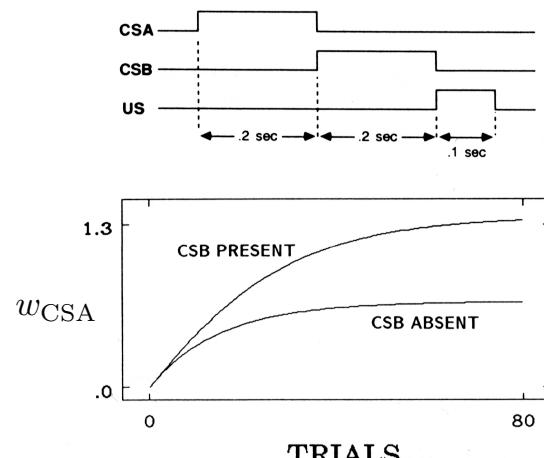
⁵In our formalism, for each CS component CS_i present on a trial, and for each time step t during a trial, there is a separate feature x_i^t , where $x_i^t(S_{t'}) = 1$ if $t = t'$ for any t' at which CS_i is present, and equals 0 otherwise. This is different from the CSC representation in Sutton and Barto (1990) in which there are the same distinct features for each time step but no reference to external stimuli; hence the name complete serial compound.

negative ISIs); it increases to a maximum at a positive ISI where conditioning is most effective; and it then decreases to zero after an interval that varies widely with response systems. The precise shape of this dependency for the TD model depends on the values of its parameters and details of the stimulus representation, but these basic features of ISI-dependency are core properties of the TD model.

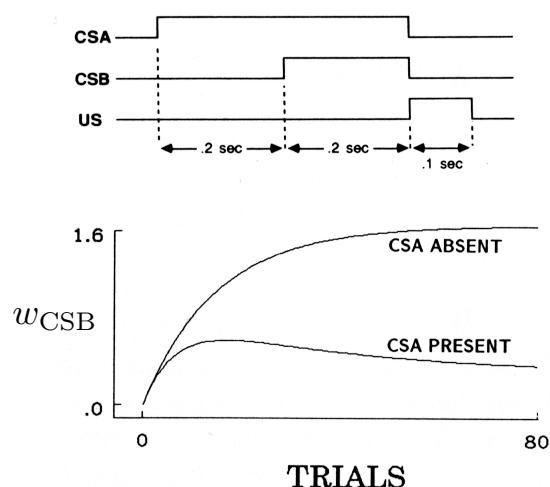
One of the theoretical issues arising with serial-compound conditioning, that is, conditioning with a compound CS whose components occur in a sequence, concerns the facilitation of remote associations. It has been found that if the empty trace interval between a first CS (CSA) and the US is filled with a second CS (CSB) to form a serial-compound stimulus, then conditioning to CSA is facilitated. Shown to the right is the behavior of the TD model with the presence representation in a simulation of such an experiment whose timing details are shown above. Consistent with the experimental results (Kehoe, 1982), the model shows facilitation of both the rate of conditioning and the asymptotic level of conditioning of the first CS due to the presence of the second CS.

A well-known demonstration of the effects on conditioning of temporal relationships among stimuli within a trial is an experiment by Egger and Miller (1962) that involved two overlapping CSs in a delay configuration as shown to the right (top). Although CSB was in a better temporal relationship with the US, the presence of CSA substantially reduced conditioning to CSB as compared to controls in which CSA was absent. Directly to the right is shown the same result being generated by the TD model in a simulation of this experiment with the presence representation.

The TD model accounts for blocking because it is an error-correcting learning rule like the Rescorla-Wagner model. Beyond accounting for basic blocking re-



Facilitation of remote associations in the TD model



The Egger-Miller effect in the TD model

sults, however, the TD model predicts (with the presence representation and more complex representations as well) that blocking is reversed if the blocked stimulus is moved earlier in time so that its onset occurs before the onset of the blocking stimulus (like CSA in the diagram to the right). This feature of the TD model's behavior deserves attention because it had not been observed at the time of the model's introduction. Recall that in blocking, if an animal has already learned that one CS predicts a US, then learning that a newly-added second CS also predicts the US is much reduced, i.e., is blocked. But if the newly-added second CS begins earlier than the pretrained CS, then—according to the TD model—learning to the newly-added CS is not blocked. In fact, as training continues and the newly-added CS gains associative strength, the pretrained CS loses associative strength. The behavior of the TD model under these conditions is shown in the lower part of Figure 14.2.

This simulation experiment differed from the Egger-Miller experiment (bottom of the preceding page) in that the shorter CS with the later onset was given prior training until it was fully associated with the US. This surprising prediction led Kehoe, Schreurs, and Graham (1987) to conduct the experiment using the well-studied rabbit nictitating membrane preparation. Their results confirmed the model's prediction, and they noted that non-TD models have considerable difficulty explaining their data.

With the TD model, an earlier predictive stimulus takes precedence over a later predictive stimulus because, like all the prediction methods described in this book, the TD model is based on the backing-up or bootstrapping idea: updates to associative strengths shift the strengths at a particular state toward the strength at later states. Another consequence of bootstrapping is that the TD model provides an account of higher-order conditioning, a feature of classical conditioning that is beyond the scope of the Rescoral-Wagner and similar models. As we described above, higher-order conditioning is the phenomenon in which a previously-conditioned CS can act as a US in conditioning another initially neutral stimulus. Figure 14.3 shows the behavior of the TD model (again with the presence representation) in a higher-order conditioning experiment—in this case it is second-order conditioning. In the first phase (not shown in the figure), CSB is trained to predict a US so that its associative strength increases, here to 1.65. In the second phase, CSA is paired with CSB in the absence of the US, in the sequential arrangement shown at the top of the figure. CSA acquires associative strength even though it is never paired with the US. With continued training, CSA's associative strength reaches a peak and then decreases because the associative strength of CSB, the secondary reinforcer, decreases so that it loses its ability to provide secondary reinforcement. CSB's associative

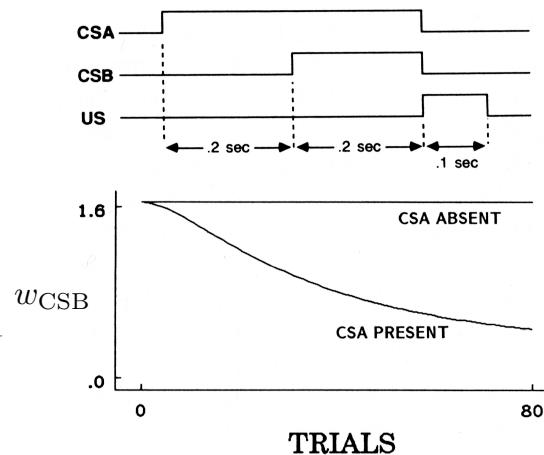


Figure 14.2: Temporal primacy overriding blocking in the TD model.

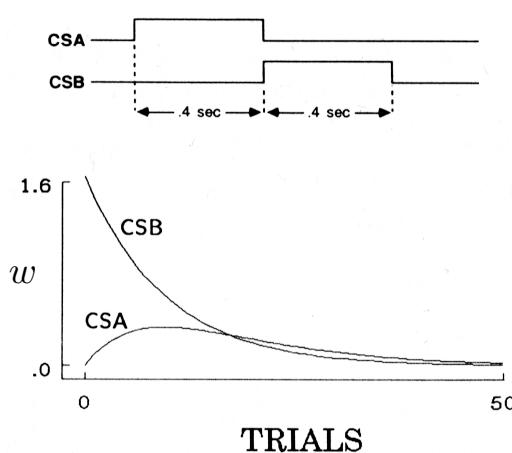


Figure 14.3: Second-order conditioning with the TD model.

can differ from $\hat{v}(S_t, \mathbf{w}_t)$, making δ_t non-zero (a temporal difference). This difference has the same status as R_{t+1} in (14.5), implying that as far as learning is concerned there is no difference between a temporal difference and the occurrence of a US. In fact, this feature of the TD algorithm is one of the major reasons for its development, which we now understand through its connection to dynamic programming as described in Chapter 6. Bootstrapping values is intimately related to second-order, and higher-order, conditioning.

In the examples of the TD model's behavior described above, we examined only the changes in the associative strengths of the CS components; we did not look at what the model predicts about properties of an animal's conditioned responses (CRs): their timing, shape, and how they develop over conditioning trials. These properties depend on the species, the response system being observed, and parameters of the conditioning trials, but in many experiments with different animals and different response systems, the magnitude of the CR, or the probability of a CR, increases as the expected time of the US approaches. For example, in classical conditioning of a rabbit's nictitating membrane response that we mentioned above, over conditioning trials the delay from CS onset to when the nictitating membrane begins to move across the eye decreases over trials, and the amplitude of this anticipatory closure gradually increases over the interval between the CS and the US until the membrane reaches maximal closure at the expected time of the US. The timing and shape of this CR is critical to its adaptive significance—covering the eye too early reduces vision (even though the nictitating membrane is translucent), while covering it too late is of little protective value. Capturing CR features like these is challenging for models of classical conditioning.

The TD model does not include as part of its definition any mechanism for translating the time course of the US prediction, $\hat{v}(S_t, \mathbf{w}_t)$, into a profile that can be compared with the properties of an animal's CR. The simplest choice is to let the time course of

strength decreases because the US does not occur in these higher-order conditioning trials. These are *extinction trials* for CSB because its predictive relationship to the US is disrupted so that its ability to act as a reinforcer decreases. This same pattern is seen in animal experiments. This extinction of conditioned reinforcement in higher-order conditioning trials makes it difficult to demonstrate higher-order conditioning unless the original predictive relationships are periodically refreshed by occasionally inserting first-order trials.

The TD model produces an analog of second- and higher-order conditioning because $\gamma\hat{v}(S_{t+1}, \mathbf{w}_t) - \hat{v}(S_t, \mathbf{w}_t)$ appears in the TD error δ_t (14.5). This means that as a result of previous learning, $\gamma\hat{v}(S_{t+1}, \mathbf{w}_t)$

a simulated CR equal the time course of the US prediction. In this case, features of simulated CRs and how they change over trials depend only on the stimulus representation chosen and the values of the model's parameters α , γ , and λ .

Figure 14.4 shows the time courses of US predictions at different points during learning with the three representations shown in Figure 14.1. For these simulations the US occurred 25 time steps after the onset of the CS, and $\alpha = .05$, $\lambda = .95$ and $\gamma = .97$. With the CSC representation (Figure 14.4 left), the curve of the US prediction formed by the TD model increases exponentially throughout the interval between the CS and the US until it reaches a maximum exactly when the US occurs (at time step 25). This exponential increase is the result of discounting in the TD model learning rule. With the presence representation (Figure 14.4 middle), the US prediction is nearly constant while the stimulus is present because there is only one weight, or associative strength, to be learned for each stimulus. Consequently, the TD model with the presence representation cannot recreate many features of CR timing. With an MS representation (Figure 14.4 right), the development of the TD model's US prediction is more complicated. After 200 trials the prediction's profile is a reasonable approximation of the US prediction curve produced with the CSC representation.

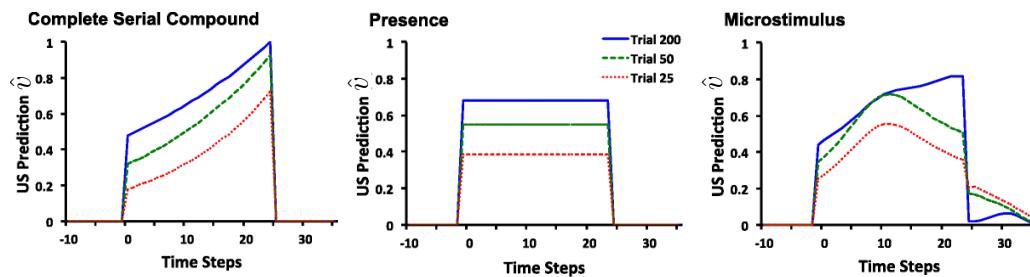


Figure 14.4: Time course of US prediction over the course of acquisition for the TD model with three different stimulus representations. Left: With the complete serial compound (CSC), the US prediction increases exponentially through the interval, peaking at the time of the US. At asymptote (trial 200), the US prediction peaks at the US intensity (1 in these simulations). Middle: With the presence representation, the US prediction converges to an almost constant level. This constant level is determined by the US intensity and the length of the CS-US interval. Right: With the microstimulus representation, at asymptote, the TD model approximates the exponentially increasing time course depicted with the CSC through a linear combination of the different microstimuli. Adapted with minor changes from *Learning & Behavior*, Evaluating the TD Model of Classical Conditioning, volume 40, 2012, E. A. Ludvig, R. S. Sutton, E. J. Kehoe. With permission of Springer.

The US prediction curves shown in Figure 14.4 were not intended to precisely match profiles of CRs as they develop during conditioning in any particular animal experiment, but they illustrate the strong influence that the stimulus representation has on predictions derived from the TD model. Further, although we can only mention it here, how the stimulus representation interacts with discounting and eligibility traces is important

in determining properties of the US prediction profiles produced by the TD model. Another dimension beyond what we can discuss here is the influence of different response-generation mechanisms that translate US predictions into CR profiles; the profiles shown in Figure 14.4 are “raw” US prediction profiles. Even without any special assumption about how an animal’s brain might produce overt responses from US predictions, however, the profiles in Figure 14.4 for the CSC and MS representations increase as the time of the US approaches and reach a maximum at the time of the US, as is seen in many animal conditioning experiments.

The TD model, when combined with particular stimulus representations and response-generation mechanisms, is able to account for a surprisingly wide range of phenomena observed in animal classical conditioning experiments, but it is far from being a perfect model. To generate other details of classical conditioning the model needs to be extended, perhaps by adding model-based elements and mechanisms for adaptively altering some of its parameters. Other approaches to modeling classical conditioning depart significantly from the Rescorla–Wagner-style error-correction process. Bayesian models, for example, work within a probabilistic framework in which experience revises probability estimates. All of these models usefully contribute to our understanding of classical conditioning.

Perhaps the most notable feature of the TD model is that it is based on a theory—the theory we have described in this book—that suggests an account of what an animal’s nervous system is *trying to do* while undergoing conditioning: it is trying to form accurate *long-term predictions*, consistent with the limitations imposed by the way stimuli are represented and how the nervous system works. In other words, it suggests a *normative account* of classical conditioning in which long-term, instead of immediate, prediction is a key feature.

The development of the TD model of classical conditioning is one instance in which the explicit goal was to model some of the details of animal learning behavior. In addition to its standing as an *algorithm*, then, TD learning is also the basis of this *model* of aspects of biological learning. As we discuss in Chapter 15, TD learning has also turned out to underlie an influential model of the activity of neurons that produce dopamine, a chemical in the brain of mammals that is deeply involved in reward processing. These are instances in which reinforcement learning theory makes detailed contact with animal behavioral and neural data.

We now turn to considering correspondences between reinforcement learning and animal behavior in instrumental conditioning experiments, the other major type of laboratory experiment studied by animal learning psychologists.

14.3 Instrumental Conditioning

In *instrumental conditioning* experiments learning depends on the consequences of behavior: the delivery of a reinforcing stimulus is contingent on what the animal does. In classical conditioning experiments, in contrast, the reinforcing stimulus—the US—is delivered independently of the animal’s behavior. Instrumental conditioning is usually considered to be the same as *operant conditioning*, the term B. F. Skinner (1938, 1963) introduced for experiments with behavior-contingent reinforcement, though the experi-

ments and theories of those who use these two terms differ in a number of ways, some of which we touch on below. We will exclusively use the term instrumental conditioning for experiments in which reinforcement is contingent upon behavior. The roots of instrumental conditioning go back to experiments performed by the American psychologist Edward Thorndike one hundred years before publication of the first edition of this book.

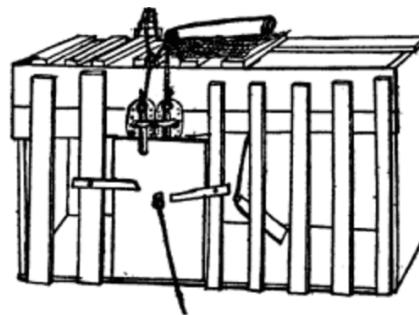
Thorndike observed the behavior of cats when they were placed in “puzzle boxes,” such as the one at the right, from which they could escape by appropriate actions. For example, a cat could open the door of one box by performing a sequence of three separate actions: depressing a platform at the back of the box, pulling a string by clawing at it, and pushing a bar up or down. When first placed in a puzzle box, with food visible outside, all but a few of Thorndike’s cats displayed “evident signs of discomfort” and extraordinarily vigorous activity “to strive instinctively to escape from confinement” (Thorndike, 1898).

In experiments with different cats and boxes with different escape mechanisms, Thorndike recorded the amounts of time each cat took to escape over multiple experiences in each box. He observed that the time almost invariably decreased with successive experiences, for example, from 300 seconds to 6 or 7 seconds. He described cats’ behavior in a puzzle box like this:

The cat that is clawing all over the box in her impulsive struggle will probably claw the string or loop or button so as to open the door. And gradually all the other non-successful impulses will be stamped out and the particular impulse leading to the successful act will be stamped in by the resulting pleasure, until, after many trials, the cat will, when put in the box, immediately claw the button or loop in a definite way. (Thorndike 1898, p. 13)

These and other experiments (some with dogs, chicks, monkeys, and even fish) led Thorndike to formulate a number of “laws” of learning, the most influential being the *Law of Effect*, a version of which we quoted in Chapter 1 (page 15). This law describes what is generally known as learning by trial and error. As mentioned in Chapter 1, many aspects of the Law of Effect have generated controversy, and its details have been modified over the years. Still the law—in one form or another—expresses an enduring principle of learning.

Essential features of reinforcement learning algorithms correspond to features of animal learning described by the Law of Effect. First, reinforcement learning algorithms are *selectional*, meaning that they try alternatives and select among them by comparing their consequences. Second, reinforcement learning algorithms are *associative*, meaning that the alternatives found by selection are associated with particular situations, or states, to form the agent’s policy. Like learning described by the Law of Effect, reinforcement



One of Thorndike’s puzzle boxes.

Reprinted from Thorndike, *Animal Intelligence: An Experimental Study of the Associative Processes in Animals*, *The Psychological Review, Series of Monograph Supplements* II(4), Macmillan, New York, 1898.

learning is not just the process of *finding* actions that produce a lot of reward, but also of *connecting* these actions to situations or states. Thorndike used the phrase learning by “selecting and connecting” (Hilgard, 1956). Natural selection in evolution is a prime example of a selectional process, but it is not associative (at least as it is commonly understood); supervised learning is associative, but it is not selectional because it relies on instructions that directly tell the agent how to change its behavior.

In computational terms, 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 all reinforcement learning algorithms, whether memory takes the form of an agent’s policy, value function, or environment model.

A reinforcement learning algorithm’s need to search means that it has to explore in some way. Animals clearly explore as well, and early animal learning researchers disagreed about the degree of guidance an animal uses in selecting its actions in situations like Thorndike’s puzzle boxes. Are actions the result of “absolutely random, blind groping” (Woodworth, 1938, p. 777), or is there some degree of guidance, either from prior learning, reasoning, or other means? Although some thinkers, including Thorndike, seem to have taken the former position, others favored more deliberate exploration. Reinforcement learning algorithms allow wide latitude for how much guidance an agent can employ in selecting actions. The forms of exploration we have used in the algorithms presented in this book, such as ε -greedy and upper-confidence-bound action selection, are merely among the simplest. More sophisticated methods are possible, with the only stipulation being that there has to be *some* form of exploration for the algorithms to work effectively.

The feature of our treatment of reinforcement learning allowing the set of actions available at any time to depend on the environment’s current state echoes something Thorndike observed in his cats’ puzzle-box behaviors. The cats selected actions from those that they instinctively perform in their current situation, which Thorndike called their “instinctual impulses.” First placed in a puzzle box, a cat instinctively scratches, claws, and bites with great energy: a cat’s instinctual responses to finding itself in a confined space. Successful actions are selected from these and not from every possible action or activity. This is like the feature of our formalism where the action selected from a state s belongs to a set of admissible actions, $\mathcal{A}(s)$. Specifying these sets is an important aspect of reinforcement learning because it can radically simplify learning. They are like an animal’s instinctual impulses. On the other hand, Thorndike’s cats might have been exploring according to an instinctual context-specific *ordering* over actions rather than by just selecting from a set of instinctual impulses. This is another way to make reinforcement learning easier.

Among the most prominent animal learning researchers influenced by the Law of Effect were Clark Hull (e.g., Hull, 1943) and B. F. Skinner (e.g., Skinner, 1938). At the center of their research was the idea of selecting behavior on the basis of its consequences. Reinforcement learning has features in common with Hull’s theory, which included eligibility-like mechanisms and secondary reinforcement to account for the ability to learn when there is a significant time interval between an action and the consequent reinforcing

stimulus (see Section 14.4). Randomness also played a role in Hull's theory through what he called "behavioral oscillation" to introduce exploratory behavior.

Skinner did not fully subscribe to the memory aspect of the Law of Effect. Being averse to the idea of associative linkages, he instead emphasized selection from spontaneously-emitted behavior. He introduced the term "operant" to emphasize the key role of an action's effects on an animal's environment. Unlike the experiments of Thorndike and others, which consisted of sequences of separate trials, Skinner's operant conditioning experiments allowed animal subjects to behave for extended periods of time without interruption. He invented the operant conditioning chamber, now called a "Skinner box," the most basic version of which contains a lever or key that an animal can press to obtain a reward, such as food or water, which would be delivered according to a well-defined rule, called a reinforcement schedule. By recording the cumulative number of lever presses as a function of time, Skinner and his followers could investigate the effect of different reinforcement schedules on the animal's rate of lever-pressing. Modeling results from experiments like these using the reinforcement learning principles we present in this book is not well developed, but we mention some exceptions in the Bibliographic and Historical Remarks section at the end of this chapter.

Another of Skinner's contributions resulted from his recognition of the effectiveness of training an animal by reinforcing successive approximations of the desired behavior, a process he called *shaping*. Although this technique had been used by others, including Skinner himself, its significance was impressed upon him when he and colleagues were attempting to train a pigeon to bowl by swiping a wooden ball with its beak. After waiting for a long time without seeing any swipe that they could reinforce, they

... decided to reinforce any response that had the slightest resemblance to a swipe—perhaps, at first, merely the behavior of looking at the ball—and then to select responses which more closely approximated the final form. The result amazed us. In a few minutes, the ball was caroming off the walls of the box as if the pigeon had been a champion squash player. (Skinner, 1958, p. 94)

Not only did the pigeon learn a behavior that is unusual for pigeons, it learned quickly through an interactive process in which its behavior and the reinforcement contingencies changed in response to each other. Skinner compared the process of altering reinforcement contingencies to the work of a sculptor shaping clay into a desired form. Shaping is a powerful technique for computational reinforcement learning systems as well. When it is difficult for an agent to receive any non-zero reward signal at all, either due to sparseness of rewarding situations or their inaccessibility given initial behavior, starting with an easier problem and incrementally increasing its difficulty as the agent learns can be an effective, and sometimes indispensable, strategy.

A concept from psychology that is especially relevant in the context of instrumental conditioning is *motivation*, which refers to processes that influence the direction and strength, or vigor, of behavior. Thorndike's cats, for example, were motivated to escape from puzzle boxes because they wanted the food that was sitting just outside. Obtaining this goal was rewarding to them and reinforced the actions allowing them to escape. It is difficult to link the concept of motivation, which has many dimensions, in a precise

way to reinforcement learning’s computational perspective, but there are clear links with some of its dimensions.

In one sense, a reinforcement learning agent’s reward signal is at the base of its motivation: the agent is motivated to maximize the total reward it receives over the long run. A key facet of motivation, then, is what makes an agent’s experience rewarding. In reinforcement learning, reward signals depend on the state of the reinforcement learning agent’s environment and the agent’s actions. Further, as pointed out in Chapter 1, the state of the agent’s environment not only includes information about what is external to the machine, like an organism or a robot, that houses the agent, but also what is internal to this machine. Some internal state components correspond to what psychologists call an animal’s *motivational state*, which influences what is rewarding to the animal. For example, an animal will be more rewarded by eating when it is hungry than when it has just finished a satisfying meal. The concept of state dependence is broad enough to allow for many types of modulating influences on the generation of reward signals.

Value functions provide a further link to psychologists’ concept of motivation. If the most basic motive for selecting an action is to obtain as much reward as possible, for a reinforcement learning agent that selects actions using a value function, a more proximal motive is to *ascend the gradient of its value function*, that is, to select actions expected to lead to the most highly-valued next states (or what is essentially the same thing, to select actions with the greatest action-values). For these agents, value functions are the main driving force determining the direction of their behavior.

Another dimension of motivation is that an animal’s motivational state not only influences learning, but also influences the strength, or vigor, of the animal’s behavior after learning. For example, after learning to find food in the goal box of a maze, a hungry rat will run faster to the goal box than one that is not hungry. This aspect of motivation does not link so cleanly to the reinforcement learning framework we present here, but in the Bibliographical and Historical Remarks section at the end of this chapter we cite several publications that propose theories of behavioral vigor based on reinforcement learning.

We turn now to the subject of learning when reinforcing stimuli occur well after the events they reinforce. The mechanisms used by reinforcement learning algorithms to enable learning with delayed reinforcement—eligibility traces and TD learning—closely correspond to psychologists’ hypotheses about how animals can learn under these conditions.

14.4 Delayed Reinforcement

The Law of Effect requires a backward effect on connections, and some early critics of the law could not conceive of how the present could affect something that was in the past. This concern was amplified by the fact that learning can even occur when there is a considerable delay between an action and the consequent reward or penalty. Similarly, in classical conditioning, learning can occur when US onset follows CS offset by a non-negligible time interval. We call this the problem of delayed reinforcement, which is related to what Minsky (1961) called the “credit-assignment problem for learning systems”: how do you

distribute credit for success among the many decisions that may have been involved in producing it? The reinforcement learning algorithms presented in this book include two basic mechanisms for addressing this problem. The first is the use of eligibility traces, and the second is the use of TD methods to learn value functions that provide nearly immediate evaluations of actions (in tasks like instrumental conditioning experiments) or that provide immediate prediction targets (in tasks like classical conditioning experiments). Both of these methods correspond to similar mechanisms proposed in theories of animal learning.

Pavlov (1927) pointed out that every stimulus must leave a trace in the nervous system that persists for some time after the stimulus ends, and he proposed that stimulus traces make learning possible when there is a temporal gap between the CS offset and the US onset. To this day, conditioning under these conditions is called *trace conditioning* (page 344). Assuming a trace of the CS remains when the US arrives, learning occurs through the simultaneous presence of the trace and the US. We discuss some proposals for trace mechanisms in the nervous system in Chapter 15.

Stimulus traces were also proposed as a means for bridging the time interval between actions and consequent rewards or penalties in instrumental conditioning. In Hull's influential learning theory, for example, "molar stimulus traces" accounted for what he called an animal's *goal gradient*, a description of how the maximum strength of an instrumentally-conditioned response decreases with increasing delay of reinforcement (Hull, 1932, 1943). Hull hypothesized that an animal's actions leave internal stimuli whose traces decay exponentially as functions of time since an action was taken. Looking at the animal learning data available at the time, he hypothesized that the traces effectively reach zero after 30 to 40 seconds.

The eligibility traces used in the algorithms described in this book are like Hull's traces: they are decaying traces of past state visitations, or of past state-action pairs. Eligibility traces were introduced by Klopff (1972) in his neuronal theory in which they are temporally-extended traces of past activity at synapses, the connections between neurons. Klopff's traces are more complex than the exponentially-decaying traces our algorithms use, and we discuss this more when we take up his theory in Section 15.9.

To account for goal gradients that extend over longer time periods than spanned by stimulus traces, Hull (1943) proposed that longer gradients result from conditioned reinforcement passing backwards from the goal, a process acting in conjunction with his molar stimulus traces. Animal experiments showed that if conditions favor the development of conditioned reinforcement during a delay period, learning does not decrease with increased delay as much as it does under conditions that obstruct secondary reinforcement. Conditioned reinforcement is favored if there are stimuli that regularly occur during the delay interval. Then it is as if reward is not actually delayed because there is more immediate conditioned reinforcement. Hull therefore envisioned that there is a primary gradient based on the delay of the primary reinforcement mediated by stimulus traces, and that this is progressively modified, and lengthened, by conditioned reinforcement.

Algorithms presented in this book that use both eligibility traces and value functions to enable learning with delayed reinforcement correspond to Hull's hypothesis about how

animals are able to learn under these conditions. The actor–critic architecture discussed in Sections 13.5, 15.7, and 15.8 illustrates this correspondence most clearly. The critic uses a TD algorithm to learn a value function associated with the system’s current behavior, that is, to predict the current policy’s return. The actor updates the current policy based on the critic’s predictions, or more exactly, on changes in the critic’s predictions. The TD error produced by the critic acts as a conditioned reinforcement signal for the actor, providing an immediate evaluation of performance even when the primary reward signal itself is considerably delayed. Algorithms that estimate action-value functions, such as Q-learning and Sarsa, similarly use TD learning principles to enable learning with delayed reinforcement by means of conditioned reinforcement. The close parallel between TD learning and the activity of dopamine producing neurons that we discuss in Chapter 15 lends additional support to links between reinforcement learning algorithms and this aspect of Hull’s learning theory.

14.5 Cognitive Maps

Model-based reinforcement learning algorithms use environment models that have elements in common with what psychologists call *cognitive maps*. Recall from our discussion of planning and learning in Chapter 8 that by an environment model we mean anything an agent can use to predict how its environment will respond to its actions in terms of state transitions and rewards, and by planning we mean any process that computes a policy from such a model. Environment models consist of two parts: the state-transition part encodes knowledge about the effect of actions on state changes, and the reward-model part encodes knowledge about the reward signals expected for each state or each state–action pair. A model-based algorithm selects actions by using a model to predict the consequences of possible courses of action in terms of future states and the reward signals expected to arise from those states. The simplest kind of planning is to compare the predicted consequences of collections of “imagined” sequences of decisions.

Questions about whether or not animals use environment models, and if so, what are the models like and how are they learned, have played influential roles in the history of animal learning research. Some researchers challenged the then-prevailing stimulus-response (S–R) view of learning and behavior, which corresponds to the simplest model-free way of learning policies, by demonstrating *latent learning*. In the earliest latent learning experiment, two groups of rats were run in a maze. For the experimental group, there was no reward during the first stage of the experiment, but food was suddenly introduced into the goal box of the maze at the start of the second stage. For the control group, food was in the goal box throughout both stages. The question was whether or not rats in the experimental group would have learned anything during the first stage in the absence of food reward. Although the experimental rats did not *appear* to learn much during the first, unrewarded, stage, as soon as they discovered the food that was introduced in the second stage, they rapidly caught up with the rats in the control group. It was concluded that “during the non-reward period, the rats [in the experimental group] were developing a latent learning of the maze which they were able to utilize as soon as reward was introduced” (Blodgett, 1929).

Latent learning is most closely associated with the psychologist Edward Tolman, who interpreted this result, and others like it, as showing that animals could learn a “cognitive map of the environment” in the absence of rewards or penalties, and that they could use the map later when they were motivated to reach a goal (Tolman, 1948). A cognitive map could also allow a rat to plan a route to the goal that was different from the route the rat had used in its initial exploration. Explanations of results like these led to the enduring controversy lying at the heart of the behaviorist/cognitive dichotomy in psychology. In modern terms, cognitive maps are not restricted to models of spatial layouts but are more generally environment models, or models of an animal’s “task space” (e.g., Wilson, Takahashi, Schoenbaum, and Niv, 2014). The cognitive map explanation of latent learning experiments is analogous to the claim that animals use model-based algorithms, and that environment models can be learned even without explicit rewards or penalties. Models are then used for planning when the animal is motivated by the appearance of rewards or penalties.

Tolman’s account of how animals learn cognitive maps was that they learn stimulus-stimulus, or S–S, associations by experiencing successions of stimuli as they explore an environment. In psychology this is called *expectancy theory*: given S–S associations, the occurrence of a stimulus generates an expectation about the stimulus to come next. This is much like what control engineers call *system identification*, in which a model of a system with unknown dynamics is learned from labeled training examples. In the simplest discrete-time versions, training examples are S–S' pairs, where S is a state and S', the subsequent state, is the label. When S is observed, the model creates the “expectation” that S' will be observed next. Models more useful for planning involve actions as well, so that examples look like SA–S', where S' is expected when action A is executed in state S. It is also useful to learn how the environment generates rewards. In this case, examples are of the form S–R or SA–R, where R is a reward signal associated with S or the SA pair. These are all forms of supervised learning by which an agent can acquire cognitive-like maps whether or not it receives any non-zero reward signals while exploring its environment.

14.6 Habitual and Goal-directed Behavior

The distinction between model-free and model-based reinforcement learning algorithms corresponds to the distinction psychologists make between *habitual* and *goal-directed* control of learned behavioral patterns. Habits are behavior patterns triggered by appropriate stimuli and then performed more-or-less automatically. Goal-directed behavior, according to how psychologists use the phrase, is purposeful in the sense that it is controlled by knowledge of the value of goals and the relationship between actions and their consequences. Habits are sometimes said to be controlled by antecedent stimuli, whereas goal-directed behavior is said to be controlled by its consequences (Dickinson, 1980, 1985). Goal-directed control has the advantage that it can rapidly change an animal’s behavior when the environment changes its way of reacting to the animal’s actions. While habitual behavior responds quickly to input from an accustomed environment, it is unable to quickly adjust to changes in the environment. The development of goal-directed

behavioral control was likely a major advance in the evolution of animal intelligence.

Figure 14.5 illustrates the difference between model-free and model-based decision strategies in a hypothetical task in which a rat has to navigate a maze that has distinctive goal boxes, each delivering an associated reward of the magnitude shown (Figure 14.5 top). Starting at S_1 , the rat has to first select left (L) or right (R) and then has to select L or R again at S_2 or S_3 to reach one of the goal boxes. The goal boxes are the terminal states of each episode of the rat's episodic task. A model-free strategy (Figure 14.5 lower left) relies on stored values for state-action pairs. These action values are estimates of the highest return the rat can expect for each action taken from each (nonterminal) state. They are obtained over many trials of running the maze from start to finish. When the action values have become good enough estimates of the optimal returns, the rat just has to select at each state the action with the largest action value in order to make optimal

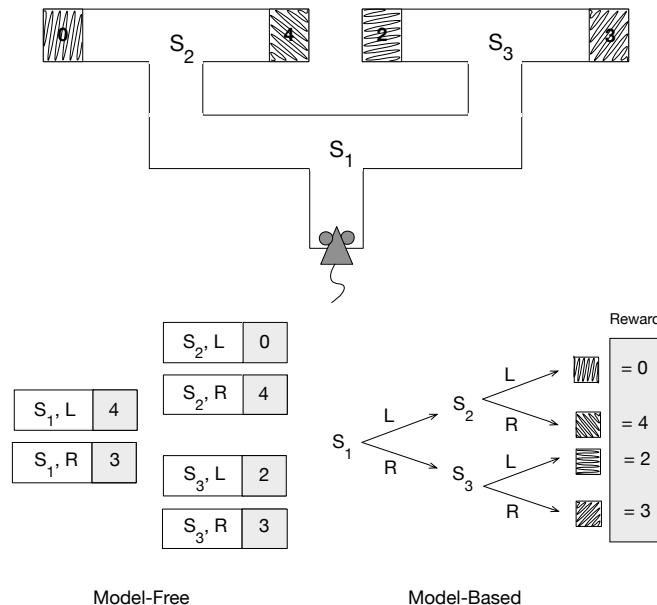


Figure 14.5: Model-based and model-free strategies to solve a hypothetical sequential action-selection problem. Top: a rat navigates a maze with distinctive goal boxes, each associated with a reward having the value shown. Lower left: a model-free strategy relies on stored action values for all the state-action pairs obtained over many learning trials. To make decisions the rat just has to select at each state the action with the largest action value for that state. Lower right: in a model-based strategy, the rat learns an environment model, consisting of knowledge of state-action-next-state transitions and a reward model consisting of knowledge of the reward associated with each distinctive goal box. The rat can decide which way to turn at each state by using the model to simulate sequences of action choices to find a path yielding the highest return. Adapted from *Trends in Cognitive Science*, volume 10, number 8, Y. Niv, D. Joel, and P. Dayan, A Normative Perspective on Motivation, p. 376, 2006, with permission from Elsevier.

decisions. In this case, when the action-value estimates become accurate enough, the rat selects L from S_1 and R from S_2 to obtain the maximum return of 4. A different model-free strategy might simply rely on a cached policy instead of action values, making direct links from S_1 to L and from S_2 to R. In neither of these strategies do decisions rely on an environment model. There is no need to consult a state-transition model, and no connection is required between the features of the goal boxes and the rewards they deliver.

Figure 14.5 (lower right) illustrates a model-based strategy. It uses an environment model consisting of a state-transition model and a reward model. The state-transition model is shown as a decision tree, and the reward model associates the distinctive features of the goal boxes with the rewards to be found in each. (The rewards associated with states S_1 , S_2 , and S_3 are also part of the reward model, but here they are zero and are not shown.) A model-based agent can decide which way to turn at each state by using the model to simulate sequences of action choices to find a path yielding the highest return. In this case the return is the reward obtained from the outcome at the end of the path. Here, with a sufficiently accurate model, the rat would select L and then R to obtain reward of 4. Comparing the predicted returns of simulated paths is a simple form of planning, which can be done in a variety of ways as discussed in Chapter 8.

When the environment of a model-free agent changes the way it reacts to the agent's actions, the agent has to acquire new experience in the changed environment during which it can update its policy and/or value function. In the model-free strategy shown in Figure 14.5 (lower left), for example, if one of the goal boxes were to somehow shift to delivering a different reward, the rat would have to traverse the maze, possibly many times, to experience the new reward upon reaching that goal box, all the while updating either its policy or its action-value function (or both) based on this experience. The key point is that for a model-free agent to change the action its policy specifies for a state, or to change an action value associated with a state, it has to move to that state, act from it, possibly many times, and experience the consequences of its actions.

A model-based agent can accommodate changes in its environment without this kind of 'personal experience' with the states and actions affected by the change. A change in its model automatically (through planning) changes its policy. Planning can determine the consequences of changes in the environment that have never been linked together in the agent's own experience. For example, again referring to the maze task of Figure 14.5, imagine that a rat with a previously learned transition and reward model is placed directly in the goal box to the right of S_2 to find that the reward available there now has value 1 instead of 4. The rat's reward model will change even though the action choices required to find that goal box in the maze were not involved. The planning process will bring knowledge of the new reward to bear on maze running without the need for additional experience in the maze; in this case changing the policy to right turns at both S_1 and S_3 to obtain a return of 3.

Exactly this logic is the basis of *outcome-devaluation experiments* with animals. Results from these experiments provide insight into whether an animal has learned a habit or if its behavior is under goal-directed control. Outcome-devaluation experiments are like latent-learning experiments in that the reward changes from one stage to the next. After

an initial rewarded stage of learning, the reward value of an outcome is changed, including being shifted to zero or even to a negative value.

An early important experiment of this type was conducted by Adams and Dickinson (1981). They trained rats via instrumental conditioning until the rats energetically pressed a lever for sucrose pellets in a training chamber. The rats were then placed in the same chamber with the lever retracted and allowed non-contingent food, meaning that pellets were made available to them independently of their actions. After 15-minutes of this free-access to the pellets, rats in one group were injected with the nausea-inducing poison lithium chloride. This was repeated for three sessions, in the last of which none of the injected rats consumed any of the non-contingent pellets, indicating that the reward value of the pellets had been decreased—the pellets had been devalued. In the next stage taking place a day later, the rats were again placed in the chamber and given a session of extinction training, meaning that the response lever was back in place but disconnected from the pellet dispenser so that pressing it did not release pellets. The question was whether the rats that had the reward value of the pellets decreased would lever-press less than rats that did not have the reward value of the pellets decreased, even without experiencing the devalued reward as a result of lever-pressing. It turned out that the injected rats had significantly lower response rates than the non-injected rats *right from the start of the extinction trials*.

Adams and Dickinson concluded that the injected rats associated lever pressing with consequent nausea by means of a cognitive map linking lever pressing to pellets, and pellets to nausea. Hence, in the extinction trials, the rats “knew” that the consequences of pressing the lever would be something they did not want, and so they reduced their lever-pressing right from the start. The important point is that they reduced lever-pressing without ever having experienced lever-pressing directly followed by being sick: no lever was present when they were made sick. They seemed able to combine knowledge of the outcome of a behavioral choice (pressing the lever will be followed by getting a pellet) with the reward value of the outcome (pellets are to be avoided) and hence could alter their behavior accordingly. Not every psychologist agrees with this “cognitive” account of this kind of experiment, and it is not the only possible way to explain these results, but the model-based planning explanation is widely accepted.

Nothing prevents an agent from using both model-free and model-based algorithms, and there are good reasons for using both. We know from our own experience that with enough repetition, goal-directed behavior tends to turn into habitual behavior. Experiments show that this happens for rats too. Adams (1982) conducted an experiment to see if extended training would convert goal-directed behavior into habitual behavior. He did this by comparing the effect of outcome devaluation on rats that experienced different amounts of training. If extended training made the rats less sensitive to devaluation compared to rats that received less training, this would be evidence that extended training made the behavior more habitual. Adams’ experiment closely followed the Adams and Dickinson (1981) experiment just described. Simplifying a bit, rats in one group were trained until they made 100 rewarded lever-presses, and rats in the other group—the overtrained group—were trained until they made 500 rewarded lever-presses. After this training, the reward value of the pellets was decreased (using lithium chloride injections) for rats

in both groups. Then both groups of rats were given a session of extinction training. Adams' question was whether devaluation would effect the rate of lever-pressing for the overtrained rats less than it would for the non-overtrained rats, which would be evidence that extended training reduces sensitivity to outcome devaluation. It turned out that devaluation strongly decreased the lever-pressing rate of the non-overtrained rats. For the overtrained rats, in contrast, devaluation had little effect on their lever-pressing; in fact, if anything, it made it more vigorous. (The full experiment included control groups showing that the different amounts of training did not by themselves significantly effect lever-pressing rates after learning.) This result suggested that while the non-overtrained rats were acting in a goal-directed manner sensitive to their knowledge of the outcome of their actions, the overtrained rats had developed a lever-pressing habit.

Viewing this and other results like it from a computational perspective provides insight as to why one might expect animals to behave habitually in some circumstances, in a goal-directed way in others, and why they shift from one mode of control to another as they continue to learn. While animals undoubtedly use algorithms that do not exactly match those we have presented in this book, one can gain insight into animal behavior by considering the tradeoffs that various reinforcement learning algorithms imply. An idea developed by computational neuroscientists Daw, Niv, and Dayan (2005) is that animals use both model-free and model-based processes. Each process proposes an action, and the action chosen for execution is the one proposed by the process judged to be the more trustworthy of the two as determined by measures of confidence that are maintained throughout learning. Early in learning the planning process of a model-based system is more trustworthy because it chains together short-term predictions which can become accurate with less experience than long-term predictions of the model-free process. But with continued experience, the model-free process becomes more trustworthy because planning is prone to making mistakes due to model inaccuracies and short-cuts necessary to make planning feasible, such as various forms of "tree-pruning": the removal of unpromising search tree branches. According to this idea one would expect a shift from goal-directed behavior to habitual behavior as more experience accumulates. Other ideas have been proposed for how animals arbitrate between goal-directed and habitual control, and both behavioral and neuroscience research continues to examine this and related questions.

The distinction between model-free and model-based algorithms is proving to be useful for this research. One can examine the computational implications of these types of algorithms in abstract settings that expose basic advantages and limitations of each type. This serves both to suggest and to sharpen questions that guide the design of experiments necessary for increasing psychologists' understanding of habitual and goal-directed behavioral control.

14.7 Summary

Our goal in this chapter has been to discuss correspondences between reinforcement learning and the experimental study of animal learning in psychology. We emphasized at the outset that reinforcement learning as described in this book is not intended

to model details of animal behavior. It is an abstract computational framework that explores idealized situations from the perspective of artificial intelligence and engineering. But many of the basic reinforcement learning algorithms were inspired by psychological theories, and in some cases, these algorithms have contributed to the development of new animal learning models. This chapter described the most conspicuous of these correspondences.

The distinction in reinforcement learning between algorithms for prediction and algorithms for control parallels animal learning theory's distinction between classical, or Pavlovian, conditioning and instrumental conditioning. The key difference between instrumental and classical conditioning experiments is that in the former the reinforcing stimulus is contingent upon the animal's behavior, whereas in the latter it is not. Learning to predict via a TD algorithm corresponds to classical conditioning, and we described the *TD model of classical conditioning* as one instance in which reinforcement learning principles account for some details of animal learning behavior. This model generalizes the influential Rescorla–Wagner model by including the temporal dimension where events within individual trials influence learning, and it provides an account of second-order conditioning, where predictors of reinforcing stimuli become reinforcing themselves. It also is the basis of an influential view of the activity of dopamine neurons in the brain, something we take up in Chapter 15.

Learning by trial and error is at the base of the control aspect of reinforcement learning. We presented some details about Thorndike's experiments with cats and other animals that led to his *Law of Effect*, which we discussed here and in Chapter 1 (page 15). We pointed out that in reinforcement learning, exploration does not have to be limited to "blind groping"; trials can be generated by sophisticated methods using innate and previously learned knowledge as long as there is *some* exploration. We discussed the training method B. F. Skinner called *shaping* in which reward contingencies are progressively altered to train an animal to successively approximate a desired behavior. Shaping is not only indispensable for animal training, it is also an effective tool for training reinforcement learning agents. There is also a connection to the idea of an animal's motivational state, which influences what an animal will approach or avoid and what events are rewarding or punishing for the animal.

The reinforcement learning algorithms presented in this book include two basic mechanisms for addressing the problem of delayed reinforcement: eligibility traces and value functions learned via TD algorithms. Both mechanisms have antecedents in theories of animal learning. Eligibility traces are similar to stimulus traces of early theories, and value functions correspond to the role of secondary reinforcement in providing nearly immediate evaluative feedback.

The next correspondence the chapter addressed is that between reinforcement learning's environment models and what psychologists call *cognitive maps*. Experiments conducted in the mid 20th century purported to demonstrate the ability of animals to learn cognitive maps as alternatives to, or as additions to, state-action associations, and later use them to guide behavior, especially when the environment changes unexpectedly. Environment models in reinforcement learning are like cognitive maps in that they can be learned by supervised learning methods without relying on reward signals, and then they can be

used later to plan behavior.

Reinforcement learning's distinction between *model-free* and *model-based* algorithms corresponds to the distinction in psychology between *habitual* and *goal-directed* behavior. Model-free algorithms make decisions by accessing information that has been stored in a policy or an action-value function, whereas model-based methods select actions as the result of planning ahead using a model of the agent's environment. Outcome-devaluation experiments provide information about whether an animal's behavior is habitual or under goal-directed control. Reinforcement learning theory has helped clarify thinking about these issues.

Animal learning clearly informs reinforcement learning, but as a type of machine learning, reinforcement learning is directed toward designing and understanding effective learning algorithms, not toward replicating or explaining details of animal behavior. We focused on aspects of animal learning that relate in clear ways to methods for solving prediction and control problems, highlighting the fruitful two-way flow of ideas between reinforcement learning and psychology without venturing deeply into many of the behavioral details and controversies that have occupied the attention of animal learning researchers. Future development of reinforcement learning theory and algorithms will likely exploit links to many other features of animal learning as the computational utility of these features becomes better appreciated. We expect that a flow of ideas between reinforcement learning and psychology will continue to bear fruit for both disciplines.

Many connections between reinforcement learning and areas of psychology and other behavioral sciences are beyond the scope of this chapter. We largely omit discussing links to the psychology of decision making, which focuses on how actions are selected, or how decisions are made, *after* learning has taken place. We also do not discuss links to ecological and evolutionary aspects of behavior studied by ethologists and behavioral ecologists: how animals relate to one another and to their physical surroundings, and how their behavior contributes to evolutionary fitness. Optimization, MDPs, and dynamic programming figure prominently in these fields, and our emphasis on agent interaction with dynamic environments connects to the study of agent behavior in complex "ecologies." Multi-agent reinforcement learning, omitted in this book, has connections to social aspects of behavior. Despite the lack of treatment here, reinforcement learning should by no means be interpreted as dismissing evolutionary perspectives. Nothing about reinforcement learning implies a *tabula rasa* view of learning and behavior. Indeed, experience with engineering applications has highlighted the importance of building into reinforcement learning systems knowledge that is analogous to what evolution provides to animals.

Bibliographical and Historical Remarks

Ludvig, Bellemare, and Pearson (2011) and Shah (2012) review reinforcement learning in the contexts of psychology and neuroscience. These publications are useful companions to this chapter and the following chapter on reinforcement learning and neuroscience.

- 14.1** Dayan, Niv, Seymour, and Daw (2006) focused on interactions between classical and instrumental conditioning, particularly situations where classically-conditioned and instrumental responses are in conflict. They proposed a Q-learning framework for modeling aspects of this interaction. Modayil and Sutton (2014) used a mobile robot to demonstrate the effectiveness of a control method combining a fixed response with online prediction learning. Calling this *Pavlovian control*, they emphasized that it differs from the usual control methods of reinforcement learning, being based on predictively executing fixed responses and not on reward maximization. The electro-mechanical machine of Ross (1933) and especially the learning version of Walter's turtle (Walter, 1951) were very early illustrations of Pavlovian control.
- 14.2.1** Kamin (1968) first reported blocking, now commonly known as Kamin blocking, in classical conditioning. Moore and Schmajuk (2008) provide an excellent summary of the blocking phenomenon, the research it stimulated, and its lasting influence on animal learning theory. Gibbs, Cool, Land, Kehoe, and Gormezano (1991) describe second-order conditioning of the rabbit's nictitating membrane response and its relationship to conditioning with serial-compound stimuli. Finch and Culler (1934) reported obtaining fifth-order conditioning of a dog's foreleg withdrawal "when the *motivation* of the animal is maintained through the various orders."
- 14.2.2** The idea built into the Rescorla–Wagner model that learning occurs when animals are surprised is derived from Kamin (1969). Models of classical conditioning other than Rescorla and Wagner's include the models of Klopf (1988), Grossberg (1975), Mackintosh (1975), Moore and Stickney (1980), Pearce and Hall (1980), and Courville, Daw, and Touretzky (2006). Schmajuk (2008) review models of classical conditioning. Wagner (2008) provides a modern psychological perspective on the Rescorla–Wagner model and similar elemental theories of learning.
- 14.2.3** An early version of the TD model of classical conditioning appeared in Sutton and Barto (1981a), which also included the early model's prediction that temporal primacy overrides blocking, later shown by Kehoe, Schreurs, and Graham (1987) to occur in the rabbit nictitating membrane preparation. Sutton and Barto (1981a) contains the earliest recognition of the near identity between the Rescorla–Wagner model and the Least-Mean-Square (LMS), or Widrow-Hoff, learning rule (Widrow and Hoff, 1960). This early model was revised following Sutton's development of the TD algorithm (Sutton, 1984, 1988) and was first presented as the TD model in Sutton and Barto (1987) and more completely in Sutton and Barto (1990), upon which this section is largely based. Additional exploration

of the TD model and its possible neural implementation was conducted by Moore and colleagues (Moore, Desmond, Berthier, Blazis, Sutton, and Barto, 1986; Moore and Blazis, 1989; Moore, Choi, and Brunzell, 1998; Moore, Marks, Castagna, and Polewan, 2001). Klopff's (1988) drive-reinforcement theory of classical conditioning extends the TD model to address additional experimental details, such as the S-shape of acquisition curves. In some of these publications TD is taken to mean Time Derivative instead of Temporal Difference.

- 14.2.4** Ludvig, Sutton, and Kehoe (2012) evaluated the performance of the TD model in previously unexplored tasks involving classical conditioning and examined the influence of various stimulus representations, including the microstimulus representation that they introduced earlier (Ludvig, Sutton, and Kehoe, 2008). Earlier investigations of the influence of various stimulus representations and their possible neural implementations on response timing and topography in the context of the TD model are those of Moore and colleagues cited above. Although not in the context of the TD model, representations like the microstimulus representation of Ludvig et al. (2012) have been proposed and studied by Grossberg and Schmajuk (1989), Brown, Bullock, and Grossberg (1999), Buhusi and Schmajuk (1999), and Machado (1997). The figures on pages 353–355 are adapted from Sutton and Barto (1990).

- 14.3** Section 1.7 includes comments on the history of trial-and-error learning and the Law of Effect. The idea that Thorndike's cats might have been exploring according to an instinctual context-specific ordering over actions rather than by just selecting from a set of instinctual impulses was suggested by Peter Dayan (personal communication). Selfridge, Sutton, and Barto (1985) illustrated the effectiveness of shaping in a pole-balancing reinforcement learning task. Other examples of shaping in reinforcement learning are Gullapalli and Barto (1992), Mahadevan and Connell (1992), Mataric (1994), Dorigo and Colombette (1994), Saksida, Raymond, and Touretzky (1997), and Randløv and Alstrøm (1998). Ng (2003) and Ng, Harada, and Russell (1999) used the term shaping in a sense somewhat different from Skinner's, focussing on the problem of how to alter the reward signal without altering the set of optimal policies.

Dickinson and Balleine (2002) discuss the complexity of the interaction between learning and motivation. Wise (2004) provides an overview of reinforcement learning and its relation to motivation. Daw and Shohamy (2008) link motivation and learning to aspects of reinforcement learning theory. See also McClure, Daw, and Montague (2003), Niv, Joel, and Dayan (2006), Rangel, Camerer, and Montague (2008), and Dayan and Berridge (2014). McClure et al. (2003), Niv, Daw, and Dayan (2006), and Niv, Daw, Joel, and Dayan (2007) present theories of behavioral vigor related to the reinforcement learning framework.

- 14.4** Spence, Hull's student and collaborator at Yale, elaborated the role of higher-order reinforcement in addressing the problem of delayed reinforcement (Spence, 1947). Learning over very long delays, as in taste-aversion conditioning with

delays up to several hours, led to interference theories as alternatives to decaying-trace theories (e.g., Revusky and Garcia, 1970; Boakes and Costa, 2014). Other views of learning under delayed reinforcement invoke roles for awareness and working memory (e.g., Clark and Squire, 1998; Seo, Barracough, and Lee, 2007).

- 14.5** Thistlethwaite (1951) provides an extensive review of latent learning experiments up to the time of its publication. Ljung (1998) provides an overview of model learning, or system identification, techniques in engineering. Gopnik, Glymour, Sobel, Schulz, Kushnir, and Danks (2004) present a Bayesian theory about how children learn models.
- 14.6** Connections between habitual and goal-directed behavior and model-free and model-based reinforcement learning were first proposed by Daw, Niv, and Dayan (2005). The hypothetical maze task used to explain habitual and goal-directed behavioral control is based on the explanation of Niv, Joel, and Dayan (2006). Dolan and Dayan (2013) review four generations of experimental research related to this issue and discuss how it can move forward on the basis of reinforcement learning's model-free/model-based distinction. Dickinson (1980, 1985) and Dickinson and Balleine (2002) discuss experimental evidence related to this distinction. Donahoe and Burgos (2000) alternatively argue that model-free processes can account for the results of outcome-devaluation experiments. Dayan and Berridge (2014) argue that classical conditioning involves model-based processes. Rangel, Camerer, and Montague (2008) review many of the outstanding issues involving habitual, goal-directed, and Pavlovian modes of control.

Comments on Terminology— The traditional meaning of *reinforcement* in psychology is the strengthening of a pattern of behavior (by increasing either its intensity or frequency) as a result of an animal receiving a stimulus (or experiencing the omission of a stimulus) in an appropriate temporal relationship with another stimulus or with a response. Reinforcement produces changes that remain in future behavior. Sometimes in psychology reinforcement refers to the process of producing lasting changes in behavior, whether the changes strengthen or weaken a behavior pattern (Mackintosh, 1983). Letting reinforcement refer to weakening in addition to strengthening is at odds with the everyday meaning of reinforce, and its traditional use in psychology, but it is a useful extension that we have adopted here. In either case, a stimulus considered to be the cause of the behavioral change is called a *reinforcer*.

Psychologists do not generally use the specific phrase *reinforcement learning* as we do. Animal learning pioneers probably regarded reinforcement and learning as being synonymous, so it would be redundant to use both words. Our use of the phrase follows its use in computational and engineering research, influenced mostly by Minsky (1961). But the phrase is lately gaining currency in psychology and neuroscience, likely because strong parallels have surfaced between reinforcement learning algorithms and animal learning—parallels described in this chapter and the next.

According to common usage, a *reward* is an object or event that an animal will approach and work for. A reward may be given to an animal in recognition of its ‘good’

behavior, or given in order to make the animal's behavior 'better.' Similarly, a *penalty* is an object or event that the animal usually avoids and that is given as a consequence of 'bad' behavior, usually in order to change that behavior. *Primary reward* is reward due to machinery built into an animal's nervous system by evolution to improve its chances of survival and reproduction, for example, reward produced by the taste of nourishing food, sexual contact, successful escape, and many other stimuli and events that predicted reproductive success over the animal's ancestral history. As explained in Section 14.2.1, *higher-order reward* is reward delivered by stimuli that predict primary reward, either directly or indirectly by predicting other stimuli that predict primary reward. Reward is *secondary* if its rewarding quality is the result of directly predicting primary reward.

In this book we call R_t the 'reward signal at time t ' or sometimes just the 'reward at time t ', but we do not think of it as an object or event in the agent's environment. Because R_t is a number—not an object or an event—it is more like a reward signal in neuroscience, which is a signal internal to the brain, like the activity of neurons, that influences decision making and learning. This signal might be triggered when the animal perceives an attractive (or an aversive) object, but it can also be triggered by things that do not physically exist in the animal's external environment, such as memories, ideas, or hallucinations. Because our R_t can be positive, negative, or zero, it might be better to call a negative R_t a penalty, and an R_t equal to zero a neutral signal, but for simplicity we generally avoid these terms.

In reinforcement learning, the process that generates all the R_t s defines the problem the agent is trying to solve. The agent's objective is to keep the magnitude of R_t as large as possible over time. In this respect, R_t is like primary reward for an animal if we think of the problem the animal faces as the problem of obtaining as much primary reward as possible over its lifetime (and thereby, through the prospective "wisdom" of evolution, improve its chances of solving its real problem, which is to pass its genes on to future generations). However, as we suggest in Chapter 15, it is unlikely that there is a single "master" reward signal like R_t in an animal's brain.

Not all reinforcers are rewards or penalties. Sometimes reinforcement is not the result of an animal receiving a stimulus that evaluates its behavior by labeling the behavior good or bad. A behavior pattern can be reinforced by a stimulus that arrives to an animal no matter how the animal behaved. As described in Section 14.1, whether the delivery of reinforcer depends, or does not depend, on preceding behavior is the defining difference between instrumental, or operant, conditioning experiments and classical, or Pavlovian, conditioning experiments. Reinforcement is at work in both types of experiments, but only in the former is it feedback that evaluates past behavior. (Though it has often been pointed out that even when the reinforcing US in a classical conditioning experiment is not contingent on the subject's preceding behavior, its reinforcing value can be influenced by this behavior, an example being that a closed eye makes an air puff to the eye less aversive.)

The distinction between reward signals and reinforcement signals is a crucial point when we discuss neural correlates of these signals in the next chapter. Like a reward signal, for us, the reinforcement signal at any specific time is a positive or negative number, or zero. A reinforcement signal is the major factor directing changes a learning algorithm

makes in an agent's policy, value estimates, or environment models. The definition that makes the most sense to us is that a reinforcement signal at any time is a number that multiplies (possibly along with some constants) a vector to determine parameter updates in some learning algorithm.

For some algorithms, the reward signal alone is the critical multiplier in the parameter-update equation. For these algorithms the reinforcement signal is the same as the reward signal. But for most of the algorithms we discuss in this book, reinforcement signals include terms in addition to the reward signal, an example being a TD error $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$, which is the reinforcement signal for TD state-value learning (and analogous TD errors for action-value learning). In this reinforcement signal, R_{t+1} is the *primary reinforcement* contribution, and the temporal difference in predicted values, $\gamma V(S_{t+1}) - V(S_t)$ (or an analogous temporal difference for action values), is the *conditioned reinforcement* contribution. Thus, whenever $\gamma V(S_{t+1}) - V(S_t) = 0$, δ_t signals 'pure' primary reinforcement; and whenever $R_{t+1} = 0$, it signals 'pure' conditioned reinforcement, but it often signals a mixture of these. Note as we mentioned in Section 6.1, this δ_t is not available until time $t + 1$. We therefore think of δ_t as the reinforcement signal at time $t + 1$, which is fitting because it reinforces predictions and/or actions made earlier at step t .

A possible source of confusion is the terminology used by the famous psychologist B. F. Skinner and his followers. For Skinner, positive reinforcement occurs when the consequences of an animal's behavior increase the frequency of that behavior; punishment occurs when the behavior's consequences decrease that behavior's frequency. Negative reinforcement occurs when behavior leads to the removal of an aversive stimulus (that is, a stimulus the animal does not like), thereby increasing the frequency of that behavior. Negative punishment, on the other hand, occurs when behavior leads to the removal of an appetitive stimulus (that is, a stimulus the animal likes), thereby decreasing the frequency of that behavior. We find no critical need for these distinctions because our approach is more abstract than this, with both reward and reinforcement signals allowed to take on both positive and negative values. (But note especially that when our reinforcement signal is negative, it is not the same as Skinner's negative reinforcement.)

On the other hand, it has often been pointed out that using a single number as a reward or a penalty signal, depending only on its sign, is at odds with the fact that animals' appetitive and aversive systems have qualitatively different properties and involve different brain mechanisms. This points to a direction in which the reinforcement learning framework might be developed in the future to exploit computational advantages of separate appetitive and aversive systems, but for now we are passing over these possibilities.

Another discrepancy in terminology is how we use the word *action*. To many cognitive scientists, an action is purposeful in the sense of being the result of an animal's knowledge about the relationship between the behavior in question and the consequences of that behavior. An action is goal-directed and the result of a decision, in contrast to a response, which is triggered by a stimulus; the result of a reflex or a habit. We use the word action without differentiating among what others call actions, decisions, and responses. These are important distinctions, but for us they are encompassed by differences between

model-free and model-based reinforcement learning algorithms, which we discussed above in relation to habitual and goal-directed behavior in Section 14.6. Dickinson (1985) discusses the distinction between responses and actions.

A term used a lot in this book is *control*. What we mean by control is entirely different from what it means to animal learning psychologists. By control we mean that an agent influences its environment to bring about states or events that the agent prefers: the agent exerts control over its environment. This is the sense of control used by control engineers. In psychology, on the other hand, control typically means that an animal's behavior is influenced by—is controlled by—the stimuli the animal receives (stimulus control) or the reinforcement schedule it experiences. Here the environment is controlling the agent. Control in this sense is the basis of behavior modification therapy. Of course, both of these directions of control are at play when an agent interacts with its environment, but our focus is on the agent as controller; not the environment as controller. A view equivalent to ours, and perhaps more illuminating, is that the agent is actually controlling the input it receives from its environment (Powers, 1973). This is *not* what psychologists mean by stimulus control.

Sometimes reinforcement learning is understood to refer solely to learning policies directly from rewards (and penalties) without the involvement of value functions or environment models. This is what psychologists call stimulus-response, or S-R, learning. But for us, along with most of today's psychologists, reinforcement learning is much broader than this, including in addition to S-R learning, methods involving value functions, environment models, planning, and other processes that are commonly thought to belong to the more cognitive side of mental functioning.

Chapter 15

Neuroscience

Neuroscience is the multidisciplinary study of nervous systems: how they regulate bodily functions; control behavior; change over time as a result of development, learning, and aging; and how cellular and molecular mechanisms make these functions possible. One of the most exciting aspects of reinforcement learning is the mounting evidence from neuroscience that the nervous systems of humans and many other animals implement algorithms that correspond in striking ways to reinforcement learning algorithms. The main objective of this chapter is to explain these parallels and what they suggest about the neural basis of reward-related learning in animals.

The most remarkable point of contact between reinforcement learning and neuroscience involves dopamine, a chemical deeply involved in reward processing in the brains of mammals. Dopamine appears to convey temporal-difference (TD) errors to brain structures where learning and decision making take place. This parallel is expressed by the *reward prediction error hypothesis of dopamine neuron activity*, a hypothesis that resulted from the convergence of computational reinforcement learning and results of neuroscience experiments. In this chapter we discuss this hypothesis, the neuroscience findings that led to it, and why it is a significant contribution to understanding brain reward systems. We also discuss parallels between reinforcement learning and neuroscience that are less striking than this dopamine/TD-error parallel but that provide useful conceptual tools for thinking about reward-based learning in animals. Other elements of reinforcement learning have the potential to impact the study of nervous systems, but their connections to neuroscience are still relatively undeveloped. We discuss several of these evolving connections that we think will grow in importance over time.

As we outlined in the history section of this book's introductory chapter (Section 1.7), many aspects of reinforcement learning were influenced by neuroscience. A second objective of this chapter is to acquaint readers with ideas about brain function that have contributed to our approach to reinforcement learning. Some elements of reinforcement learning are easier to understand when seen in light of theories of brain function. This is particularly true for the idea of the eligibility trace, one of the basic mechanisms of reinforcement learning, that originated as a conjectured property of synapses, the structures by which nerve cells—neurons—communicate with one another.

In this chapter we do not delve very deeply into the enormous complexity of the neural systems underlying reward-based learning in animals: this chapter is too short, and we are not neuroscientists. We do not try to describe—or even to name—the very many brain structures and pathways, or any of the molecular mechanisms, believed to be involved in these processes. We also do not do justice to hypotheses and models that are alternatives to those that align so well with reinforcement learning. It should not be surprising that there are differing views among experts in the field. We can only provide a glimpse into this fascinating and developing story. We hope, though, that this chapter convinces you that a very fruitful channel has emerged connecting reinforcement learning and its theoretical underpinnings to the neuroscience of reward-based learning in animals.

Many excellent publications cover links between reinforcement learning and neuroscience, some of which we cite in this chapter's final section. Our treatment differs from most of these because we assume familiarity with reinforcement learning as presented in the earlier chapters of this book, but we do not assume knowledge of neuroscience. We begin with a brief introduction to the neuroscience concepts needed for a basic understanding of what is to follow.

15.1 Neuroscience Basics

Some basic information about nervous systems is helpful for following what we cover in this chapter. Terms that we refer to later are italicized. Skipping this section will not be a problem if you already have an elementary knowledge of neuroscience.

Neurons, the main components of nervous systems, are cells specialized for processing and transmitting information using electrical and chemical signals. They come in many forms, but a neuron typically has a cell body, *dendrites*, and a single *axon*. Dendrites are structures that branch from the cell body to receive input from other neurons (or to also receive external signals in the case of sensory neurons). A neuron's axon is a fiber that carries the neuron's output to other neurons (or to muscles or glands). A neuron's output consists of sequences of electrical pulses called *action potentials* that travel along the axon. Action potentials are also called *spikes*, and a neuron is said to *fire* when it generates a spike. In models of neural networks it is common to use real numbers to represent a neuron's *firing rate*, the average number of spikes per some unit of time.

A neuron's axon can branch widely so that the neuron's action potentials reach many targets. The branching structure of a neuron's axon is called the neuron's *axonal arbor*. Because the conduction of an action potential is an active process, not unlike the burning of a fuse, when an action potential reaches an axonal branch point it "lights up" action potentials on all of the outgoing branches (although propagation to a branch can sometimes fail). As a result, the activity of a neuron with a large axonal arbor can influence many target sites.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

15.3 The Reward Prediction Error Hypothesis

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

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

activity of dopamine neurons in mammals. The experiments that led to this hypothesis were performed in the 1980s and early 1990s in the laboratory of neuroscientist Wolfram Schultz. Section 15.4 describes these influential experiments, Section 15.6 explains how the results of these experiments align with TD errors, and the Bibliographical and Historical Remarks section at the end of this chapter includes a guide to the literature surrounding the development of this influential hypothesis.

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

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

In simulated trials with these assumptions about background firing rate and input representation, TD errors of the TD model are remarkably similar to dopamine neuron phasic activity. Previewing our description of details about these similarities in Section 15.4 below, the TD errors parallel the following features of dopamine neuron activity: 1) the phasic response of a dopamine neuron only occurs when a rewarding event is unpredicted; 2) early in learning, neutral cues that precede a reward do not cause substantial phasic dopamine responses, but with continued learning these cues gain predictive value and come to elicit phasic dopamine responses; 3) if an even earlier cue reliably precedes a cue that has already acquired predictive value, the phasic dopamine response shifts to the earlier cue, ceasing for the later cue; and 4) if after learning, the predicted rewarding event is omitted, a dopamine neuron's response decreases below its baseline level shortly after the expected time of the rewarding event.

Although not every dopamine neuron monitored in the experiments of Schultz and

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

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

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

15.4 Dopamine

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

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

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

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

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

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

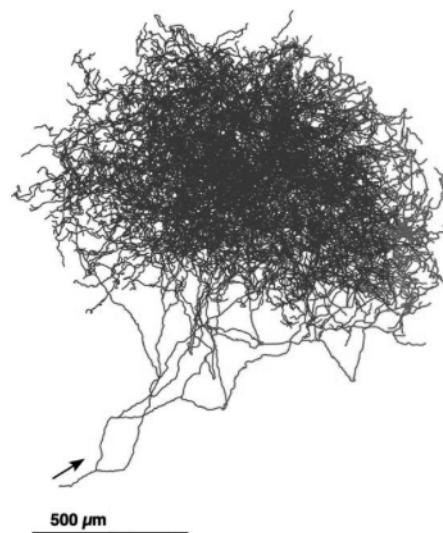
Additional evidence for the reinforcing function of dopamine comes from optogenetic experiments with fruit flies, except in these animals dopamine’s effect is the opposite of its effect in mammals: optically triggered bursts of dopamine neuron activity act just like electric foot shock in reinforcing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

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

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

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

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



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

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

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

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

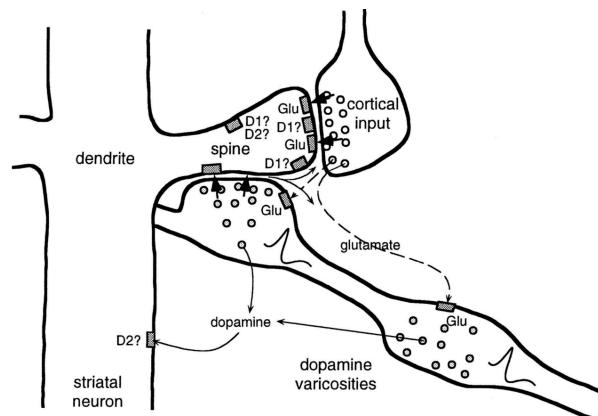


Figure 15.1: Spine of a striatal neuron showing input from both cortical and dopamine neurons. Axons of cortical neurons influence striatal neurons via corticostriatal synapses releasing the neurotransmitter glutamate at the tips of spines covering the dendrites of striatal neurons. An axon of a VTA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

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

15.5 Experimental Support for the Reward Prediction Error Hypothesis

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

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

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

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

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

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

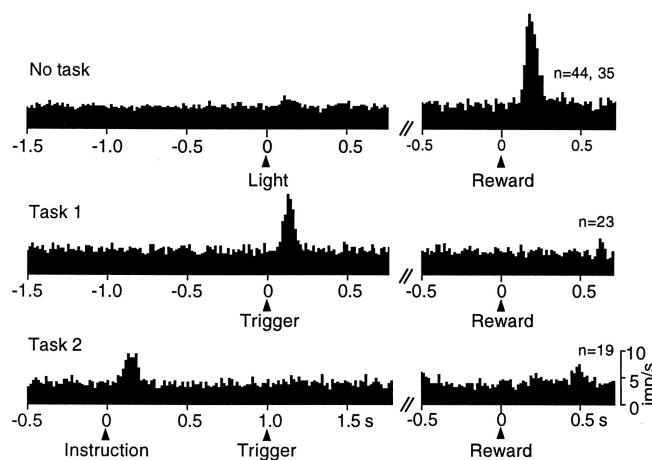


Figure 15.2: The response of dopamine neurons shifts from initial responses to primary reward to earlier predictive stimuli. These are plots of the number of action potentials produced by monitored dopamine neurons within small time intervals, averaged over all the monitored dopamine neurons (ranging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

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

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

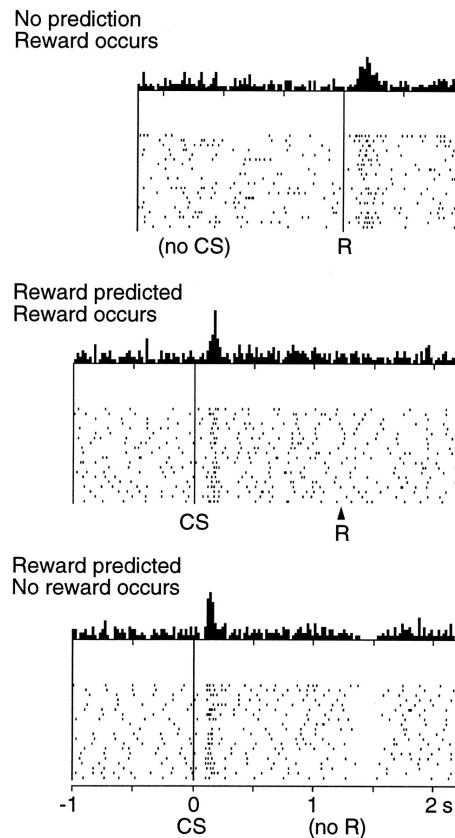


Figure 15.3: The response of dopamine neurons drops below baseline shortly after the time when an expected reward fails to occur. Top: dopamine neurons are activated by the unpredicted delivery of a drop of apple juice. Middle: dopamine neurons respond to a conditioned stimulus (CS) that predicts reward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

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

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

15.6 TD Error/Dopamine Correspondence

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

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

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

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

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

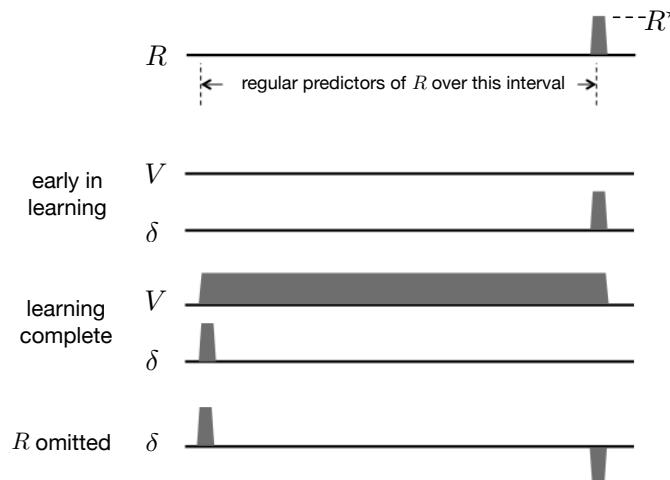


Figure 15.4: The behavior of the TD error δ during TD learning is consistent with features of the phasic activation of dopamine neurons. (Here δ is the TD error *available* at time t , i.e., δ_{t-1}). *Top:* a sequence of states, shown as an interval of regular predictors, is followed by a non-zero reward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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

15.7 Neural Actor–Critic

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

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

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

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

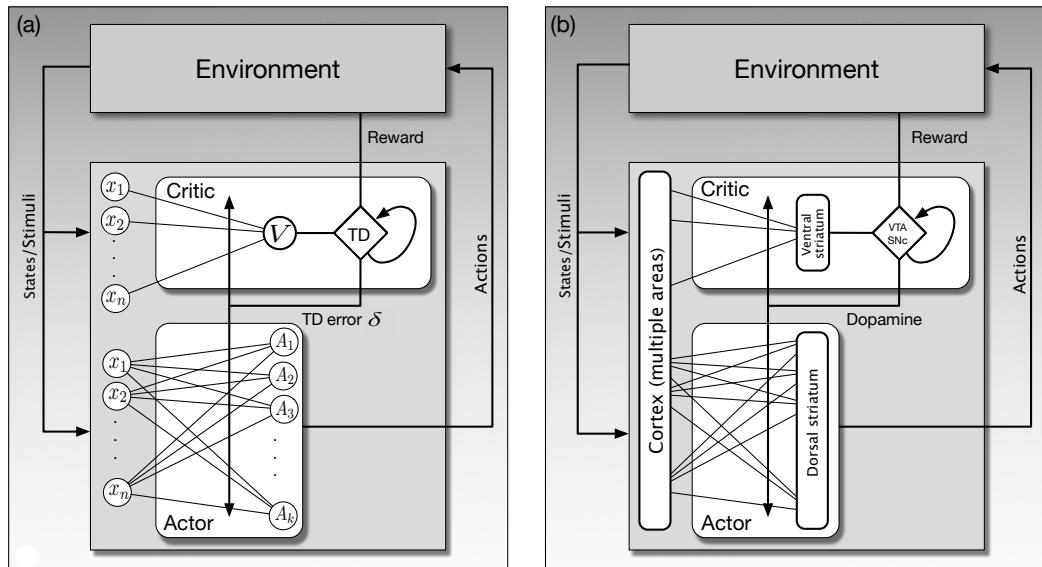


Figure 15.5: Actor–critic ANN and a hypothetical neural implementation. a) Actor–critic algorithm as an ANN. The actor adjusts a policy based on the TD error δ it receives from the critic; the critic adjusts state-value parameters using the same δ . The critic produces a TD error from the reward signal,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

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

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

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

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

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

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

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

15.8 Actor and Critic Learning Rules

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

15.9 Hedonistic Neurons

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

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

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

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

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

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

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

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

15.10 Collective Reinforcement Learning

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

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

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

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

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

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

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

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

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

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

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

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

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

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

15.11 Model-based Methods in the Brain

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

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

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

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

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

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

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

substrates of these systems?

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

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

15.12 Addiction

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

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

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

15.13 Summary

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

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

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

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

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

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

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

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

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

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

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

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

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

Bibliographical and Historical Remarks

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

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

- 15.1** There are many good expositions of basic neuroscience. Kandel, Schwartz, Jessell, Siegelbaum, and Hudspeth (2013) is an authoritative and very comprehensive source.
- 15.2** Berridge and Kringelbach (2008) reviewed the neural basis of reward and pleasure, pointing out that reward processing has many dimensions and involves many neural systems. Space prevents discussion of the influential research of Berridge and Robinson (1998), who distinguish between the hedonic impact of a stimulus, which they call “liking,” and the motivational effect, which they call “wanting.” Hare, O’Doherty, Camerer, Schultz, and Rangel (2008) examined the neural basis of value-related signals from an economic perspective, distinguishing between goal values, decision values, and prediction errors. Decision value is goal value minus action cost. See also Rangel, Camerer, and Montague (2008), Rangel and Hare (2010), and Peters and Büchel (2010).
- 15.3** The reward prediction error hypothesis of dopamine neuron activity is most prominently discussed by Schultz, Dayan, and Montague (1997). The hypothesis was first explicitly put forward by Montague, Dayan, and Sejnowski (1996). As they stated the hypothesis, it referred to reward prediction errors (RPEs) but not specifically to TD errors; however, their development of the hypothesis made it clear that they were referring to TD errors. The earliest recognition of the TD-error/dopamine connection of which we are aware is that of Montague, Dayan, Nowlan, Pouget, and Sejnowski (1993), who proposed a TD-error-modulated Hebbian learning rule motivated by results on dopamine signaling from Schultz’s group. The connection was also pointed out in an abstract by Quartz, Dayan, Montague, and Sejnowski (1992). Montague and Sejnowski (1994) emphasized the importance of prediction in the brain and outlined how predictive Hebbian learning modulated by TD errors could be implemented via a diffuse neuromodulatory system, such as the dopamine system. Friston, Tononi, Reke, Sporns,

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

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

- 15.5** Schultz's 1998 survey article is a good entrée into the very extensive literature on reward predicting signaling of dopamine neurons. Berns, McClure, Pagnoni, and Montague (2001), Breiter, Aharon, Kahneman, Dale, and Shizgal (2001), Pagnoni, Zink, Montague, and Berns (2002), and O'Doherty, Dayan, Friston, Critchley, and Dolan (2003) described functional brain imaging studies supporting the existence of signals like TD errors in the human brain.
- 15.6** This section roughly follows Barto (1995a) in explaining how TD errors mimic the main results from Schultz's group on the phasic responses of dopamine neurons.
- 15.7** This section is largely based on Takahashi, Schoenbaum, and Niv (2008) and Niv (2009). To the best of our knowledge, Barto (1995a) and Houk, Adams, and Barto (1995) first speculated about possible implementations of actor–critic algorithms in the basal ganglia. On the basis of functional magnetic resonance imaging of human subjects while engaged in instrumental conditioning, O'Doherty, Dayan, Schultz, Deichmann, Friston, and Dolan (2004) suggested that the actor and the critic are most likely located respectively in the dorsal and ventral striatum. Gershman, Moustafa, and Ludvig (2014) focused on how time is represented in reinforcement learning models of the basal ganglia, discussing evidence for, and implications of, various computational approaches to time representation.
- The hypothetical neural implementation of the actor–critic architecture described in this section includes very little detail about known basal ganglia anatomy and physiology. In addition to the more detailed hypothesis of Houk, Adams, and Barto (1995), a number of other hypotheses include more specific connections to anatomy and physiology and are claimed to explain additional data. These include hypotheses proposed by Suri and Schultz (1998, 1999), Brown, Bullock, and Grossberg (1999), Contreras-Vidal and Schultz (1999), Suri, Bargas, and Arbib (2001), O'Reilly and Frank (2006), and O'Reilly, Frank, Hazy, and Watz (2007). Joel, Niv, and Ruppin (2002) critically evaluated the anatomical plausibility of several of these models and present an alternative intended to accommodate some neglected features of basal ganglionic circuitry.
- 15.8** The actor learning rule discussed here is more complicated than the one in the early actor–critic network of Barto et al. (1983). Actor-unit eligibility traces in that network were traces of just $A_t \times \mathbf{x}(S_t)$ instead of the full $(A_t - \pi(A_t|S_t, \theta))\mathbf{x}(S_t)$. That work did not benefit from the policy-gradient theory presented in Chapter 13 or the contributions of Williams (1986, 1992), who showed how an ANN of Bernoulli-logistic units could implement a policy-gradient method.
- Reynolds and Wickens (2002) proposed a three-factor rule for synaptic plasticity in the corticostriatal pathway in which dopamine modulates changes in corticostriatal synaptic efficacy. They discussed the experimental support for this kind of learning rule and its possible molecular basis. The definitive demonstration of spike-timing-dependent plasticity (STDP) is attributed to Markram, Lübke, Frotscher, and Sakmann (1997), with evidence from earlier experiments

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

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

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

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

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

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

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

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

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

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

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

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

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

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

- 15.11** Yin and Knowlton (2006) reviewed findings from outcome-devaluation experiments with rodents supporting the view that habitual and goal-directed behavior (as psychologists use the phrase) are respectively most associated with processing in the dorsolateral striatum (DLS) and the dorsomedial striatum (DMS). Results of functional imaging experiments with human subjects in the outcome-devaluation setting by Valentin, Dickinson, and O'Doherty (2007) suggest that the orbitofrontal cortex (OFC) is an important component of goal-directed choice. Single unit recordings in monkeys by Padoa-Schioppa and Assad (2006) support the role of the OFC in encoding values guiding choice behavior. Rangel, Camerer, and Montague (2008) and Rangel and Hare (2010) reviewed findings from the perspective of neuroeconomics about how the brain makes goal-directed decisions. Pezzulo, van der Meer, Lansink, and Pennartz (2014) reviewed the neuroscience of internally generated sequences and presented a model of how these mechanisms might be components of model-based planning. Daw and Shohamy (2008) proposed that while dopamine signaling connects well to habitual, or model-free, behavior, other processes are involved in goal-directed, or model-based, behavior. Data from experiments by Bromberg-Martin, Matsumoto, Hong, and Hikosaka (2010) indicate that dopamine signals contain information pertinent to both habitual and goal-directed behavior. Doll, Simon, and Daw (2012) argued that there may not a clear separation in the brain between mechanisms that subserve habitual and goal-directed learning and choice.
- 15.12** Keiflin and Janak (2015) reviewed connections between TD errors and addiction. Nutt, Lingford-Hughes, Erritzoe, and Stokes (2015) critically evaluated the hypothesis that addiction is due to a disorder of the dopamine system. Montague, Dolan, Friston, and Dayan (2012) outlined the goals and early efforts in the field of computational psychiatry, and Adams, Huys, and Roiser (2015) reviewed more recent progress.

Chapter 16

Applications and Case Studies

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

16.1 TD-Gammon

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

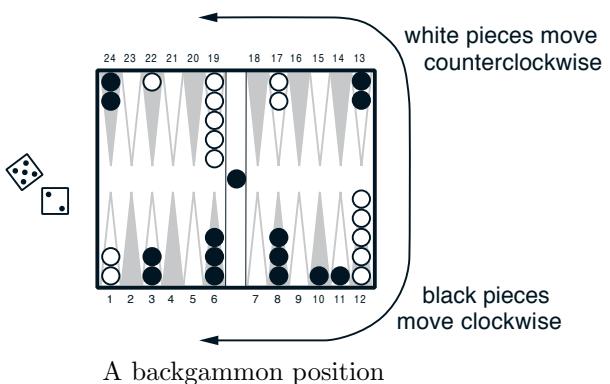
Backgammon is a major game in the sense that it is played throughout the world, with numerous tournaments and regular world championship matches. It is in part a game of chance, and it is a popular vehicle for waging significant sums of money. There are probably more professional backgammon players than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

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

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

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

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



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

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

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

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

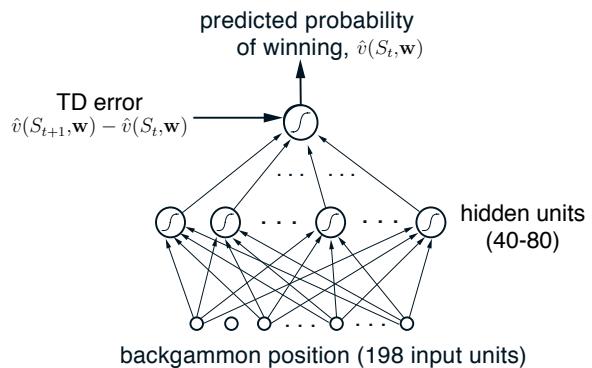


Figure 16.1: The TD-Gammon ANN

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Table 16.1: Summary of TD-Gammon Results

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

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

16.2 Samuel’s Checkers Player

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

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

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

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

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

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

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

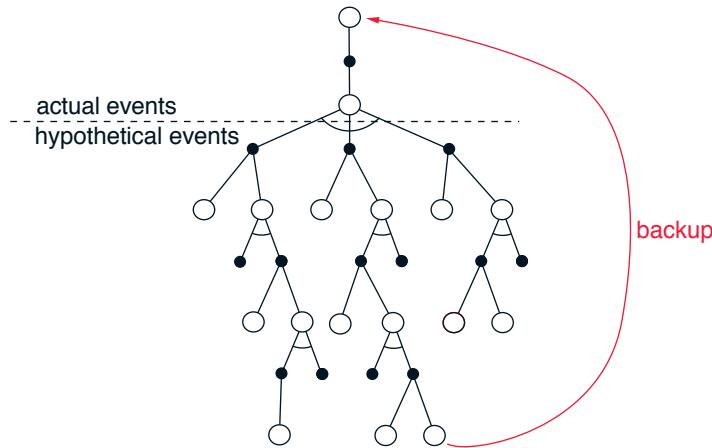


Figure 16.2: The backup diagram for Samuel’s checkers player.

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

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

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

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

16.3 Watson’s Daily-Double Wagering

IBM WATSON¹ is the system developed by a team of IBM researchers to play the popular TV quiz show *Jeopardy!*.² It gained fame in 2011 by winning first prize in an exhibition match against human champions. Although the main technical achievement demonstrated by WATSON was its ability to quickly and accurately answer natural language questions over broad areas of general knowledge, its winning *Jeopardy!* performance also relied on sophisticated decision-making strategies for critical parts of the game. Tesauro, Gondek, Lechner, Fan, and Prager (2012, 2013) adapted Tesauro’s TD-Gammon system described above to create the strategy used by WATSON in “Daily-Double” (DD) wagering in its celebrated winning performance against human champions. These authors report that the effectiveness of this wagering strategy went well beyond what human players are able to do in live game play, and that it, along with other advanced strategies, was an important contributor to WATSON’s impressive winning performance. Here we focus only on DD wagering because it is the component of WATSON that owes the most to reinforcement learning.

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

¹Registered trademark of IBM Corp.

²Registered trademark of Jeopardy Productions Inc.

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

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

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

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

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

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

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

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

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

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

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

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

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

16.4 Optimizing Memory Control

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

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

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

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

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

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

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

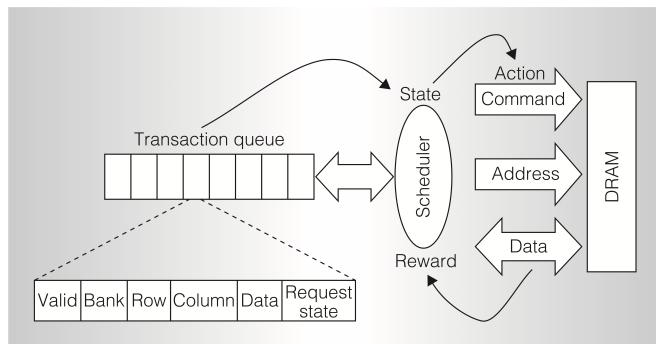


Figure 16.3: High-level view of the reinforcement learning DRAM controller. The scheduler is the reinforcement learning agent. Its environment is represented by features of the transaction queue, and its actions are commands to the DRAM system. ©2009 IEEE. Reprinted, with permission, from J. F. Martínez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

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

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

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

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

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

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

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

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

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

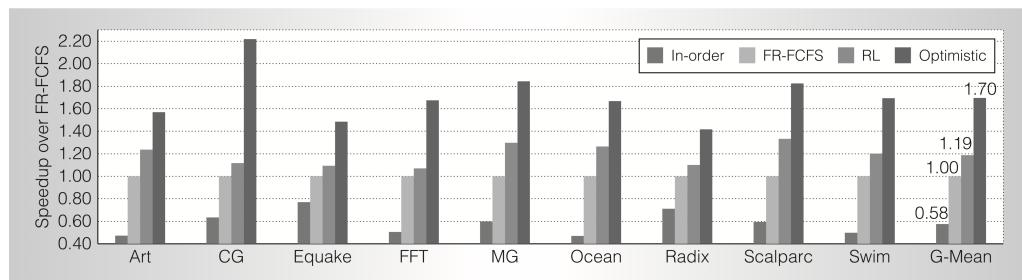


Figure 16.4: Performances of four controllers over a suite of 9 simulated benchmark applications. The controllers are: the simplest ‘in-order’ controller, FR-FCFS, the learning controller RL, and the unrealizable Optimistic controller which ignores all timing and resource constraints to provide a performance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

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

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

16.5 Human-level Video Game Play

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

A team of researchers at Google DeepMind developed an impressive demonstration that a deep multi-layer ANN can automate the feature design process (Mnih et al., 2013, 2015). Multi-layer ANNs have been used for function approximation in reinforcement learning ever since the 1986 popularization of the backpropagation algorithm as a method for learning internal representations (Rumelhart, Hinton, and Williams, 1986; see Section 9.6). Striking results have been obtained by coupling reinforcement learning with backpropagation. The results obtained by Tesauro and colleagues with TD-Gammon and WATSON discussed above are notable examples. These and other applications benefited from the ability of multi-layer ANNs to learn task-relevant features. However, in all the examples of which we are aware, the most impressive demonstrations required the network’s input to be represented in terms of specialized features handcrafted for the given problem. This is vividly apparent in the TD-Gammon results. TD-Gammon 0.0, whose network input was essentially a “raw” representation of the backgammon board, meaning that it involved very little knowledge of backgammon, learned to play approximately as well as

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

Mnih et al. developed a reinforcement learning agent called *deep Q-network* (DQN) that combined Q-learning with a *deep convolutional ANN*, a many-layered, or deep, ANN specialized for processing spatial arrays of data such as images. We describe deep convolutional ANNs in Section 9.6. By the time of Mnih et al.’s work with DQN, deep ANNs, including deep convolutional ANNs, had produced impressive results in many applications, but they had not been widely used in reinforcement learning.

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

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

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

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

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

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

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

An essential point here is that these preprocessing steps were exactly the same for all 46

games. No game-specific prior knowledge was involved beyond the general understanding that it should still be possible to learn good policies with this reduced dimension and that stacking adjacent frames should help with the partial observability of some of the games. Because no game-specific prior knowledge beyond this minimal amount was used in preprocessing the image frames, we can think of the $84 \times 84 \times 4$ input vectors as being “raw” input to DQN.

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

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

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

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

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

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

The gradient in (16.3) was computed by backpropagation. Imagining again that there

was a separate network for each action, for the update at time step t , backpropagation was applied only to the network corresponding to A_t . Mnih et al. took advantage of techniques shown to improve the basic backpropagation algorithm when applied to large networks. They used a *mini-batch method* that updated weights only after accumulating gradient information over a small batch of images (here after 32 images). This yielded smoother sample gradients compared to the usual procedure that updates weights after each action. They also used a gradient-ascent algorithm called RMSProp (Tieleman and Hinton, 2012) that accelerates learning by adjusting the step-size parameter for each weight based on a running average of the magnitudes of recent gradients for that weight.

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

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

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

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

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

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

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

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

16.6 Mastering the Game of Go

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

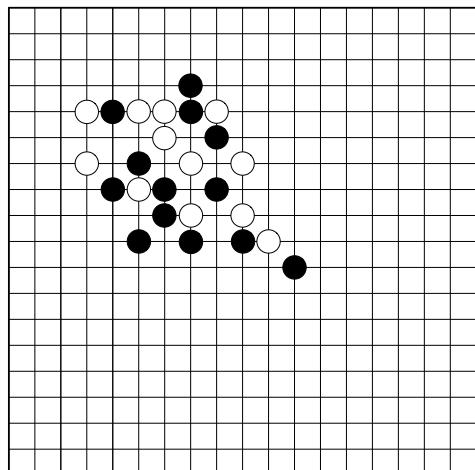
A team at DeepMind (Silver et al., 2016) developed the program *AlphaGo* that broke

this barrier by combining deep ANNs (Section 9.6), supervised learning, Monte Carlo tree search (MCTS, Section 8.11), and reinforcement learning. By the time of Silver et al.’s 2016 publication, *AlphaGo* had been shown to be decisively stronger than other current Go programs, and it had defeated the European Go champion Fan Hui 5 games to 0. These were the first victories of a Go program over a professional human Go player without handicap in full Go games. Shortly thereafter, a similar version of *AlphaGo* won stunning victories over the 18-time world champion Lee Sedol, winning 4 out of a 5 games in a challenge match, making worldwide headline news. Artificial intelligence researchers thought that it would be many more years, perhaps decades, before a program reached this level of play.

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

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

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



A Go board configuration

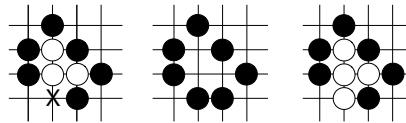


Figure 16.5: Go capturing rule. Left: the three white stones are not surrounded because point X is unoccupied. Middle: if black places a stone on X, the three white stones are captured and removed from the board. Right: if white places a stone on point X first, the capture is blocked.

appeal for thousands of years.

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

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

next execution might be just this new root node, or it might include descendants of this node left over from MCTS's previous execution. The remainder of the tree is discarded.

16.6.1 AlphaGo

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

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

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

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

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

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

policy network. This was a deep convolutional ANN with the same structure as the SL policy network. It was initialized with the final weights of the SL policy network that were learned via supervised learning, and then policy-gradient reinforcement learning was used to improve upon the SL policy. In the second stage of training the value network, the team used Monte Carlo policy evaluation on data obtained from a large number of simulated self-play games with moves selected by the RL policy network.

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

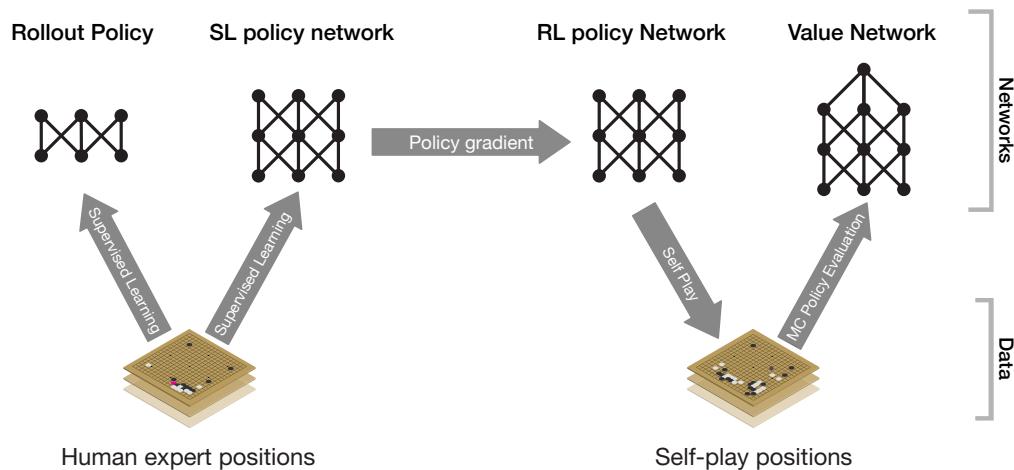


Figure 16.6: *AlphaGo* pipeline. Adapted with permission from Macmillan Publishers Ltd: *Nature*, vol. 529(7587), p. 485, ©2016.

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

Training the SL policy network took approximately 3 weeks using a distributed implementation of stochastic gradient ascent on 50 processors. The network achieved 57% accuracy, where the best accuracy achieved by other groups at the time of publication

was 44.4%. Training the RL policy network was done by policy gradient reinforcement learning over simulated games between the RL policy network's current policy and opponents using policies randomly selected from policies produced by earlier iterations of the learning algorithm. Playing against a randomly selected collection of opponents prevented overfitting to the current policy. The reward signal was +1 if the current policy won, -1 if it lost, and zero otherwise. These games directly pitted the two policies against one another without involving MCTS. By simulating many games in parallel on 50 processors, the DeepMind team trained the RL policy network on a million games in a single day. In testing the final RL policy, they found that it won more than 80% of games played against the SL policy, and it won 85% of games played against a Go program using MCTS that simulated 100,000 games per move.

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

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

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

Several methods worked together to produce *AlphaGo*'s impressive playing skill. The DeepMind team evaluated different versions of *AlphaGo* in order to assess the contributions

made by these various components. The parameter η in (16.4) controlled the mixing of game state evaluations produced by the value network and by rollouts. With $\eta = 0$, *AlphaGo* used just the value network without rollouts, and with $\eta = 1$, evaluation relied just on rollouts. They found that *AlphaGo* using just the value network played better than the rollout-only *AlphaGo*, and in fact played better than the strongest of all other Go programs existing at the time. The best play resulted from setting $\eta = 0.5$, indicating that combining the value network with rollouts was particularly important to *AlphaGo*'s success. These evaluation methods complemented one another: the value network evaluated the high-performance RL policy that was too slow to be used in live play, while rollouts using the weaker but much faster rollout policy were able to add precision to the value network's evaluations for specific states that occurred during games.

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

16.6.2 AlphaGo Zero

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

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

Instead of selecting self-play actions according to the probabilities \mathbf{p} , however, *AlphaGo Zero* used these probabilities, together with the network's value output, to direct each execution of MCTS, which returned new move probabilities, shown in Figure 16.7 as the policies π_i . These policies benefitted from the many simulations that MCTS conducted

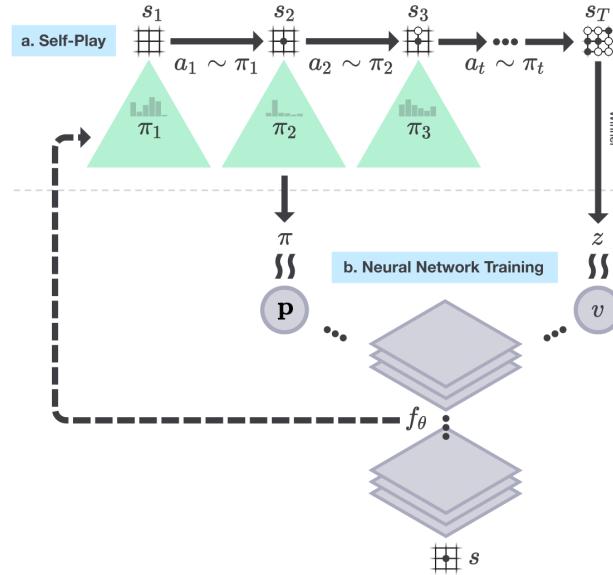


Figure 16.7: *AlphaGo Zero* self-play reinforcement learning. a) The program played many games against itself, one shown here as a sequence of board positions s_i , $i = 1, 2, \dots, T$, with moves a_i , $i = 1, 2, \dots, T$, and winner z . Each move a_i was determined by action probabilities π_i returned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

each time it executed. The result was that the policy actually followed by *AlphaGo Zero* was an improvement over the policy given by the network's outputs p . Silver et al. (2017a) wrote that “MCTS may therefore be viewed as a powerful *policy improvement* operator.”

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

is why features describing past board positions and the color feature were needed.

The network was “two-headed,” meaning that after a number of initial layers, the network split into two separate “heads” of additional layers that separately fed into two sets of output units. In this case, one head fed 362 output units producing $19^2 + 1$ move probabilities \mathbf{p} , one for each possible stone placement plus pass; the other head fed just one output unit producing the scalar v , an estimate of the probability that the current player will win from the current board position. The network before the split consisted of 41 convolutional layers, each followed by batch normalization, and with skip connections added to implement residual learning by pairs of layers (see Section 9.6). Overall, move probabilities and values were computed by 43 and 44 layers respectively.

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

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

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

different from how humans play. In fact, *AlphaGo Zero* discovered, and came to prefer, some novel variations of classical move sequences.

Final tests of *AlphaGo Zero*'s algorithm were conducted with a version having a larger ANN and trained over 29 million self-play games, which took about 40 days, again starting with random weights. This version achieved an Elo rating of 5,185. The team pitted this version of *AlphaGo Zero* against a program called *AlphaGo Master*, the strongest program at the time, that was identical to *AlphaGo Zero* but, like *AlphaGo*, used human data and features. *AlphaGo Master*'s Elo rating was 4,858, and it had defeated the strongest human professional players 60 to 0 in online games. In a 100 game match, *AlphaGo Zero* with the larger network and more extensive learning defeated *AlphaGo Master* 89 games to 11, thus providing a convincing demonstration of the problem-solving power of *AlphaGo Zero*'s algorithm.

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

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

16.7 Personalized Web Services

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

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

maximize the *click-through rate* (CTR), which is the ratio of the total number of clicks all users make on a webpage to the total number of visits to the page. Their contextual bandit algorithm improved over a standard non-associative bandit algorithm by 12.5%.

Theocharous, Thomas, and Ghavamzadeh (2015) argued that better results are possible by formulating personalized recommendation as a Markov decision problem (MDP) with the objective of maximizing the total number of clicks users make over repeated visits to a website. Policies derived from the contextual bandit formulation are greedy in the sense that they do not take long-term effects of actions into account. These policies effectively treat each visit to a website as if it were made by a new visitor uniformly sampled from the population of the website's visitors. By not using the fact that many users repeatedly visit the same websites, greedy policies do not take advantage of possibilities provided by long-term interactions with individual users.

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

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

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

returns have high variance.

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

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

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

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

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

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

Figure 16.8 illustrates how these metrics differ. Each circle represents a user visit to the site; black circles are visits at which the user clicks. Each row represents visits by a particular user. By not distinguishing between visitors, the CTR for these sequences is 0.35, whereas the LTV is 1.5. Because LTV is larger than CTR to the extent that individual users revisit the site, it is an indicator of how successful a policy is in encouraging users to engage in extended interactions with the site.

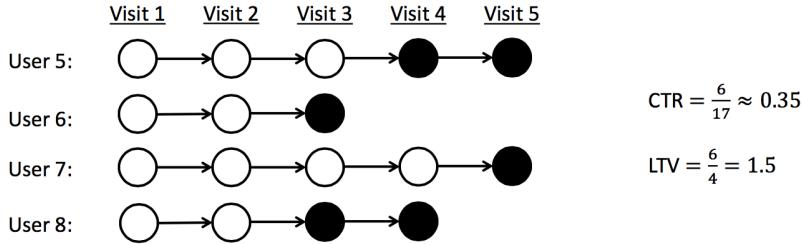


Figure 16.8: Click through rate (CTR) versus life-time value (LTV). Each circle represents a user visit; black circles are visits at which the user clicks. Adapted from Theocharous et al. (2015).

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

16.8 Thermal Soaring

Birds and gliders take advantage of upward air currents—thermals—to gain altitude in order to maintain flight while expending little, or no, energy. Thermal soaring, as this behavior is called, is a complex skill requiring responding to subtle environmental cues to increase altitude by exploiting a rising column of air for as long as possible. Reddy, Celani, Sejnowski, and Vergassola (2016) used reinforcement learning to investigate thermal soaring policies that are effective in the strong atmospheric turbulence usually accompanying rising air currents. Their primary goal was to provide insight into the cues birds sense and how they use them to achieve their impressive thermal soaring performance, but the results also contribute to technology relevant to autonomous gliders. Reinforcement learning had previously been applied to the problem of navigating efficiently

to the vicinity of a thermal updraft (Woodbury, Dunn, and Valasek, 2014) but not to the more challenging problem of soaring within the turbulence of the updraft itself.

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

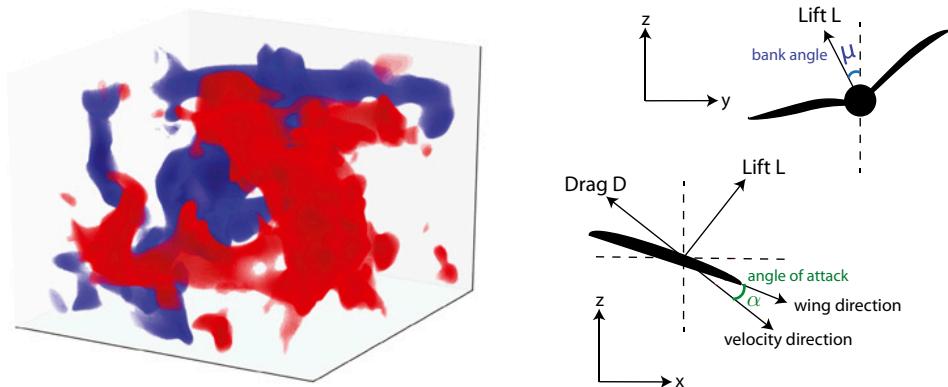


Figure 16.9: Thermal soaring model: Left: snapshot of the vertical velocity field of the simulated cube of air: in red (blue) is a region of large upward (downward) flow. Right: diagram of powerless flight showing bank angle μ and angle of attack α . Adapted with permission From PNAS vol. 113(22), p.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

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

Because a goal of their study was to try to determine what minimal set of sensory cues are necessary for effective soaring, both to shed light on the cues birds might use for soaring and to minimize the sensing complexity required for automated glider soaring,

the authors tried various sets of signals as input to the reinforcement learning agent. They started by using state aggregation (Section 9.3) of a four-dimensional state space with dimensions giving local vertical wind speed, local vertical wind acceleration, torque depending on the difference between the vertical wind velocities at the left and right wing tips, and the local temperature. Each dimension was discretized into three bins: positive high, negative high, and small. Results, described below, showed that only two of these dimensions were critical for effective soaring behavior.

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

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

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

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

Each learning episode took place with the agent controlling simulated flight in an independently generated period of simulated turbulent air currents. Each episode lasted 2.5 minutes simulated with a 1 second time step. Learning effectively converged after a few hundred episodes. The left panel of Figure 16.10 shows a sample trajectory before learning where the agent selects actions randomly. Starting at the top of the volume shown, the glider’s trajectory is in the direction indicated by the arrow and quickly loses altitude. Figure 16.10’s right panel is a trajectory after learning. The glider starts at the same place (here appearing at the bottom of the volume) and gains altitude by spiraling

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

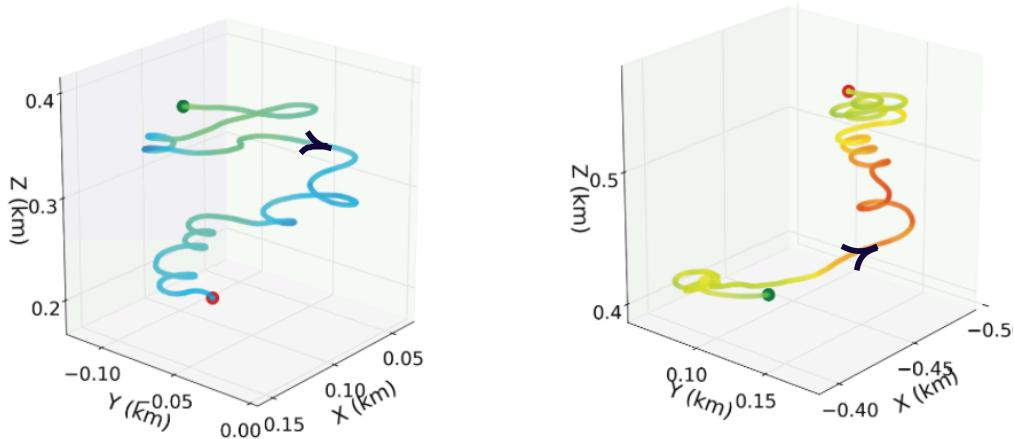


Figure 16.10: Sample thermal soaring trajectories, with arrows showing the direction of flight from the same starting point (note that the altitude scales are shifted). Left: before learning: the agent selects actions randomly and the glider descends. Right: after learning: the glider gains altitude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

within the rising column of air. Although Reddy et al. found that performance varied widely over different simulated periods of air flow, the number of times the glider touched the ground consistently decreased to nearly zero as learning progressed.

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

Due to the fact that soaring in different levels of turbulence requires different policies, training was done in conditions ranging from weak to strong turbulence. In strong turbulence the rapidly changing wind and glider velocities allowed less time for the controller to react. This reduced the amount of control possible compared to what was possible for maneuvering when fluctuations were weak. Reddy et al. examined the policies Sarsa learned under these different conditions. Common to policies learned in all regimes were these features: when sensing negative wind acceleration, bank sharply in the direction of the wing with the higher lift; when sensing large positive wind acceleration and no torque, do nothing. However, different levels of turbulence led to policy differences.

Policies learned in strong turbulence were more conservative in that they preferred small bank angles, whereas in weak turbulence, the best action was to turn as much as possible by banking sharply. Systematic study of the bank angles preferred by the policies learned under the different conditions led the authors to suggest that by detecting when vertical wind acceleration crosses a certain threshold the controller can adjust its policy to cope with different turbulence regimes.

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

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

Chapter 17

Frontiers

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

17.1 General Value Functions and Auxiliary Tasks

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

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

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

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

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

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

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

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

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

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

17.2 Temporal Abstraction via Options

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

17.3 Observations and State

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

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

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

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

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

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

Second, we can recover the idea of state as used in this book from the sequence of observations and actions. Let us use the word *history*, and the notation H_t , for an initial portion of the trajectory up to an observation: $H_t \doteq A_0, O_1, \dots, A_{t-1}, O_t$. The history represents the most that we can know about the past without looking outside of the data stream (because the history is the whole past data stream). Of course, the history grows with t and can become large and unwieldy. The idea of state is that of some compact summary of the history that is as useful as the actual history for predicting the future. Let us be clear about exactly what this means. To be a summary of the history, the state must be a function of history, $S_t = f(H_t)$, and to be as useful for predicting the future as the whole history, it must have what is known as the *Markov property*. Formally, this is a property of the function f . A function f has the Markov property if and only if any two histories h and h' that are mapped by f to the same state ($f(h) = f(h')$) also have the same probabilities for their next observation,

$$f(h) = f(h') \Rightarrow \Pr\{O_{t+1} = o | H_t = h, A_t = a\} = \Pr\{O_{t+1} = o | H_t = h', A_t = a\}, \quad (17.5)$$

for all $o \in \mathcal{O}$ and $a \in \mathcal{A}$. If f is Markov, then $S_t = f(H_t)$ is a state as we have used the term in this book. Let us henceforth call it a *Markov state* to distinguish it from states that are summaries of the history but fall short of the Markov property (which we will consider shortly).

A Markov state is a good basis for predicting the next observation (17.5) but, more importantly, it is also a good basis for predicting or controlling *anything*. For example, let a *test* be any specific sequence of alternating actions and observations that might occur in the future. For example, a three-step test is denoted $\tau = a_1 o_1 a_2 o_2 a_3 o_3$. The probability of this test given a specific history h is defined as

$$p(\tau|h) \doteq \Pr\{O_{t+1} = o_1, O_{t+2} = o_2, O_{t+3} = o_3 | H_t = h, A_t = a_1, A_{t+1} = a_2, A_{t+2} = a_3\}. \quad (17.6)$$

If f is Markov and h and h' are any two histories that map to the same state under f , then for any test τ of any length, its probabilities given the two histories must also be the same:

$$f(h) = f(h') \Rightarrow p(\tau|h) = p(\tau|h'). \quad (17.7)$$

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

The third step in extending reinforcement learning to partial observability is to deal with certain computational considerations. In particular, we want the state to be a *compact* summary of the history. For example, the identity function completely satisfies the conditions for a Markov f , but would nevertheless be of little use because the corresponding state $S_t = H_t$ would grow with time and become unwieldy, as mentioned earlier, but more fundamentally because it would never recur; the agent would never

encounter the same state twice (in a continuing task) and thus could never benefit from a tabular learning method. We want our states to be compact as well as Markov. There is a similar issue regarding how state is obtained and updated. We don't really want a function f that takes whole histories. Instead, for computational reasons we prefer to obtain the same effect as f with an incremental, recursive update that computes S_{t+1} from S_t , incorporating the next increment of data, A_t and O_{t+1} :

$$S_{t+1} \doteq u(S_t, A_t, O_{t+1}), \text{ for all } t \geq 0, \quad (17.8)$$

with the first state S_0 given. The function u is called the *state-update* function. For example, if f were the identity ($S_t = H_t$), then u would merely extend S_t by appending A_t and O_{t+1} to it. Given f , it is always possible to construct a corresponding u , but it may not be computationally convenient and, as in the identity example, it may not produce a compact state. The state-update function is a central part of any agent architecture that handles partial observability. It must be efficiently computable, as no actions or predictions can be made until the state is available. An overall diagram of such an agent architecture is given in Figure 17.1.

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

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

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

$$u(\mathbf{s}, a, o)[i] \doteq \frac{\sum_{x=1}^d \mathbf{s}[x] p(i, o|x, a)}{\sum_{x=1}^d \sum_{x'=1}^d \mathbf{s}[x] p(x', o|x, a)}, \quad (17.9)$$

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

Another example of Markov states is provided by *Predictive State Representations*, or *PSRs*. PSRs address the weakness of the POMDP approach that the semantics of its agent state S_t are grounded in the environment state, X_t , which is never observed and

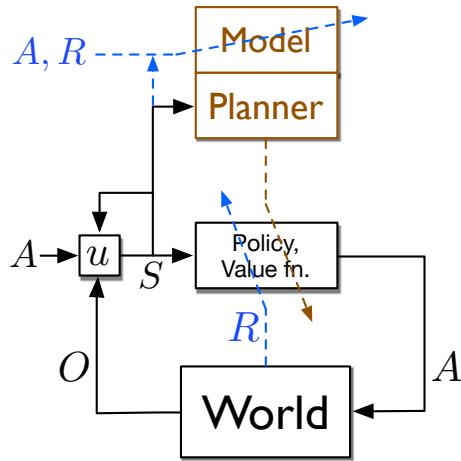


Figure 17.1: A conceptual agent architecture including a model, a planner, and a state-update function. The world in this case receives actions A and emits observations O . The observations and a copy of the action are used by the state-update function u to produce the new state. The new state is input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

thus is difficult to learn about. In PSRs and related approaches, the semantics of the agent state is instead grounded in predictions about future observations and actions, which are readily observable. In PSRs, a Markov state is defined as a d -vector of the probabilities of d specially chosen “core” tests as defined above (17.6). The vector is then updated by a state-update function u that is analogous to Bayes rule, but with a semantics grounded in observable data, which arguably makes it easier to learn. This approach has been extended in many ways, including end-tests, compositional tests, powerful “spectral” methods, and closed-loop and temporally abstract tests learned by TD methods. Some of the best theoretical developments are for systems known as *Observable Operator Models* (OOMs) and Sequential Systems (Thon, 2017).

The fourth and final step in our brief outline of how to handle partial observability in reinforcement learning is to re-introduce approximation. As discussed in the introduction to Part II, to approach artificial intelligence ambitiously one must embrace approximation. This is just as true for states as it is for value functions. We must accept and work with an approximate notion of state. The approximate state will play the same role in our algorithms as before, so we continue to use the notation S_t for the state used by the agent, even though it may not be Markov.

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

What happens when the Markov property (17.5) is only approximately satisfied? Unfortunately, long-term prediction performance can degrade dramatically when the one-step predictions defining the Markov property become even slightly inaccurate. Longer-term tests, GVF_s, and state-update functions may all approximate poorly. The short-term and long-term approximation objectives are just different, and there are no useful theoretical guarantees at present.

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

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

17.4 Designing Reward Signals

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

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

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

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

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

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

$$\hat{v}(s, \mathbf{w}) \doteq \mathbf{w}^\top \mathbf{x}(s) + v_0(s), \quad (17.10)$$

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

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

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

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

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

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

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

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

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

17.5 Remaining Issues

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

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

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

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

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

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

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

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

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

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

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

17.6 The Future of Artificial Intelligence

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

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

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

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

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

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

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

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

Some of these warnings are particularly relevant to reinforcement learning. Because reinforcement learning is based on optimization, it inherits the pluses and minuses of all optimization methods. On the minus side is the problem of devising objective functions, or reward signals in the case of reinforcement learning, so that optimization produces

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

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

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

beneficial as it is today in improving the quality, efficiency, and cost-effectiveness of processes on which we have come to rely. One of the most pressing areas for future reinforcement learning research is to adapt and extend methods developed in control engineering with the goal of making it acceptably safe to fully embed reinforcement learning agents into physical environments.

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

Bibliographical and Historical Remarks

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

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

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

- 17.2** The formalization of temporally abstract courses of action as options was introduced by Sutton, Precup, and Singh (1999), building on prior work by Parr (1998) and Sutton (1995a), and on classical work on Semi-MDPs (e.g., see Puterman, 1994). Precup’s (2000) PhD thesis developed option ideas fully. An important limitation of these early works is that they did not treat the off-policy case with function approximation. Intra-option learning in general requires off-policy learning, which could not be done reliably with function approximation at that time. Although now we have a variety of stable off-policy learning methods using function approximation, their combination with option ideas had not been significantly explored at the time of publication of this book. Barto and Mahadevan (2003) and Hengst (2012) review the options formalism and other approaches to temporal abstraction.

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

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

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

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

were developed by Tanner (2006; Sutton and Tanner, 2005) and then extended to options (Sutton, Rafols, and Koop, 2006).

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

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

Skinner's shaping should not be confused with the "potential-based shaping" technique introduced by Ng, Harada, and Russell (1999). Their technique has been shown by Wiewiora (2003) to be equivalent to the simpler idea of providing an initial approximation to the value function, as in (17.10).

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

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

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

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

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

References

- Abbeel, P., Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the 21st International Conference on Machine Learning*. ACM, New York.
- Abramson, B. (1990). Expected-outcome: A general model of static evaluation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(2):182–193.
- Adams, C. D. (1982). Variations in the sensitivity of instrumental responding to reinforcer devaluation. *The Quarterly Journal of Experimental Psychology*, 34(2):77–98.
- Adams, C. D., Dickinson, A. (1981). Instrumental responding following reinforcer devaluation. *The Quarterly Journal of Experimental Psychology*, 33(2):109–121.
- Adams, R. A., Huys, Q. J. M., Roiser, J. P. (2015). Computational Psychiatry: towards a mathematically informed understanding of mental illness. *Journal of Neurology, Neurosurgery & Psychiatry*. doi:10.1136/jnnp-2015-310737
- Agrawal, R. (1995). Sample mean based index policies with $O(\log n)$ regret for the multi-armed bandit problem. *Advances in Applied Probability*, 27(4):1054–1078.
- Agre, P. E. (1988). *The Dynamic Structure of Everyday Life*. Ph.D. thesis, Massachusetts Institute of Technology, Cambridge MA. AI-TR 1085, MIT Artificial Intelligence Laboratory.
- Agre, P. E., Chapman, D. (1990). What are plans for? *Robotics and Autonomous Systems*, 6(1-2):17–34.
- Aizerman, M. A., Braverman, E. I., Rozonoer, L. I. (1964). Probability problem of pattern recognition learning and potential functions method. *Avtomat. i Telemekh*, 25(9):1307–1323.
- Albus, J. S. (1971). A theory of cerebellar function. *Mathematical Biosciences*, 10(1-2):25–61.
- Albus, J. S. (1981). *Brain, Behavior, and Robotics*. Byte Books, Peterborough, NH.
- Aleksandrov, V. M., Sysoev, V. I., Shemeneva, V. V. (1968). Stochastic optimization of systems. *Izv. Akad. Nauk SSSR, Tekh. Kibernetika*:14–19.
- Amari, S. I. (1998). Natural gradient works efficiently in learning. *Neural Computation*, 10(2):251–276.
- An, P. C. E. (1991). *An Improved Multi-dimensional CMAC Neural network: Receptive Field Function and Placement*. Ph.D. thesis, University of New Hampshire, Durham.
- An, P. C. E., Miller, W. T., Parks, P. C. (1991). Design improvements in associative memories for cerebellar model articulation controllers (CMAC). *Artificial Neural Networks*, pp. 1207–1210, Elsevier North-Holland. <http://www.incompleteideas.net/papers/AnMillerParks1991.pdf>
- Anderson, C. W. (1986). *Learning and Problem Solving with Multilayer Connectionist Systems*. Ph.D. thesis, University of Massachusetts, Amherst.
- Anderson, C. W. (1987). Strategy learning with multilayer connectionist representations. In *Proceedings of the 4th International Workshop on Machine Learning*, pp. 103–114. Morgan Kaufmann.

- Anderson, C. W. (1989). Learning to control an inverted pendulum using neural networks. *IEEE Control Systems Magazine*, 9(3):31–37.
- Anderson, J. A., Silverstein, J. W., Ritz, S. A., Jones, R. S. (1977). Distinctive features, categorical perception, and probability learning: Some applications of a neural model. *Psychological Review*, 84(5):413–451.
- Andreae, J. H. (1963). STELLA, A scheme for a learning machine. In *Proceedings of the 2nd IFAC Congress, Basle*, pp. 497–502. Butterworths, London.
- Andreae, J. H. (1969a). A learning machine with monologue. *International Journal of Man-Machine Studies*, 1(1):1–20.
- Andreae, J. H. (1969b). Learning machines—a unified view. In A. R. Meetham and R. A. Hudson (Eds.), *Encyclopedia of Information, Linguistics, and Control*, pp. 261–270. Pergamon, Oxford.
- Andreae, J. H. (1977). *Thinking with the Teachable Machine*. Academic Press, London.
- Andreae, J. H. (2017a). A model of how the brain learns: A short introduction to multiple context associative learning (MCAL) and the PP system. Unpublished report.
- Andreae, J. H. (2017b). Working memory for the associative learning of language. Unpublished report.
- Arthur, W. B. (1991). Designing economic agents that act like human agents: A behavioral approach to bounded rationality. *The American Economic Review*, 81(2):353–359.
- Asadi, K., Allen, C., Roderick, M., Mohamed, A. R., Konidaris, G., Littman, M. (2017). Mean actor critic. ArXiv:1709.00503.
- Atkeson, C. G. (1992). Memory-based approaches to approximating continuous functions. In *Sante Fe Institute Studies in the Sciences of Complexity*, Proceedings Vol. 12, pp. 521–521. Addison-Wesley.
- Atkeson, C. G., Moore, A. W., Schaal, S. (1997). Locally weighted learning. *Artificial Intelligence Review*, 11:11–73.
- Auer, P., Cesa-Bianchi, N., Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256.
- Bacon, P. L., Harb, J., Precup, D. (2017). The Option-Critic Architecture. In *Proceedings of the Association for the Advancement of Artificial Intelligence*, pp. 1726–1734.
- Baird, L. C. (1995). Residual algorithms: Reinforcement learning with function approximation. In *Proceedings of the 12th International Conference on Machine Learning (ICML 1995)*, pp. 30–37. Morgan Kaufmann.
- Baird, L. C. (1999). *Reinforcement Learning through Gradient Descent*. Ph.D. thesis, Carnegie Mellon University, Pittsburgh PA.
- Baird, L. C., Klopff, A. H. (1993). Reinforcement learning with high-dimensional, continuous actions. Wright Laboratory, Wright-Patterson Air Force Base, Tech. Rep. WL-TR-93-1147.
- Baird, L., Moore, A. W. (1999). Gradient descent for general reinforcement learning. In *Advances in Neural Information Processing Systems 11 (NIPS 1998)*, pp. 968–974. MIT Press, Cambridge MA.
- Baldassarre, G., Mirolli, M. (Eds.) (2013). *Intrinsically Motivated Learning in Natural and Artificial Systems*. Springer-Verlag, Berlin Heidelberg.
- Balke, A., Pearl, J. (1994). Counterfactual probabilities: Computational methods, bounds and applications. In *Proceedings of the Tenth International Conference on Uncertainty in Artificial Intelligence (UAI-1994)*, pp. 46–54. Morgan Kaufmann.
- Baras, D., Meir, R. (2007). Reinforcement learning, spike-time-dependent plasticity, and the BCM rule. *Neural Computation*, 19(8):2245–2279.

- Barnard, E. (1993). Temporal-difference methods and Markov models. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(2):357–365.
- Barreto, A. S., Precup, D., Pineau, J. (2011). Reinforcement learning using kernel-based stochastic factorization. In *Advances in Neural Information Processing Systems 24 (NIPS 2011)*, pp. 720–728. Curran Associates, Inc.
- Bartlett, P. L., Baxter, J. (1999). Hebbian synaptic modifications in spiking neurons that learn. Technical report, Research School of Information Sciences and Engineering, Australian National University.
- Bartlett, P. L., Baxter, J. (2000). A biologically plausible and locally optimal learning algorithm for spiking neurons. Rapport technique, Australian National University.
- Barto, A. G. (1985). Learning by statistical cooperation of self-interested neuron-like computing elements. *Human Neurobiology*, 4(4):229–256.
- Barto, A. G. (1986). Game-theoretic cooperativity in networks of self-interested units. In J. S. Denker (Ed.), *Neural Networks for Computing*, pp. 41–46. American Institute of Physics, New York.
- Barto, A. G. (1989). From chemotaxis to cooperativity: Abstract exercises in neuronal learning strategies. In R. Durbin, R. Maill and G. Mitchison (Eds.), *The Computing Neuron*, pp. 73–98. Addison-Wesley, Reading, MA.
- Barto, A. G. (1990). Connectionist learning for control: An overview. In T. Miller, R. S. Sutton, and P. J. Werbos (Eds.), *Neural Networks for Control*, pp. 5–58. MIT Press, Cambridge, MA.
- Barto, A. G. (1991). Some learning tasks from a control perspective. In L. Nadel and D. L. Stein (Eds.), *1990 Lectures in Complex Systems*, pp. 195–223. Addison-Wesley, Redwood City, CA.
- Barto, A. G. (1992). Reinforcement learning and adaptive critic methods. In D. A. White and D. A. Sofge (Eds.), *Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches*, pp. 469–491. Van Nostrand Reinhold, New York.
- Barto, A. G. (1995a). Adaptive critics and the basal ganglia. In J. C. Houk, J. L. Davis, and D. G. Beiser (Eds.), *Models of Information Processing in the Basal Ganglia*, pp. 215–232. MIT Press, Cambridge, MA.
- Barto, A. G. (1995b). Reinforcement learning. In M. A. Arbib (Ed.), *Handbook of Brain Theory and Neural Networks*, pp. 804–809. MIT Press, Cambridge, MA.
- Barto, A. G. (2011). Adaptive real-time dynamic programming. In C. Sammut and G. I Webb (Eds.), *Encyclopedia of Machine Learning*, pp. 19–22. Springer Science and Business Media.
- Barto, A. G. (2013). Intrinsic motivation and reinforcement learning. In G. Baldassarre and M. Mirolli (Eds.), *Intrinsically Motivated Learning in Natural and Artificial Systems*, pp. 17–47. Springer-Verlag, Berlin Heidelberg.
- Barto, A. G., Anandan, P. (1985). Pattern recognizing stochastic learning automata. *IEEE Transactions on Systems, Man, and Cybernetics*, 15(3):360–375.
- Barto, A. G., Anderson, C. W. (1985). Structural learning in connectionist systems. In *Program of the Seventh Annual Conference of the Cognitive Science Society*, pp. 43–54.
- Barto, A. G., Anderson, C. W., Sutton, R. S. (1982). Synthesis of nonlinear control surfaces by a layered associative search network. *Biological Cybernetics*, 43(3):175–185.
- Barto, A. G., Bradtke, S. J., Singh, S. P. (1991). Real-time learning and control using asynchronous dynamic programming. Technical Report 91-57. Department of Computer and Information Science, University of Massachusetts, Amherst.
- Barto, A. G., Bradtke, S. J., Singh, S. P. (1995). Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72(1-2):81–138.

- Barto, A. G., Duff, M. (1994). Monte Carlo matrix inversion and reinforcement learning. In *Advances in Neural Information Processing Systems 6 (NIPS 1993)*, pp. 687–694. Morgan Kaufmann, San Francisco.
- Barto, A. G., Jordan, M. I. (1987). Gradient following without back-propagation in layered networks. In M. Caudill and C. Butler (Eds.), *Proceedings of the IEEE First Annual Conference on Neural Networks*, pp. II629–II636. SOS Printing, San Diego.
- Barto, A. G., Mahadevan, S. (2003). Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(4):341–379.
- Barto, A. G., Singh, S. P. (1990). On the computational economics of reinforcement learning. In *Connectionist Models: Proceedings of the 1990 Summer School*. Morgan Kaufmann.
- Barto, A. G., Sutton, R. S. (1981a). Goal seeking components for adaptive intelligence: An initial assessment. Technical Report AFWAL-TR-81-1070. Air Force Wright Aeronautical Laboratories/Avionics Laboratory, Wright-Patterson AFB, OH.
- Barto, A. G., Sutton, R. S. (1981b). Landmark learning: An illustration of associative search. *Biological Cybernetics*, 42(1):1–8.
- Barto, A. G., Sutton, R. S. (1982). Simulation of anticipatory responses in classical conditioning by a neuron-like adaptive element. *Behavioural Brain Research*, 4(3):221–235.
- Barto, A. G., Sutton, R. S., Anderson, C. W. (1983). Neuronlike elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 13(5):835–846. Reprinted in J. A. Anderson and E. Rosenfeld (Eds.), *Neurocomputing: Foundations of Research*, pp. 535–549. MIT Press, Cambridge, MA, 1988.
- Barto, A. G., Sutton, R. S., Brouwer, P. S. (1981). Associative search network: A reinforcement learning associative memory. *Biological Cybernetics*, 40(3):201–211.
- Barto, A. G., Sutton, R. S., Watkins, C. J. C. H. (1990). Learning and sequential decision making. In M. Gabriel and J. Moore (Eds.), *Learning and Computational Neuroscience: Foundations of Adaptive Networks*, pp. 539–602. MIT Press, Cambridge, MA.
- Baxter, J., Bartlett, P. L. (2001). Infinite-horizon policy-gradient estimation. *Journal of Artificial Intelligence Research*, 15:319–350.
- Baxter, J., Bartlett, P. L., Weaver, L. (2001). Experiments with infinite-horizon, policy-gradient estimation. *Journal of Artificial Intelligence Research*, 15:351–381.
- Bellemare, M. G., Dabney, W., Munos, R. (2017). A distributional perspective on reinforcement learning. ArXiv:1707.06887.
- Bellemare, M. G., Naddaf, Y., Veness, J., Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279.
- Bellemare, M. G., Veness, J., Bowling, M. (2012). Investigating contingency awareness using Atari 2600 games. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence (AAAI-12)*, pp. 864–871. AAAI Press, Menlo Park, CA.
- Bellman, R. E. (1956). A problem in the sequential design of experiments. *Sankhya*, 16:221–229.
- Bellman, R. E. (1957a). *Dynamic Programming*. Princeton University Press, Princeton.
- Bellman, R. E. (1957b). A Markov decision process. *Journal of Mathematics and Mechanics*, 6(5):679–684.
- Bellman, R. E., Dreyfus, S. E. (1959). Functional approximations and dynamic programming. *Mathematical Tables and Other Aids to Computation*, 13:247–251.
- Bellman, R. E., Kalaba, R., Kotkin, B. (1973). Polynomial approximation—A new computational technique in dynamic programming: Allocation processes. *Mathematical Computation*, 17:155–161.

- Bengio, Y. (2009). Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–27.
- Bengio, Y., Courville, A. C., Vincent, P. (2012). Unsupervised feature learning and deep learning: A review and new perspectives. *CoRR* 1, ArXiv:1206.5538.
- Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517.
- Berg, H. C. (1975). Chemotaxis in bacteria. *Annual review of biophysics and bioengineering*, 4(1):119–136.
- Berns, G. S., McClure, S. M., Pagnoni, G., Montague, P. R. (2001). Predictability modulates human brain response to reward. *The journal of neuroscience*, 21(8):2793–2798.
- Berridge, K. C., Kringelbach, M. L. (2008). Affective neuroscience of pleasure: reward in humans and animals. *Psychopharmacology*, 199(3):457–480.
- Berridge, K. C., Robinson, T. E. (1998). What is the role of dopamine in reward: hedonic impact, reward learning, or incentive salience? *Brain Research Reviews*, 28(3):309–369.
- Berry, D. A., Fristedt, B. (1985). *Bandit Problems*. Chapman and Hall, London.
- Bertsekas, D. P. (1982). Distributed dynamic programming. *IEEE Transactions on Automatic Control*, 27(3):610–616.
- Bertsekas, D. P. (1983). Distributed asynchronous computation of fixed points. *Mathematical Programming*, 27(1):107–120.
- Bertsekas, D. P. (1987). *Dynamic Programming: Deterministic and Stochastic Models*. Prentice-Hall, Englewood Cliffs, NJ.
- Bertsekas, D. P. (2005). *Dynamic Programming and Optimal Control, Volume 1*, third edition. Athena Scientific, Belmont, MA.
- Bertsekas, D. P. (2012). *Dynamic Programming and Optimal Control, Volume 2: Approximate Dynamic Programming*, fourth edition. Athena Scientific, Belmont, MA.
- Bertsekas, D. P. (2013). Rollout algorithms for discrete optimization: A survey. In *Handbook of Combinatorial Optimization*, pp. 2989–3013. Springer, New York.
- Bertsekas, D. P., Tsitsiklis, J. N. (1989). *Parallel and Distributed Computation: Numerical Methods*. Prentice-Hall, Englewood Cliffs, NJ.
- Bertsekas, D. P., Tsitsiklis, J. N. (1996). *Neuro-Dynamic Programming*. Athena Scientific, Belmont, MA.
- Bertsekas, D. P., Tsitsiklis, J. N., Wu, C. (1997). Rollout algorithms for combinatorial optimization. *Journal of Heuristics*, 3(3):245–262.
- Bertsekas, D. P., Yu, H. (2009). Projected equation methods for approximate solution of large linear systems. *Journal of Computational and Applied Mathematics*, 227(1):27–50.
- Bhat, N., Farias, V., Moallemi, C. C. (2012). Non-parametric approximate dynamic programming via the kernel method. In *Advances in Neural Information Processing Systems 25 (NIPS 2012)*, pp. 386–394. Curran Associates, Inc.
- Bhatnagar, S., Sutton, R., Ghavamzadeh, M., Lee, M. (2009). Natural actor–critic algorithms. *Automatica*, 45(11).
- Biermann, A. W., Fairfield, J. R. C., Beres, T. R. (1982). Signature table systems and learning. *IEEE Transactions on Systems, Man, and Cybernetics*, 12(5):635–648.
- Bishop, C. M. (1995). *Neural Networks for Pattern Recognition*. Clarendon, Oxford.
- Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer Science + Business Media New York LLC.
- Blodgett, H. C. (1929). The effect of the introduction of reward upon the maze performance of rats. *University of California Publications in Psychology*, 4:113–134.

- Boakes, R. A., Costa, D. S. J. (2014). Temporal contiguity in associative learning: Interference and decay from an historical perspective. *Journal of Experimental Psychology: Animal Learning and Cognition*, 40(4):381–400.
- Booker, L. B. (1982). *Intelligent Behavior as an Adaptation to the Task Environment*. Ph.D. thesis, University of Michigan, Ann Arbor.
- Bostrom, N. (2014). *Superintelligence: Paths, Dangers, Strategies*. Oxford University Press.
- Bottou, L., Vapnik, V. (1992). Local learning algorithms. *Neural Computation*, 4(6):888–900.
- Boyan, J. A. (1999). Least-squares temporal difference learning. In *Proceedings of the 16th International Conference on Machine Learning (ICML 1999)*, pp. 49–56.
- Boyan, J. A. (2002). Technical update: Least-squares temporal difference learning. *Machine Learning*, 49(2):233–246.
- Boyan, J. A., Moore, A. W. (1995). Generalization in reinforcement learning: Safely approximating the value function. In *Advances in Neural Information Processing Systems 7 (NIPS 1994)*, pp. 369–376. MIT Press, Cambridge, MA.
- Bradtko, S. J. (1993). Reinforcement learning applied to linear quadratic regulation. In *Advances in Neural Information Processing Systems 5 (NIPS 1992)*, pp. 295–302. Morgan Kaufmann.
- Bradtko, S. J. (1994). *Incremental Dynamic Programming for On-Line Adaptive Optimal Control*. Ph.D. thesis, University of Massachusetts, Amherst. Appeared as CMPSCI Technical Report 94-62.
- Bradtko, S. J., Barto, A. G. (1996). Linear least-squares algorithms for temporal difference learning. *Machine Learning*, 22:33–57.
- Bradtko, S. J., Ydstie, B. E., Barto, A. G. (1994). Adaptive linear quadratic control using policy iteration. In *Proceedings of the American Control Conference*, pp. 3475–3479. American Automatic Control Council, Evanston, IL.
- Brafman, R. I., Tennenholtz, M. (2003). R-max – a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3:213–231.
- Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):5–32.
- Breiter, H. C., Aharon, I., Kahneman, D., Dale, A., Shizgal, P. (2001). Functional imaging of neural responses to expectancy and experience of monetary gains and losses. *Neuron*, 30(2):619–639.
- Breland, K., Breland, M. (1961). The misbehavior of organisms. *American Psychologist*, 16(11):681–684.
- Bridle, J. S. (1990). Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimates of parameters. In *Advances in Neural Information Processing Systems 2 (NIPS 1989)*, pp. 211–217. Morgan Kaufmann, San Mateo, CA.
- Broomhead, D. S., Lowe, D. (1988). Multivariable functional interpolation and adaptive networks. *Complex Systems*, 2:321–355.
- Bromberg-Martin, E. S., Matsumoto, M., Hong, S., Hikosaka, O. (2010). A pallidus-habenula-dopamine pathway signals inferred stimulus values. *Journal of Neurophysiology*, 104(2):1068–1076.
- Browne, C.B., Powley, E., Whitehouse, D., Lucas, S.M., Cowling, P.I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., Colton, S. (2012). A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43.
- Brown, J., Bullock, D., Grossberg, S. (1999). How the basal ganglia use parallel excitatory and inhibitory learning pathways to selectively respond to unexpected rewarding cues. *The Journal of Neuroscience*, 19(23):10502–10511.
- Bryson, A. E., Jr. (1996). Optimal control—1950 to 1985. *IEEE Control Systems*, 13(3):26–33.

- Buchanan, B. G., Mitchell, T., Smith, R. G., Johnson, C. R., Jr. (1978). Models of learning systems. *Encyclopedia of Computer Science and technology*, 11.
- Buhusi, C. V., Schmajuk, N. A. (1999). Timing in simple conditioning and occasion setting: A neural network approach. *Behavioural Processes*, 45(1):33–57.
- Bušoniu, L., Lazaric, A., Ghavamzadeh, M., Munos, R., Babuška, R., De Schutter, B. (2012). Least-squares methods for policy iteration. In M. Wiering and M. van Otterlo (Eds.), *Reinforcement Learning: State-of-the-Art*, pp. 75–109. Springer-Verlag Berlin Heidelberg.
- Bush, R. R., Mosteller, F. (1955). *Stochastic Models for Learning*. Wiley, New York.
- Byrne, J. H., Gingrich, K. J., Baxter, D. A. (1990). Computational capabilities of single neurons: Relationship to simple forms of associative and nonassociative learning in *aplysia*. In R. D. Hawkins and G. H. Bower (Eds.), *Computational Models of Learning*, pp. 31–63. Academic Press, New York.
- Calabresi, P., Picconi, B., Tozzi, A., Filippo, M. D. (2007). Dopamine-mediated regulation of corticostriatal synaptic plasticity. *Trends in Neuroscience*, 30(5):211–219.
- Camerer, C. (2011). *Behavioral Game Theory: Experiments in Strategic Interaction*. Princeton University Press.
- Campbell, D. T. (1960). Blind variation and selective survival as a general strategy in knowledge-processes. In M. C. Yovits and S. Cameron (Eds.), *Self-Organizing Systems*, pp. 205–231. Pergamon, New York.
- Cao, X. R. (2009). Stochastic learning and optimization—A sensitivity-based approach. *Annual Reviews in Control*, 33(1):11–24.
- Cao, X. R., Chen, H. F. (1997). Perturbation realization, potentials, and sensitivity analysis of Markov processes. *IEEE Transactions on Automatic Control*, 42(10):1382–1393.
- Carlström, J., Nordström, E. (1997). Control of self-similar ATM call traffic by reinforcement learning. In *Proceedings of the International Workshop on Applications of Neural Networks to Telecommunications 3*, pp. 54–62. Erlbaum, Hillsdale, NJ.
- Chapman, D., Kaelbling, L. P. (1991). Input generalization in delayed reinforcement learning: An algorithm and performance comparisons. In *Proceedings of the Twelfth International Conference on Artificial Intelligence (IJCAI-91)*, pp. 726–731. Morgan Kaufmann, San Mateo, CA.
- Chaslot, G., Bakkes, S., Szita, I., Spronck, P. (2008). Monte-Carlo tree search: A new framework for game AI. In *Proceedings of the Fourth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIDE-08)*, pp. 216–217. AAAI Press, Menlo Park, CA.
- Chow, C.-S., Tsitsiklis, J. N. (1991). An optimal one-way multigrid algorithm for discrete-time stochastic control. *IEEE Transactions on Automatic Control*, 36(8):898–914.
- Chrismann, L. (1992). Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, pp. 183–188. AAAI/MIT Press, Menlo Park, CA.
- Christensen, J., Korf, R. E. (1986). A unified theory of heuristic evaluation functions and its application to learning. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pp. 148–152. Morgan Kaufmann.
- Cichosz, P. (1995). Truncating temporal differences: On the efficient implementation of $\text{TD}(\lambda)$ for reinforcement learning. *Journal of Artificial Intelligence Research*, 2:287–318.
- Ciosek, K., Whiteson, S. (2017). Expected policy gradients. ArXiv:1706.05374v1. A revised version appeared in *Proceedings of the Annual Conference of the Association for the Advancement of Artificial Intelligence (AAAI-2018)*, pp. 2868–2875.

- Ciosek, K., Whiteson, S. (2018). Expected policy gradients for reinforcement learning. ArXiv: 1801.03326.
- Claridge-Chang, A., Roorda, R. D., Vrontou, E., Sjulson, L., Li, H., Hirsh, J., Miesenböck, G. (2009). Writing memories with light-addressable reinforcement circuitry. *Cell*, 139(2):405–415.
- Clark, R. E., Squire, L. R. (1998). Classical conditioning and brain systems: the role of awareness. *Science*, 280(5360):77–81.
- Clark, W. A., Farley, B. G. (1955). Generalization of pattern recognition in a self-organizing system. In *Proceedings of the 1955 Western Joint Computer Conference*, pp. 86–91.
- Clouse, J. (1996). *On Integrating Apprentice Learning and Reinforcement Learning* TITLE2. Ph.D. thesis, University of Massachusetts, Amherst. Appeared as CMPSCI Technical Report 96-026.
- Clouse, J., Utgoff, P. (1992). A teaching method for reinforcement learning systems. In *Proceedings of the 9th International Workshop on Machine Learning*, pp. 92–101. Morgan Kaufmann.
- Cobo, L. C., Zang, P., Isbell, C. L., Thomaz, A. L. (2011). Automatic state abstraction from demonstration. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI-11)*, pp. 1243–1248. AAAI Press.
- Connell, J. (1989). A colony architecture for an artificial creature. Technical Report AI-TR-1151. MIT Artificial Intelligence Laboratory, Cambridge, MA.
- Connell, M. E., Utgoff, P. E. (1987). Learning to control a dynamic physical system. *Computational intelligence*, 3(1):330–337.
- Contreras-Vidal, J. L., Schultz, W. (1999). A predictive reinforcement model of dopamine neurons for learning approach behavior. *Journal of Computational Neuroscience*, 6(3):191–214.
- Coulom, R. (2006). Efficient selectivity and backup operators in Monte-Carlo tree search. In *Proceedings of the 5th International Conference on Computers and Games (CG'06)*, pp. 72–83. Springer-Verlag Berlin, Heidelberg.
- Courville, A. C., Daw, N. D., Touretzky, D. S. (2006). Bayesian theories of conditioning in a changing world. *Trends in Cognitive Science*, 10(7):294–300.
- Craik, K. J. W. (1943). *The Nature of Explanation*. Cambridge University Press, Cambridge.
- Cross, J. G. (1973). A stochastic learning model of economic behavior. *The Quarterly Journal of Economics*, 87(2):239–266.
- Crow, T. J. (1968). Cortical synapses and reinforcement: a hypothesis. *Nature*, 219(5155):736–737.
- Curtiss, J. H. (1954). A theoretical comparison of the efficiencies of two classical methods and a Monte Carlo method for computing one component of the solution of a set of linear algebraic equations. In H. A. Meyer (Ed.), *Symposium on Monte Carlo Methods*, pp. 191–233. Wiley, New York.
- Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314.
- Cziko, G. (1995). *Without Miracles: Universal Selection Theory and the Second Darwinian Revolution*. MIT Press, Cambridge, MA.
- Dabney, W. (2014). *Adaptive step-sizes for reinforcement learning*. PhD thesis, University of Massachusetts, Amherst.
- Dabney, W., Barto, A. G. (2012). Adaptive step-size for online temporal difference learning. In *Proceedings of the Annual Conference of the Association for the Advancement of Artificial Intelligence (AAAI)*.

- Daniel, J. W. (1976). Splines and efficiency in dynamic programming. *Journal of Mathematical Analysis and Applications*, 54:402–407.
- Dann, C., Neumann, G., Peters, J. (2014). Policy evaluation with temporal differences: A survey and comparison. *Journal of Machine Learning Research*, 15:809–883.
- Daw, N. D., Courville, A. C., Touretzky, D. S. (2003). Timing and partial observability in the dopamine system. In *Advances in Neural Information Processing Systems 15 (NIPS 2002)*, pp. 99–106. MIT Press, Cambridge, MA.
- Daw, N. D., Courville, A. C., Touretzky, D. S. (2006). Representation and timing in theories of the dopamine system. *Neural Computation*, 18(7):1637–1677.
- Daw, N. D., Niv, Y., Dayan, P. (2005). Uncertainty based competition between prefrontal and dorsolateral striatal systems for behavioral control. *Nature Neuroscience*, 8(12):1704–1711.
- Daw, N. D., Shohamy, D. (2008). The cognitive neuroscience of motivation and learning. *Social Cognition*, 26(5):593–620.
- Dayan, P. (1991). Reinforcement comparison. In D. S. Touretzky, J. L. Elman, T. J. Sejnowski, and G. E. Hinton (Eds.), *Connectionist Models: Proceedings of the 1990 Summer School*, pp. 45–51. Morgan Kaufmann.
- Dayan, P. (1992). The convergence of TD(λ) for general λ . *Machine Learning*, 8(3):341–362.
- Dayan, P. (2002). Matters temporal. *Trends in Cognitive Sciences*, 6(3):105–106.
- Dayan, P., Abbott, L. F. (2001). *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. MIT Press, Cambridge, MA.
- Dayan, P., Berridge, K. C. (2014). Model-based and model-free Pavlovian reward learning: Revaluation, revision, and revaluation. *Cognitive, Affective, & Behavioral Neuroscience*, 14(2):473–492.
- Dayan, P., Niv, Y. (2008). Reinforcement learning: the good, the bad and the ugly. *Current Opinion in Neurobiology*, 18(2):185–196.
- Dayan, P., Niv, Y., Seymour, B., Daw, N. D. (2006). The misbehavior of value and the discipline of the will. *Neural Networks*, 19(8):1153–1160.
- Dayan, P., Sejnowski, T. (1994). TD(λ) converges with probability 1. *Machine Learning*, 14(3):295–301.
- De Asis, K., Hernandez-Garcia, J. F., Holland, G. Z., Sutton, R. S. (2017). Multi-step Reinforcement Learning: A Unifying Algorithm. ArXiv:1703.01327.
- de Farias, D. P. (2002). The Linear Programming Approach to Approximate Dynamic Programming: Theory and Application. Stanford University PhD thesis.
- de Farias, D. P., Van Roy, B. (2003). The linear programming approach to approximate dynamic programming. *Operations Research* 51(6):850–865.
- Dean, T., Lin, S.-H. (1995). Decomposition techniques for planning in stochastic domains. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, pp. 1121–1127. Morgan Kaufmann. See also Technical Report CS-95-10, Brown University, Department of Computer Science, 1995.
- Degrif, T., White, M., Sutton, R. S. (2012). Off-policy actor-critic. In *Proceedings of the 29th International Conference on Machine Learning (ICML 2012)*. ArXiv:1205.4839, 2012.
- Denardo, E. V. (1967). Contraction mappings in the theory underlying dynamic programming. *SIAM Review*, 9(2):165–177.
- Dennett, D. C. (1978). Why the Law of Effect Will Not Go Away. *Brainstorms*, pp. 71–89. Bradford/MIT Press, Cambridge, MA.
- Derthick, M. (1984). Variations on the Boltzmann machine learning algorithm. Carnegie-Mellon University Department of Computer Science Technical Report No. CMU-CS-84-120.

- Deutsch, J. A. (1953). A new type of behaviour theory. *British Journal of Psychology. General Section*, 44(4):304–317.
- Deutsch, J. A. (1954). A machine with insight. *Quarterly Journal of Experimental Psychology*, 6(1):6–11.
- Dick, T. (2015). *Policy Gradient Reinforcement Learning Without Regret*. M.Sc. thesis, University of Alberta.
- Dickinson, A. (1980). *Contemporary Animal Learning Theory*. Cambridge University Press.
- Dickinson, A. (1985). Actions and habits: the development of behavioral autonomy. *Phil. Trans. R. Soc. Lond. B*, 308(1135):67–78.
- Dickinson, A., Balleine, B. W. (2002). The role of learning in motivation. In C. R. Gallistel (Ed.), *Stevens' Handbook of Experimental Psychology*, volume 3, pp. 497–533. Wiley, NY.
- Dietterich, T. G., Buchanan, B. G. (1984). The role of the critic in learning systems. In O. G. Selfridge, E. L. Risland, and M. A. Arbib (Eds.), *Adaptive Control of Ill-Defined Systems*, pp. 127–147. Plenum Press, NY.
- Dietterich, T. G., Flann, N. S. (1995). Explanation-based learning and reinforcement learning: A unified view. In A. Prieditis and S. Russell (Eds.), *Proceedings of the 12th International Conference on Machine Learning*, pp. 176–184. Morgan Kaufmann.
- Dietterich, T. G., Wang, X. (2002). Batch value function approximation via support vectors. In *Advances in Neural Information Processing Systems 14*, pp. 1491–1498. MIT Press, Cambridge, MA.
- Diuk, C., Cohen, A., Littman, M. L. (2008). An object-oriented representation for efficient reinforcement learning. In *Proceedings of the 25th International Conference on Machine Learning*, pp. 240–247. ACM, New York.
- Dolan, R. J., Dayan, P. (2013). Goals and habits in the brain. *Neuron*, 80(2):312–325.
- Doll, B. B., Simon, D. A., Daw, N. D. (2012). The ubiquity of model-based reinforcement learning. *Current Opinion in Neurobiology*, 22(6):1–7.
- Donahoe, J. W., Burgos, J. E. (2000). Behavior analysis and revaluation. *Journal of the Experimental Analysis of Behavior*, 74(3):331–346.
- Dorigo, M., Colombetti, M. (1994). Robot shaping: Developing autonomous agents through learning. *Artificial Intelligence*, 71(2):321–370.
- Doya, K. (1996). Temporal difference learning in continuous time and space. In *Advances in Neural Information Processing Systems 8 (NIPS 1995)*, pp. 1073–1079. MIT Press, Cambridge, MA.
- Doya, K., Sejnowski, T. J. (1995). A novel reinforcement model of birdsong vocalization learning. In *Advances in Neural Information Processing Systems 7 (NIPS 1994)*, pp. 101–108. MIT Press, Cambridge, MA.
- Doya, K., Sejnowski, T. J. (1998). A computational model of birdsong learning by auditory experience and auditory feedback. In P. W. F. Poon and J. F. Brugge (Eds.), *Central Auditory Processing and Neural Modeling*, pp. 77–88. Springer, Boston, MA.
- Doyle, P. G., Snell, J. L. (1984). *Random Walks and Electric Networks*. The Mathematical Association of America. Carus Mathematical Monograph 22.
- Dreyfus, S. E., Law, A. M. (1977). *The Art and Theory of Dynamic Programming*. Academic Press, New York.
- Du, S. S., Chen, J., Li, L., Xiao, L., Zhou, D. (2017). Stochastic variance reduction methods for policy evaluation. *Proceedings of the 34th International Conference on Machine Learning*, pp. 1049–1058. ArXiv:1702.07944.
- Duda, R. O., Hart, P. E. (1973). *Pattern Classification and Scene Analysis*. Wiley, New York.

- Duff, M. O. (1995). Q-learning for bandit problems. In *Proceedings of the 12th International Conference on Machine Learning*, pp. 209–217. Morgan Kaufmann.
- Egger, D. M., Miller, N. E. (1962). Secondary reinforcement in rats as a function of information value and reliability of the stimulus. *Journal of Experimental Psychology*, 64:97–104.
- Eshel, N., Tian, J., Bukwicz, M., Uchida, N. (2016). Dopamine neurons share common response function for reward prediction error. *Nature Neuroscience*, 19(3):479–486.
- Estes, W. K. (1943). Discriminative conditioning. I. A discriminative property of conditioned anticipation. *Journal of Experimental Psychology*, 32(2):150–155.
- Estes, W. K. (1948). Discriminative conditioning. II. Effects of a Pavlovian conditioned stimulus upon a subsequently established operant response. *Journal of Experimental Psychology*, 38(2):173–177.
- Estes, W. K. (1950). Toward a statistical theory of learning. *Psychological Review*, 57(2):94–107.
- Farley, B. G., Clark, W. A. (1954). Simulation of self-organizing systems by digital computer. *IRE Transactions on Information Theory*, 4(4):76–84.
- Farries, M. A., Fairhall, A. L. (2007). Reinforcement learning with modulated spike timing-dependent synaptic plasticity. *Journal of Neurophysiology*, 98(6):3648–3665.
- Feldbaum, A. A. (1965). *Optimal Control Systems*. Academic Press, New York.
- Finch, G., Culler, E. (1934). Higher order conditioning with constant motivation. *The American Journal of Psychology*:596–602.
- Finnsson, H., Björnsson, Y. (2008). Simulation-based approach to general game playing. In *Proceedings of the Association for the Advancement of Artificial Intelligence*, pp. 259–264.
- Fiorillo, C. D., Yun, S. R., Song, M. R. (2013). Diversity and homogeneity in responses of midbrain dopamine neurons. *The Journal of Neuroscience*, 33(11):4693–4709.
- Florian, R. V. (2007). Reinforcement learning through modulation of spike-timing-dependent synaptic plasticity. *Neural Computation*, 19(6):1468–1502.
- Fogel, L. J., Owens, A. J., Walsh, M. J. (1966). *Artificial Intelligence through Simulated Evolution*. John Wiley and Sons.
- French, R. M. (1999). Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135.
- Frey, U., Morris, R. G. M. (1997). Synaptic tagging and long-term potentiation. *Nature*, 385(6616):533–536.
- Frémaux, N., Sprikeler, H., Gerstner, W. (2010). Functional requirements for reward-modulated spike-timing-dependent plasticity. *The Journal of Neuroscience*, 30(40): 13326–13337
- Friedman, J. H., Bentley, J. L., Finkel, R. A. (1977). An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3(3):209–226.
- Friston, K. J., Tononi, G., Reck, G. N., Sporns, O., Edelman, G. M. (1994). Value-dependent selection in the brain: Simulation in a synthetic neural model. *Neuroscience*, 59(2):229–243.
- Fu, K. S. (1970). Learning control systems—Review and outlook. *IEEE Transactions on Automatic Control*, 15(2):210–221.
- Galanter, E., Gerstenhaber, M. (1956). On thought: The extrinsic theory. *Psychological Review*, 63(4):218–227.
- Gallistel, C. R. (2005). Deconstructing the law of effect. *Games and Economic Behavior*, 52(2):410–423.
- Gardner, M. (1973). Mathematical games. *Scientific American*, 228(1):108–115.
- Geist, M., Scherrer, B. (2014). Off-policy learning with eligibility traces: A survey. *Journal of Machine Learning Research*, 15(1):289–333.
- Gelly, S., Silver, D. (2007). Combining online and offline knowledge in UCT. *Proceedings of the*

- 24th International Conference on Machine Learning (ICML 2007), pp. 273–280.
- Gelperin, A., Hopfield, J. J., Tank, D. W. (1985). The logic of *limax* learning. In A. Selverston (Ed.), *Model Neural Networks and Behavior*, pp. 247–261. Plenum Press, New York.
- Genesereth, M., Thielscher, M. (2014). General game playing. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 8(2):1–229.
- Gershman, S. J., Moustafa, A. A., Ludvig, E. A. (2014). Time representation in reinforcement learning models of the basal ganglia. *Frontiers in Computational Neuroscience*, 7:194.
- Gershman, S. J., Pesaran, B., Daw, N. D. (2009). Human reinforcement learning subdivides structured action spaces by learning effector-specific values. *The Journal of Neuroscience*, 29(43):13524–13531.
- Ghiassian, S., Rafiee, B., Sutton, R. S. (2016). A first empirical study of emphatic temporal difference learning. Workshop on Continual Learning and Deep Learning at the Conference on Neural Information Processing Systems (NIPS 2016). ArXiv:1705.04185.
- Ghiassian, S., White, A., White, M., Sutton, R. S. (in preparation). A comparison of off-policy policy evaluation methods.
- Gibbs, C. M., Cool, V., Land, T., Kehoe, E. J., Gormezano, I. (1991). Second-order conditioning of the rabbit's nictitating membrane response. *Integrative Physiological and Behavioral Science*, 26(4):282–295.
- Gittins, J. C., Jones, D. M. (1974). A dynamic allocation index for the sequential design of experiments. In J. Gani, K. Sarkadi, and I. Vincze (Eds.), *Progress in Statistics*, pp. 241–266. North-Holland, Amsterdam–London.
- Glimcher, P. W. (2011). Understanding dopamine and reinforcement learning: The dopamine reward prediction error hypothesis. *Proceedings of the National Academy of Sciences*, 108(Supplement 3):15647–15654.
- Glimcher, P. W. (2003). *Decisions, Uncertainty, and the Brain: The science of Neuroeconomics*. MIT Press, Cambridge, MA.
- Glimcher, P. W., Fehr, E. (Eds.) (2013). *Neuroeconomics: Decision Making and the Brain, Second Edition*. Academic Press.
- Goethe, J. W. V. (1878). The Sorcerer's Apprentice. In *The Permanent Goethe*, p. 349. The Dial Press, Inc., New York.
- Goldstein, H. (1957). *Classical Mechanics*. Addison-Wesley, Reading, MA.
- Goodfellow, I., Bengio, Y., Courville, A. (2016). *Deep Learning*. MIT Press, Cambridge, MA.
- Goodwin, G. C., Sin, K. S. (1984). *Adaptive Filtering Prediction and Control*. Prentice-Hall, Englewood Cliffs, NJ.
- Gopnik, A., Glymour, C., Sobel, D., Schulz, L. E., Kushnir, T., Danks, D. (2004). A theory of causal learning in children: Causal maps and Bayes nets. *Psychological Review*, 111(1):3–32.
- Gordon, G. J. (1995). Stable function approximation in dynamic programming. In A. Prieditis and S. Russell (Eds.), *Proceedings of the 12th International Conference on Machine Learning (ICML 1995)*, pp. 261–268. Morgan Kaufmann. An expanded version was published as Technical Report CMU-CS-95-103. Carnegie Mellon University, Pittsburgh, PA, 1995.
- Gordon, G. J. (1996a). Chattering in SARSA(λ). CMU learning lab internal report.
- Gordon, G. J. (1996b). Stable fitted reinforcement learning. In *Advances in Neural Information Processing Systems 8 (NIPS 1995)*, pp. 1052–1058. MIT Press, Cambridge, MA.
- Gordon, G. J. (1999). *Approximate Solutions to Markov Decision Processes*. Ph.D. thesis, Carnegie Mellon University, Pittsburgh PA. Pittsburgh, PA.

- Gordon, G. J. (2001). Reinforcement learning with function approximation converges to a region. In *Advances in Neural Information Processing Systems 13 (NIPS 2000)*, pp. 1040–1046. MIT Press, Cambridge, MA.
- Graybiel, A. M. (2000). The basal ganglia. *Current Biology*, 10(14):R509–R511.
- Greensmith, E., Bartlett, P. L., Baxter, J. (2002). Variance reduction techniques for gradient estimates in reinforcement learning. In *Advances in Neural Information Processing Systems 14 (NIPS 2001)*, pp. 1507–1514. MIT Press, Cambridge, MA.
- Greensmith, E., Bartlett, P. L., Baxter, J. (2004). Variance reduction techniques for gradient estimates in reinforcement learning. *Journal of Machine Learning Research*, 5(Nov):1471–1530.
- Griffith, A. K. (1966). A new machine learning technique applied to the game of checkers. Technical Report Project MAC, Artificial Intelligence Memo 94. Massachusetts Institute of Technology, Cambridge, MA.
- Griffith, A. K. (1974). A comparison and evaluation of three machine learning procedures as applied to the game of checkers. *Artificial Intelligence*, 5(2):137–148.
- Grondman, I., Busoniu, L., Lopes, G. A., Babuska, R. (2012). A survey of actor–critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(6):1291–1307.
- Grossberg, S. (1975). A neural model of attention, reinforcement, and discrimination learning. *International Review of Neurobiology*, 18:263–327.
- Grossberg, S., Schmajuk, N. A. (1989). Neural dynamics of adaptive timing and temporal discrimination during associative learning. *Neural Networks*, 2(2):79–102.
- Gullapalli, V. (1990). A stochastic reinforcement algorithm for learning real-valued functions. *Neural Networks*, 3(6): 671–692.
- Gullapalli, V., Barto, A. G. (1992). Shaping as a method for accelerating reinforcement learning. In *Proceedings of the 1992 IEEE International Symposium on Intelligent Control*, pp. 554–559. IEEE.
- Gurvits, L., Lin, L.-J., Hanson, S. J. (1994). Incremental learning of evaluation functions for absorbing Markov chains: New methods and theorems. Siemens Corporate Research, Princeton, NJ.
- Hackman, L. (2012). *Faster Gradient-TD Algorithms*. M.Sc. thesis, University of Alberta, Edmonton.
- Hallak, A., Tamar, A., Mannor, S. (2015). Emphatic TD Bellman operator is a contraction. ArXiv:1508.03411.
- Hallak, A., Tamar, A., Munos, R., Mannor, S. (2016). Generalized emphatic temporal difference learning: Bias-variance analysis. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16)*, pp. 1631–1637. AAAI Press, Menlo Park, CA.
- Hammer, M. (1997). The neural basis of associative reward learning in honeybees. *Trends in Neuroscience*, 20(6):245–252.
- Hammer, M., Menzel, R. (1995). Learning and memory in the honeybee. *The Journal of Neuroscience*, 15(3):1617–1630.
- Hampson, S. E. (1983). *A Neural Model of Adaptive Behavior*. Ph.D. thesis, University of California, Irvine.
- Hampson, S. E. (1989). *Connectionist Problem Solving: Computational Aspects of Biological Learning*. Birkhauser, Boston.
- Hare, T. A., O'Doherty, J., Camerer, C. F., Schultz, W., Rangel, A. (2008). Dissociating the role of the orbitofrontal cortex and the striatum in the computation of goal values and prediction

- errors. *The Journal of Neuroscience*, 28(22):5623–5630.
- Harth, E., Tzanakou, E. (1974). Alopex: A stochastic method for determining visual receptive fields. *Vision Research*, 14(12):1475–1482.
- Hassabis, D., Maguire, E. A. (2007). Deconstructing episodic memory with construction. *Trends in Cognitive Sciences*, 11(7):299–306.
- Hauskrecht, M., Meuleau, N., Kaelbling, L. P., Dean, T., Boutilier, C. (1998). Hierarchical solution of Markov decision processes using macro-actions. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, pp. 220–229. Morgan Kaufmann.
- Hawkins, R. D., Kandel, E. R. (1984). Is there a cell-biological alphabet for simple forms of learning? *Psychological Review*, 91(3):375–391.
- Haykin, S. (1994). *Neural networks: A Comprehensive Foundation*. Macmillan, New York.
- He, K., Huertas, M., Hong, S. Z., Tie, X., Hell, J. W., Shouval, H., Kirkwood, A. (2015). Distinct eligibility traces for LTP and LTD in cortical synapses. *Neuron*, 88(3):528–538.
- He, K., Zhang, X., Ren, S., Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the 1992 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778.
- Hebb, D. O. (1949). *The Organization of Behavior: A Neuropsychological Theory*. John Wiley and Sons Inc., New York. Reissued by Lawrence Erlbaum Associates Inc., Mahwah NJ, 2002.
- Hengst, B. (2012). Hierarchical approaches. In M. Wiering and M. van Otterlo (Eds.), *Reinforcement Learning: State-of-the-Art*, pp. 293–323. Springer-Verlag Berlin Heidelberg.
- Herrnstein, R. J. (1970). On the Law of Effect. *Journal of the Experimental Analysis of Behavior*, 13(2):243–266.
- Hersh, R., Griego, R. J. (1969). Brownian motion and potential theory. *Scientific American*, 220(3):66–74.
- Hester, T., Stone, P. (2012). Learning and using models. In M. Wiering and M. van Otterlo (Eds.), *Reinforcement Learning: State-of-the-Art*, pp. 111–141. Springer-Verlag Berlin Heidelberg.
- Hesterberg, T. C. (1988). *Advances in Importance Sampling*, Ph.D. thesis, Statistics Department, Stanford University.
- Hilgard, E. R. (1956). *Theories of Learning, Second Edition*. Appleton-Century-Crofts, Inc., New York.
- Hilgard, E. R., Bower, G. H. (1975). *Theories of Learning*. Prentice-Hall, Englewood Cliffs, NJ.
- Hinton, G. E. (1984). Distributed representations. Technical Report CMU-CS-84-157. Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA.
- Hinton, G. E., Osindero, S., Teh, Y. (2006). A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554.
- Hochreiter, S., Schmidhuber, J. (1997). LSTM can solve hard time lag problems. In *Advances in Neural Information Processing Systems 9 (NIPS 1996)*, pp. 473–479. MIT Press, Cambridge, MA.
- Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor.
- Holland, J. H. (1976). Adaptation. In R. Rosen and F. M. Snell (Eds.), *Progress in Theoretical Biology*, vol. 4, pp. 263–293. Academic Press, New York.
- Holland, J. H. (1986). Escaping brittleness: The possibility of general-purpose learning algorithms applied to rule-based systems. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach*, vol. 2, pp. 593–623. Morgan Kaufmann.
- Hollerman, J. R., Schultz, W. (1998). Dopamine neurons report an error in the temporal

- prediction of reward during learning. *Nature Neuroscience*, 1(4):304–309.
- Houk, J. C., Adams, J. L., Barto, A. G. (1995). A model of how the basal ganglia generates and uses neural signals that predict reinforcement. In J. C. Houk, J. L. Davis, and D. G. Beiser (Eds.), *Models of Information Processing in the Basal Ganglia*, pp. 249–270. MIT Press, Cambridge, MA.
- Howard, R. (1960). *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, MA.
- Hull, C. L. (1932). The goal-gradient hypothesis and maze learning. *Psychological Review*, 39(1):25–43.
- Hull, C. L. (1943). *Principles of Behavior*. Appleton-Century, New York.
- Hull, C. L. (1952). *A Behavior System*. Wiley, New York.
- Ioffe, S., Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. ArXiv:1502.03167.
- İpek, E., Mutlu, O., Martínez, J. F., Caruana, R. (2008). Self-optimizing memory controllers: A reinforcement learning approach. In *ISCA '08:Proceedings of the 35th Annual International Symposium on Computer Architecture*, pp. 39–50. IEEE Computer Society Washington, DC.
- Izhikevich, E. M. (2007). Solving the distal reward problem through linkage of STDP and dopamine signaling. *Cerebral Cortex*, 17(10):2443–2452.
- Jaakkola, T., Jordan, M. I., Singh, S. P. (1994). On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, 6:1185–1201.
- Jaakkola, T., Singh, S. P., Jordan, M. I. (1995). Reinforcement learning algorithm for partially observable Markov decision problems. In *Advances in Neural Information Processing Systems 7 (NIPS 1994)*, pp. 345–352. MIT Press, Cambridge, MA.
- Jacobs, R. A. (1988). Increased rates of convergence through learning rate adaptation. *Neural Networks*, 1(4):295–307.
- Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., Kavukcuoglu, K. (2016). Reinforcement learning with unsupervised auxiliary tasks. ArXiv:1611.05397.
- Jaeger, H. (1997). Observable operator models and conditioned continuation representations. Arbeitspapiere der GMD 1043, GMD Forschungszentrum Informationstechnik, Sankt Augustin, Germany.
- Jaeger, H. (1998). *Discrete Time, Discrete Valued Observable Operator Models: A Tutorial*. GMD-Forschungszentrum Informationstechnik.
- Jaeger, H. (2000). Observable operator models for discrete stochastic time series. *Neural Computation*, 12(6):1371–1398.
- Jaeger, H. (2002). Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the ‘echo state network’ approach. German National Research Center for Information Technology, Technical Report GMD report 159, 2002.
- Joel, D., Niv, Y., Ruppin, E. (2002). Actor–critic models of the basal ganglia: New anatomical and computational perspectives. *Neural Networks*, 15(4):535–547.
- Johnson, A., Redish, A. D. (2007). Neural ensembles in CA3 transiently encode paths forward of the animal at a decision point. *The Journal of Neuroscience*, 27(45):12176–12189.
- Kaelbling, L. P. (1993a). Hierarchical learning in stochastic domains: Preliminary results. In *Proceedings of the 10th International Conference on Machine Learning (ICML 1993)*, pp. 167–173. Morgan Kaufmann.
- Kaelbling, L. P. (1993b). *Learning in Embedded Systems*. MIT Press, Cambridge, MA.
- Kaelbling, L. P. (Ed.) (1996). Special triple issue on reinforcement learning, *Machine Learning*, 22(1/2/3).
- Kaelbling, L. P., Littman, M. L., Moore, A. W. (1996). Reinforcement learning: A survey.

- Journal of Artificial Intelligence Research*, 4:237–285.
- Kakade, S. M. (2002). A natural policy gradient. In *Advances in Neural Information Processing Systems 14 (NIPS 2001)*, pp. 1531–1538. MIT Press, Cambridge, MA.
- Kakade, S. M. (2003). *On the Sample Complexity of Reinforcement Learning*. Ph.D. thesis, University of London.
- Kakutani, S. (1945). Markov processes and the Dirichlet problem. *Proceedings of the Japan Academy*, 21(3-10):227–233.
- Kalos, M. H., Whitlock, P. A. (1986). *Monte Carlo Methods*. Wiley, New York.
- Kamin, L. J. (1968). “Attention-like” processes in classical conditioning. In M. R. Jones (Ed.), *Miami Symposium on the Prediction of Behavior, 1967: Aversive Stimulation*, pp. 9–31. University of Miami Press, Coral Gables, Florida.
- Kamin, L. J. (1969). Predictability, surprise, attention, and conditioning. In B. A. Campbell and R. M. Church (Eds.), *Punishment and Aversive Behavior*, pp. 279–296. Appleton-Century-Crofts, New York.
- Kandel, E. R., Schwartz, J. H., Jessell, T. M., Siegelbaum, S. A., Hudspeth, A. J. (Eds.) (2013). *Principles of Neural Science, Fifth Edition*. McGraw-Hill Companies, Inc.
- Karampatziakis, N., Langford, J. (2010). Online importance weight aware updates. ArXiv:1011.1576.
- Kashyap, R. L., Blaydon, C. C., Fu, K. S. (1970). Stochastic approximation. In J. M. Mendel and K. S. Fu (Eds.), *Adaptive, Learning, and Pattern Recognition Systems: Theory and Applications*, pp. 329–355. Academic Press, New York.
- Kearney, A., Veeriah, V., Travnik, J., Sutton, R. S., Pilarski, P. M. (in preparation). TIDBD: Adapting Temporal-difference Step-sizes Through Stochastic Meta-descent.
- Kearns, M., Singh, S. (2002). Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 49(2-3):209–232.
- Keerthi, S. S., Ravindran, B. (1997). Reinforcement learning. In E. Fieslerm and R. Beale (Eds.), *Handbook of Neural Computation*, C3. Oxford University Press, New York.
- Kehoe, E. J. (1982). Conditioning with serial compound stimuli: Theoretical and empirical issues. *Experimental Animal Behavior*, 1:30–65.
- Kehoe, E. J., Schreurs, B. G., Graham, P. (1987). Temporal primacy overrides prior training in serial compound conditioning of the rabbit’s nictitating membrane response. *Animal Learning & Behavior*, 15(4):455–464.
- Keiflin, R., Janak, P. H. (2015). Dopamine prediction errors in reward learning and addiction: From theory to neural circuitry. *Neuron*, 88(2):247–263.
- Kimble, G. A. (1961). *Hilgard and Marquis’ Conditioning and Learning*. Appleton-Century-Crofts, New York.
- Kimble, G. A. (1967). *Foundations of Conditioning and Learning*. Appleton-Century-Crofts, New York.
- Kingma, D., Ba, J. (2014). Adam: A method for stochastic optimization. ArXiv:1412.6980.
- Klop, A. H. (1972). Brain function and adaptive systems—A heterostatic theory. Technical Report AFCRL-72-0164, Air Force Cambridge Research Laboratories, Bedford, MA. A summary appears in *Proceedings of the International Conference on Systems, Man, and Cybernetics (1974)*. IEEE Systems, Man, and Cybernetics Society, Dallas, TX.
- Klop, A. H. (1975). A comparison of natural and artificial intelligence. *SIGART Newsletter*, 53:11–13.
- Klop, A. H. (1982). *The Hedonistic Neuron: A Theory of Memory, Learning, and Intelligence*. Hemisphere, Washington, DC.
- Klop, A. H. (1988). A neuronal model of classical conditioning. *Psychobiology*, 16(2):85–125.

- Klyubin, A. S., Polani, D., Nehaniv, C. L. (2005). Empowerment: A universal agent-centric measure of control. In *Proceedings of the 2005 IEEE Congress on Evolutionary Computation* (Vol. 1, pp. 128–135). IEEE.
- Kober, J., Peters, J. (2012). Reinforcement learning in robotics: A survey. In M. Wiering, M. van Otterlo (Eds.), *Reinforcement Learning: State-of-the-Art*, pp. 579–610. Springer-Verlag.
- Kocsis, L., Szepesvári, Cs. (2006). Bandit based Monte-Carlo planning. In *Proceedings of the European Conference on Machine Learning*, pp. 282–293. Springer-Verlag Berlin Heidelberg.
- Kohonen, T. (1977). *Associative Memory: A System Theoretic Approach*. Springer-Verlag, Berlin.
- Koller, D., Friedman, N. (2009). *Probabilistic Graphical Models: Principles and Techniques*. MIT Press.
- Kolodziejski, C., Porr, B., Wörgötter, F. (2009). On the asymptotic equivalence between differential Hebbian and temporal difference learning. *Neural Computation*, 21(4):1173–1202.
- Kolter, J. Z. (2011). The fixed points of off-policy TD. In *Advances in Neural Information Processing Systems 24 (NIPS 2011)*, pp. 2169–2177. Curran Associates, Inc.
- Konda, V. R., Tsitsiklis, J. N. (2000). Actor-critic algorithms. In *Advances in Neural Information Processing Systems 12 (NIPS 1999)*, pp. 1008–1014. MIT Press, Cambridge, MA.
- Konda, V. R., Tsitsiklis, J. N. (2003). On actor-critic algorithms. *SIAM Journal on Control and Optimization*, 42(4):1143–1166.
- Konidaris, G. D., Osentoski, S., Thomas, P. S. (2011). Value function approximation in reinforcement learning using the Fourier basis . In *Proceedings of the Twenty-Fifth Conference of the Association for the Advancement of Artificial Intelligence*, pp. 380–385.
- Korf, R. E. (1988). Optimal path finding algorithms. In L. N. Kanal and V. Kumar (Eds.), *Search in Artificial Intelligence*, pp. 223–267. Springer-Verlag, Berlin.
- Korf, R. E. (1990). Real-time heuristic search. *Artificial Intelligence*, 42(2–3), 189–211.
- Koshland, D. E. (1980). *Bacterial Chemotaxis as a Model Behavioral System*. Raven Press, New York.
- Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection* (Vol. 1). MIT Press., Cambridge, MA.
- Kraft, L. G., Campagna, D. P. (1990). A summary comparison of CMAC neural network and traditional adaptive control systems. In T. Miller, R. S. Sutton, and P. J. Werbos (Eds.), *Neural Networks for Control*, pp. 143–169. MIT Press, Cambridge, MA.
- Kraft, L. G., Miller, W. T., Dietz, D. (1992). Development and application of CMAC neural network-based control. In D. A. White and D. A. Sofge (Eds.), *Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches*, pp. 215–232. Van Nostrand Reinhold, New York.
- Kumar, P. R., Varaiya, P. (1986). *Stochastic Systems: Estimation, Identification, and Adaptive Control*. Prentice-Hall, Englewood Cliffs, NJ.
- Kumar, P. R. (1985). A survey of some results in stochastic adaptive control. *SIAM Journal of Control and Optimization*, 23(3):329–380.
- Kumar, V., Kanal, L. N. (1988). The CDP, A unifying formulation for heuristic search, dynamic programming, and branch-and-bound. In L. N. Kanal and V. Kumar (Eds.), *Search in Artificial Intelligence*, pp. 1–37. Springer-Verlag, Berlin.
- Kushner, H. J., Dupuis, P. (1992). *Numerical Methods for Stochastic Control Problems in Continuous Time*. Springer-Verlag, New York.
- Kuvayev, L., Sutton, R.S. (1996). Model-based reinforcement learning with an approximate, learned model. *Proceedings of the Ninth Yale Workshop on Adaptive and Learning Systems*,

- pp. 101–105, Yale University, New Haven, CT.
- Lagoudakis, M., Parr, R. (2003). Least squares policy iteration. *Journal of Machine Learning Research*, 4(Dec):1107–1149.
- Lai, T. L., Robbins, H. (1985). Asymptotically efficient adaptive allocation rules. *Advances in Applied Mathematics*, 6(1):4–22.
- Lakshminarahan, S., Narendra, K. S. (1982). Learning algorithms for two-person zero-sum stochastic games with incomplete information: A unified approach. *SIAM Journal of Control and Optimization*, 20(4):541–552.
- Lammel, S., Lim, B. K., Malenka, R. C. (2014). Reward and aversion in a heterogeneous midbrain dopamine system. *Neuropharmacology*, 76:353–359.
- Lane, S. H., Handelman, D. A., Gelfand, J. J. (1992). Theory and development of higher-order CMAC neural networks. *IEEE Control Systems*, 12(2):23–30.
- LeCun, Y. (1985). Une procedure d'apprentissage pour rseau a seuil asymmetrique (a learning scheme for asymmetric threshold networks). In *Proceedings of Cognitiva 85*, Paris, France.
- LeCun, Y., Bottou, L., Bengio, Y., Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.
- Legenstein, R. W., Maass, D. P. (2008). A learning theory for reward-modulated spike-timing-dependent plasticity with application to biofeedback. *PLoS Computational Biology*, 4(10).
- Levy, W. B., Steward, D. (1983). Temporal contiguity requirements for long-term associative potentiation/depression in the hippocampus. *Neuroscience*, 8(4):791–797.
- Lewis, F. L., Liu, D. (Eds.) (2012). *Reinforcement Learning and Approximate Dynamic Programming for Feedback Control*. John Wiley and Sons.
- Lewis, R. L., Howes, A., Singh, S. (2014). Computational rationality: Linking mechanism and behavior through utility maximization. *Topics in Cognitive Science*, 6(2):279–311.
- Li, L. (2012). Sample complexity bounds of exploration. In M. Wiering and M. van Otterlo (Eds.), *Reinforcement Learning: State-of-the-Art*, pp. 175–204. Springer-Verlag Berlin Heidelberg.
- Li, L., Chu, W., Langford, J., Schapire, R. E. (2010). A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th International Conference on World Wide Web*, pp. 661–670. ACM, New York.
- Lin, C.-S., Kim, H. (1991). CMAC-based adaptive critic self-learning control. *IEEE Transactions on Neural Networks*, 2(5):530–533.
- Lin, L.-J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8(3-4):293–321.
- Lin, L.-J., Mitchell, T. (1992). Reinforcement learning with hidden states. In *Proceedings of the Second International Conference on Simulation of Adaptive Behavior: From Animals to Animats*, pp. 271–280. MIT Press, Cambridge, MA.
- Littman, M. L., Cassandra, A. R., Kaelbling, L. P. (1995). Learning policies for partially observable environments: Scaling up. In *Proceedings of the 12th International Conference on Machine Learning (ICML 1995)*, pp. 362–370. Morgan Kaufmann.
- Littman, M. L., Dean, T. L., Kaelbling, L. P. (1995). On the complexity of solving Markov decision problems. In *Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence*, pp. 394–402.
- Littman, M. L., Sutton, R. S., Singh (2002). Predictive representations of state. In *Advances in Neural Information Processing Systems 14 (NIPS 2001)*, pp. 1555–1561. MIT Press, Cambridge, MA.
- Liu, J. S. (2001). *Monte Carlo Strategies in Scientific Computing*. Springer-Verlag, Berlin.
- Ljung, L. (1998). System identification. In A. Procházka, J. Uhlíř, P. W. J. Rayner, and N. G.

- Kingsbury (Eds.), *Signal Analysis and Prediction*, pp. 163–173. Springer Science + Business Media New York, LLC.
- Ljung, L., Söderstrom, T. (1983). *Theory and Practice of Recursive Identification*. MIT Press, Cambridge, MA.
- Ljungberg, T., Apicella, P., Schultz, W. (1992). Responses of monkey dopamine neurons during learning of behavioral reactions. *Journal of Neurophysiology*, 67(1):145–163.
- Lovejoy, W. S. (1991). A survey of algorithmic methods for partially observed Markov decision processes. *Annals of Operations Research*, 28(1):47–66.
- Luce, D. (1959). *Individual Choice Behavior*. Wiley, New York.
- Ludvig, E. A., Bellemare, M. G., Pearson, K. G. (2011). A primer on reinforcement learning in the brain: Psychological, computational, and neural perspectives. In E. Alonso and E. Mondragón (Eds.), *Computational Neuroscience for Advancing Artificial Intelligence: Models, Methods and Applications*, pp. 111–44. Medical Information Science Reference, Hershey PA.
- Ludvig, E. A., Sutton, R. S., Kehoe, E. J. (2008). Stimulus representation and the timing of reward-prediction errors in models of the dopamine system. *Neural Computation*, 20(12):3034–3054.
- Ludvig, E. A., Sutton, R. S., Kehoe, E. J. (2012). Evaluating the TD model of classical conditioning. *Learning & behavior*, 40(3):305–319.
- Machado, A. (1997). Learning the temporal dynamics of behavior. *Psychological Review*, 104(2):241–265.
- Mackintosh, N. J. (1975). A theory of attention: Variations in the associability of stimuli with reinforcement. *Psychological Review*, 82(4):276–298.
- Mackintosh, N. J. (1983). *Conditioning and Associative Learning*. Clarendon Press, Oxford.
- Maclin, R., Shavlik, J. W. (1994). Incorporating advice into agents that learn from reinforcements. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, pp. 694–699. AAAI Press, Menlo Park, CA.
- Maei, H. R. (2011). *Gradient Temporal-Difference Learning Algorithms*. Ph.D. thesis, University of Alberta, Edmonton.
- Maei, H. R. (2018). Convergent actor-critic algorithms under off-policy training and function approximation. ArXiv:1802.07842.
- Maei, H. R., Sutton, R. S. (2010). GQ(λ): A general gradient algorithm for temporal-difference prediction learning with eligibility traces. In *Proceedings of the Third Conference on Artificial General Intelligence*, pp. 91–96.
- Maei, H. R., Szepesvári, Cs., Bhatnagar, S., Precup, D., Silver, D., Sutton, R. S. (2009). Convergent temporal-difference learning with arbitrary smooth function approximation. In *Advances in Neural Information Processing Systems 22 (NIPS 2009)*, pp. 1204–1212. Curran Associates, Inc.
- Maei, H. R., Szepesvári, Cs., Bhatnagar, S., Sutton, R. S. (2010). Toward off-policy learning control with function approximation. In *Proceedings of the 27th International Conference on Machine Learning (ICML 2010)*, pp. 719–726.
- Mahadevan, S. (1996). Average reward reinforcement learning: Foundations, algorithms, and empirical results. *Machine Learning*, 22(1):159–196.
- Mahadevan, S., Liu, B., Thomas, P., Dabney, W., Giguere, S., Jacek, N., Gemp, I., Liu, J. (2014). Proximal reinforcement learning: A new theory of sequential decision making in primal-dual spaces. ArXiv:1405.6757.
- Mahadevan, S., Connell, J. (1992). Automatic programming of behavior-based robots using reinforcement learning. *Artificial Intelligence*, 55(2-3):311–365.

- Mahmood, A. R. (2017). *Incremental Off-Policy Reinforcement Learning Algorithms*. Ph.D. thesis, University of Alberta, Edmonton.
- Mahmood, A. R., Sutton, R. S. (2015). Off-policy learning based on weighted importance sampling with linear computational complexity. In *Proceedings of the 31st Conference on Uncertainty in Artificial Intelligence (UAI-2015)*, pp. 552–561. AUAI Press Corvallis, Oregon.
- Mahmood, A. R., Sutton, R. S., Degris, T., Pilarski, P. M. (2012). Tuning-free step-size adaptation. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Proceedings*, pp. 2121–2124. IEEE.
- Mahmood, A. R., Yu, H., Sutton, R. S. (2017). Multi-step off-policy learning without importance sampling ratios. ArXiv:1702.03006.
- Mahmood, A. R., van Hasselt, H., Sutton, R. S. (2014). Weighted importance sampling for off-policy learning with linear function approximation. *Advances in Neural Information Processing Systems 27 (NIPS 2014)*, pp. 3014–3022. Curran Associates, Inc.
- Marbach, P., Tsitsiklis, J. N. (1998). Simulation-based optimization of Markov reward processes. MIT Technical Report LIDS-P-2411.
- Marbach, P., Tsitsiklis, J. N. (2001). Simulation-based optimization of Markov reward processes. *IEEE Transactions on Automatic Control*, 46(2):191–209.
- Markram, H., Lübke, J., Frotscher, M., Sakmann, B. (1997). Regulation of synaptic efficacy by coincidence of postsynaptic APs and EPSPs. *Science*, 275(5297):213–215.
- Martínez, J. F., İpek, E. (2009). Dynamic multicore resource management: A machine learning approach. *Micro, IEEE*, 29(5):8–17.
- Mataric, M. J. (1994). Reward functions for accelerated learning. In *Proceedings of the 11th International Conference on Machine Learning (ICML 1994)*, pp. 181–189. Morgan Kaufmann.
- Matsuda, W., Furuta, T., Nakamura, K. C., Hioki, H., Fujiyama, F., Arai, R., Kaneko, T. (2009). Single nigrostriatal dopaminergic neurons form widely spread and highly dense axonal arborizations in the neostriatum. *The Journal of Neuroscience*, 29(2):444–453.
- Mazur, J. E. (1994). *Learning and Behavior*, 3rd ed. Prentice-Hall, Englewood Cliffs, NJ.
- McCallum, A. K. (1993). Overcoming incomplete perception with utile distinction memory. In *Proceedings of the 10th International Conference on Machine Learning (ICML 1993)*, pp. 190–196. Morgan Kaufmann.
- McCallum, A. K. (1995). *Reinforcement Learning with Selective Perception and Hidden State*. Ph.D. thesis, University of Rochester, Rochester NY.
- McCloskey, M., Cohen, N. J. (1989). Catastrophic interference in connectionist networks: The sequential learning problem. *Psychology of Learning and Motivation*, 24:109–165.
- McClure, S. M., Daw, N. D., Montague, P. R. (2003). A computational substrate for incentive salience. *Trends in Neurosciences*, 26(8):423–428.
- McCulloch, W. S., Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5(4):115–133.
- McMahan, H. B., Gordon, G. J. (2005). Fast Exact Planning in Markov Decision Processes. In *Proceedings of the International Conference on Automated Planning and Scheduling*, pp. 151–160.
- Melo, F. S., Meyn, S. P., Ribeiro, M. I. (2008). An analysis of reinforcement learning with function approximation. In *Proceedings of the 25th International Conference on Machine Learning (ICML 2008)*, pp. 664–671.
- Mendel, J. M. (1966). A survey of learning control systems. *ISA Transactions*, 5:297–303.
- Mendel, J. M., McLaren, R. W. (1970). Reinforcement learning control and pattern recognition

- systems. In J. M. Mendel and K. S. Fu (Eds.), *Adaptive, Learning and Pattern Recognition Systems: Theory and Applications*, pp. 287–318. Academic Press, New York.
- Michie, D. (1961). Trial and error. In S. A. Barnett and A. McLaren (Eds.), *Science Survey, Part 2*, pp. 129–145. Penguin, Harmondsworth.
- Michie, D. (1963). Experiments on the mechanisation of game learning. 1. characterization of the model and its parameters. *The Computer Journal*, 6(3):232–263.
- Michie, D. (1974). *On Machine Intelligence*. Edinburgh University Press, Edinburgh.
- Michie, D., Chambers, R. A. (1968). BOXES, An experiment in adaptive control. In E. Dale and D. Michie (Eds.), *Machine Intelligence 2*, pp. 137–152. Oliver and Boyd, Edinburgh.
- Miller, R. (1981). *Meaning and Purpose in the Intact Brain: A Philosophical, Psychological, and Biological Account of Conscious Process*. Clarendon Press, Oxford.
- Miller, W. T., An, E., Glanz, F., Carter, M. (1990). The design of CMAC neural networks for control. *Adaptive and Learning Systems*, 1:140–145.
- Miller, W. T., Glanz, F. H. (1996). *UNH_CMAC verison 2.1: The University of New Hampshire Implementation of the Cerebellar Model Arithmetic Computer - CMAC*. Robotics Laboratory Technical Report, University of New Hampshire, Durham.
- Miller, S., Williams, R. J. (1992). Learning to control a bioreactor using a neural net Dyna-Q system. In *Proceedings of the Seventh Yale Workshop on Adaptive and Learning Systems*, pp. 167–172. Center for Systems Science, Dunham Laboratory, Yale University, New Haven.
- Miller, W. T., Scalera, S. M., Kim, A. (1994). Neural network control of dynamic balance for a biped walking robot. In *Proceedings of the Eighth Yale Workshop on Adaptive and Learning Systems*, pp. 156–161. Center for Systems Science, Dunham Laboratory, Yale University, New Haven.
- Minton, S. (1990). Quantitative results concerning the utility of explanation-based learning. *Artificial Intelligence*, 42(2-3):363–391.
- Minsky, M. L. (1954). *Theory of Neural-Analog Reinforcement Systems and Its Application to the Brain-Model Problem*. Ph.D. thesis, Princeton University.
- Minsky, M. L. (1961). Steps toward artificial intelligence. *Proceedings of the Institute of Radio Engineers*, 49:8–30. Reprinted in E. A. Feigenbaum and J. Feldman (Eds.), *Computers and Thought*, pp. 406–450. McGraw-Hill, New York, 1963.
- Minsky, M. L. (1967). *Computation: Finite and Infinite Machines*. Prentice-Hall, Englewood Cliffs, NJ.
- Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M. (2013). Playing atari with deep reinforcement learning. ArXiv:1312.5602.
- Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.
- Modayil, J., Sutton, R. S. (2014). Prediction driven behavior: Learning predictions that drive fixed responses. In *AAAI-14 Workshop on Artificial Intelligence and Robotics*, Quebec City, Canada.
- Modayil, J., White, A., Sutton, R. S. (2014). Multi-timescale nexting in a reinforcement learning robot. *Adaptive Behavior*, 22(2):146–160.
- Monahan, G. E. (1982). State of the art—a survey of partially observable Markov decision processes: theory, models, and algorithms. *Management Science*, 28(1):1–16.
- Montague, P. R., Dayan, P., Nowlan, S. J., Pouget, A., Sejnowski, T. J. (1993). Using aperiodic reinforcement for directed self-organization during development. In *Advances in Neural*

- Information Processing Systems 5 (NIPS 1992)*, pp. 969–976. Morgan Kaufmann.
- Montague, P. R., Dayan, P., Person, C., Sejnowski, T. J. (1995). Bee foraging in uncertain environments using predictive hebbian learning. *Nature*, 377(6551):725–728.
- Montague, P. R., Dayan, P., Sejnowski, T. J. (1996). A framework for mesencephalic dopamine systems based on predictive Hebbian learning. *The Journal of Neuroscience*, 16(5):1936–1947.
- Montague, P. R., Dolan, R. J., Friston, K. J., Dayan, P. (2012). Computational psychiatry. *Trends in Cognitive Sciences*, 16(1):72–80.
- Montague, P. R., Sejnowski, T. J. (1994). The predictive brain: Temporal coincidence and temporal order in synaptic learningmechanisms. *Learning & Memory*, 1(1):1–33.
- Moore, A. W. (1990). *Efficient Memory-Based Learning for Robot Control*. Ph.D. thesis, University of Cambridge.
- Moore, A. W., Atkeson, C. G. (1993). Prioritized sweeping: Reinforcement learning with less data and less real time. *Machine Learning*, 13(1):103–130.
- Moore, A. W., Schneider, J., Deng, K. (1997). Efficient locally weighted polynomial regression predictions. In *Proceedings of the 14th International Conference on Machine Learning (ICML 1997)*. Morgan Kaufmann.
- Moore, J. W., Blazis, D. E. J. (1989). Simulation of a classically conditioned response: A cerebellar implementation of the sutton-barto-desmond model. In J. H. Byrne and W. O. Berry (Eds.), *Neural Models of Plasticity*, pp. 187–207. Academic Press, San Diego, CA.
- Moore, J. W., Choi, J.-S., Brunzell, D. H. (1998). Predictive timing under temporal uncertainty: The time derivative model of the conditioned response. In D. A. Rosenbaum and C. E. Collyer (Eds.), *Timing of Behavior*, pp. 3–34. MIT Press, Cambridge, MA.
- Moore, J. W., Desmond, J. E., Berthier, N. E., Blazis, E. J., Sutton, R. S., Barto, A. G. (1986). Simulation of the classically conditioned nictitating membrane response by a neuron-like adaptive element: I. Response topography, neuronal firing, and interstimulus intervals. *Behavioural Brain Research*, 21(2):143–154.
- Moore, J. W., Marks, J. S., Castagna, V. E., Polewan, R. J. (2001). Parameter stability in the TD model of complex CR topographies. In *Society for Neuroscience Abstracts*, 27:642.
- Moore, J. W., Schmajuk, N. A. (2008). Kamin blocking. *Scholarpedia*, 3(5):3542.
- Moore, J. W., Stickney, K. J. (1980). Formation of attentional-associative networks in real time:Role of the hippocampus and implications for conditioning. *Physiological Psychology*, 8(2):207–217.
- Mukundan, J., Martínez, J. F. (2012). MORSE, Multi-objective reconfigurable self-optimizing memory scheduler. In *IEEE 18th International Symposium on High Performance Computer Architecture (HPCA)*, pp. 1–12.
- Müller, M. (2002). Computer Go. *Artificial Intelligence*, 134(1):145–179.
- Munos, R., Stepleton, T., Harutyunyan, A., Bellemare, M. (2016). Safe and efficient off-policy reinforcement learning. In *Advances in Neural Information Processing Systems 29 (NIPS 2016)*, pp. 1046–1054. Curran Associates, Inc.
- Naddaf, Y. (2010). *Game-Independent AI Agents for Playing Atari 2600 Console Games*. Ph.D. thesis, University of Alberta, Edmonton.
- Narendra, K. S., Thathachar, M. A. L. (1974). Learning automata—A survey. *IEEE Transactions on Systems, Man, and Cybernetics*, 4:323–334.
- Narendra, K. S., Thathachar, M. A. L. (1989). *Learning Automata: An Introduction*. Prentice-Hall, Englewood Cliffs, NJ.
- Narendra, K. S., Wheeler, R. M. (1983). An N-player sequential stochastic game with identical payoffs. *IEEE Transactions on Systems, Man, and Cybernetics*, 6:1154–1158.

- Narendra, K. S., Wheeler, R. M. (1986). Decentralized learning in finite Markov chains. *IEEE Transactions on Automatic Control*, 31(6):519–526.
- Nedić, A., Bertsekas, D. P. (2003). Least squares policy evaluation algorithms with linear function approximation. *Discrete Event Dynamic Systems*, 13(1-2):79–110.
- Ng, A. Y. (2003). *Shaping and Policy Search in Reinforcement Learning*. Ph.D. thesis, University of California, Berkeley.
- Ng, A. Y., Harada, D., Russell, S. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In I. Bratko and S. Dzeroski (Eds.), *Proceedings of the 16th International Conference on Machine Learning (ICML 1999)*, pp. 278–287.
- Ng, A. Y., Russell, S. J. (2000). Algorithms for inverse reinforcement learning. In *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pp. 663–670.
- Niv, Y. (2009). Reinforcement learning in the brain. *Journal of Mathematical Psychology*, 53(3):139–154.
- Niv, Y., Daw, N. D., Dayan, P. (2006). How fast to work: Response vigor, motivation and tonic dopamine. In *Advances in Neural Information Processing Systems 18 (NIPS 2005)*, pp. 1019–1026. MIT Press, Cambridge, MA.
- Niv, Y., Daw, N. D., Joel, D., Dayan, P. (2007). Tonic dopamine: opportunity costs and the control of response vigor. *Psychopharmacology*, 191(3):507–520.
- Niv, Y., Joel, D., Dayan, P. (2006). A normative perspective on motivation. *Trends in Cognitive Sciences*, 10(8):375–381.
- Nouri, A., Littman, M. L. (2009). Multi-resolution exploration in continuous spaces. In *Advances in Neural Information Processing Systems 21 (NIPS 2008)*, pp. 1209–1216. Curran Associates, Inc.
- Nowé, A., Vrancx, P., Hauwere, Y.-M. D. (2012). Game theory and multi-agent reinforcement learning. In M. Wiering and M. van Otterlo (Eds.), *Reinforcement Learning: State-of-the-Art*, pp. 441–467. Springer-Verlag Berlin Heidelberg.
- Nutt, D. J., Lingford-Hughes, A., Erritzoe, D., Stokes, P. R. A. (2015). The dopamine theory of addiction: 40 years of highs and lows. *Nature Reviews Neuroscience*, 16(5):305–312.
- O'Doherty, J. P., Dayan, P., Friston, K., Critchley, H., Dolan, R. J. (2003). Temporal difference models and reward-related learning in the human brain. *Neuron*, 38(2):329–337.
- O'Doherty, J. P., Dayan, P., Schultz, J., Deichmann, R., Friston, K., Dolan, R. J. (2004). Dissociable roles of ventral and dorsal striatum in instrumental conditioning. *Science*, 304(5669):452–454.
- Ólafsdóttir, H. F., Barry, C., Saleem, A. B., Hassabis, D., Spiers, H. J. (2015). Hippocampal place cells construct reward related sequences through unexplored space. *Elife*, 4:e06063.
- Oh, J., Guo, X., Lee, H., Lewis, R. L., Singh, S. (2015). Action-conditional video prediction using deep networks in Atari games. In *Advances in Neural Information Processing Systems 28 (NIPS 2015)*, pp. 2845–2853. Curran Associates, Inc.
- Olds, J., Milner, P. (1954). Positive reinforcement produced by electrical stimulation of the septal area and other regions of rat brain. *Journal of Comparative and Physiological Psychology*, 47(6):419–427.
- O'Reilly, R. C., Frank, M. J. (2006). Making working memory work: A computational model of learning in the prefrontal cortex and basal ganglia. *Neural Computation*, 18(2):283–328.
- O'Reilly, R. C., Frank, M. J., Hazy, T. E., Watz, B. (2007). PVLN, the primary value and learned value Pavlovian learning algorithm. *Behavioral Neuroscience*, 121(1):31–49.
- Omohundro, S. M. (1987). Efficient algorithms with neural network behavior. Technical Report, Department of Computer Science, University of Illinois at Urbana-Champaign.

- Ormoneit, D., Sen, Š. (2002). Kernel-based reinforcement learning. *Machine Learning*, 49(2-3):161–178.
- Oudeyer, P.-Y., Kaplan, F. (2007). What is intrinsic motivation? A typology of computational approaches. *Frontiers in Neurorobotics*, 1:6.
- Oudeyer, P.-Y., Kaplan, F., Hafner, V. V. (2007). Intrinsic motivation systems for autonomous mental development. *IEEE Transactions on Evolutionary Computation*, 11(2):265–286.
- Padoa-Schioppa, C., Assad, J. A. (2006). Neurons in the orbitofrontal cortex encode economic value. *Nature*, 441(7090):223–226.
- Page, C. V. (1977). Heuristics for signature table analysis as a pattern recognition technique. *IEEE Transactions on Systems, Man, and Cybernetics*, 7(2):77–86.
- Pagnoni, G., Zink, C. F., Montague, P. R., Berns, G. S. (2002). Activity in human ventral striatum locked to errors of reward prediction. *Nature Neuroscience*, 5(2):97–98.
- Pan, W.-X., Schmidt, R., Wickens, J. R., Hyland, B. I. (2005). Dopamine cells respond to predicted events during classical conditioning: Evidence for eligibility traces in the reward-learning network. *The Journal of Neuroscience*, 25(26):6235–6242.
- Park, J., Kim, J., Kang, D. (2005). An RLS-based natural actor-critic algorithm for locomotion of a two-linked robot arm. *Computational Intelligence and Security*:65–72.
- Parks, P. C., Militzer, J. (1991). Improved allocation of weights for associative memory storage in learning control systems. In *IFAC Design Methods of Control Systems*, Zurich, Switzerland, pp. 507–512.
- Parr, R. (1988). *Hierarchical Control and Learning for Markov Decision Processes*. Ph.D. thesis, University of California, Berkeley.
- Parr, R., Li, L., Taylor, G., Painter-Wakefield, C., Littman, M. L. (2008). An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning. In *Proceedings of the 25th international conference on Machine learning*, pp. 752–759).
- Parr, R., Russell, S. (1995). Approximating optimal policies for partially observable stochastic domains. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pp. 1088–1094. Morgan Kaufmann.
- Pavlov, P. I. (1927). *Conditioned Reflexes*. Oxford University Press, London.
- Pawlak, V., Kerr, J. N. D. (2008). Dopamine receptor activation is required for corticostriatal spike-timing-dependent plasticity. *The Journal of Neuroscience*, 28(10):2435–2446.
- Pawlak, V., Wickens, J. R., Kirkwood, A., Kerr, J. N. D. (2010). Timing is not everything: neuromodulation opens the STDP gate. *Frontiers in Synaptic Neuroscience*, 2:146. doi:10.3389/fnsyn.2010.00146.
- Pearce, J. M., Hall, G. (1980). A model for Pavlovian learning: Variation in the effectiveness of conditioning but not unconditioned stimuli. *Psychological Review*, 87(6):532–552.
- Pearl, J. (1984). *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, Reading, MA.
- Pearl, J. (1995). Causal diagrams for empirical research. *Biometrika*, 82(4):669–688.
- Pecevski, D., Maass, W., Legenstein, R. A. (2008). Theoretical analysis of learning with reward-modulated spike-timing-dependent plasticity. In *Advances in Neural Information Processing Systems 20 (NIPS 2007)*, pp. 881–888. Curran Associates, Inc.
- Peng, J. (1993). *Efficient Dynamic Programming-Based Learning for Control*. Ph.D. thesis, Northeastern University, Boston MA.
- Peng, J. (1995). Efficient memory-based dynamic programming. In *Proceedings of the 12th International Conference on Machine Learning (ICML 1995)*, pp. 438–446.
- Peng, J., Williams, R. J. (1993). Efficient learning and planning within the Dyna framework.

- Adaptive Behavior*, 1(4):437–454.
- Peng, J., Williams, R. J. (1994). Incremental multi-step Q-learning. In *Proceedings of the 11th International Conference on Machine Learning (ICML 1994)*, pp. 226–232. Morgan Kaufmann, San Francisco.
- Peng, J., Williams, R. J. (1996). Incremental multi-step Q-learning. *Machine Learning*, 22(1):283–290.
- Perkins, T. J., Pendrith, M. D. (2002). On the existence of fixed points for Q-learning and Sarsa in partially observable domains. In *Proceedings of the 19th International Conference on Machine Learning (ICML 2002)*, pp. 490–497.
- Perkins, T. J., Precup, D. (2003). A convergent form of approximate policy iteration. In *Advances in Neural Information Processing Systems 15 (NIPS 2002)*, pp. 1627–1634. MIT Press, Cambridge, MA.
- Peters, J., Büchel, C. (2010). Neural representations of subjective reward value. *Behavioral Brain Research*, 213(2):135–141.
- Peters, J., Schaal, S. (2008). Natural actor–critic. *Neurocomputing*, 71(7):1180–1190.
- Peters, J., Vijayakumar, S., Schaal, S. (2005). Natural actor–critic. In *European Conference on Machine Learning*, pp. 280–291. Springer Berlin Heidelberg.
- Pezzulo, G., van der Meer, M. A. A., Lansink, C. S., Pennartz, C. M. A. (2014). Internally generated sequences in learning and executing goal-directed behavior. *Trends in Cognitive Science*, 18(12):647–657.
- Pfeiffer, B. E., Foster, D. J. (2013). Hippocampal place-cell sequences depict future paths to remembered goals. *Nature*, 497(7447):74–79.
- Phansalkar, V. V., Thathachar, M. A. L. (1995). Local and global optimization algorithms for generalized learning automata. *Neural Computation*, 7(5):950–973.
- Poggio, T., Girosi, F. (1989). A theory of networks for approximation and learning. A.I. Memo 1140. Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA.
- Poggio, T., Girosi, F. (1990). Regularization algorithms for learning that are equivalent to multilayer networks. *Science*, 247(4945):978–982.
- Polyak, B. T. (1990). New stochastic approximation type procedures. *Automat. i Telemekh*, 7(98–107):2 (in Russian).
- Polyak, B. T., Juditsky, A. B. (1992). Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 30(4):838–855.
- Powell, M. J. D. (1987). Radial basis functions for multivariate interpolation: A review. In J. C. Mason and M. G. Cox (Eds.), *Algorithms for Approximation*, pp. 143–167. Clarendon Press, Oxford.
- Powell, W. B. (2011). *Approximate Dynamic Programming: Solving the Curses of Dimensionality*, Second edition. John Wiley and Sons.
- Powers, W. T. (1973). *Behavior: The Control of Perception*. Aldine de Gruyter, Chicago. 2nd expanded edition 2005.
- Precup, D. (2000). *Temporal Abstraction in Reinforcement Learning*. Ph.D. thesis, University of Massachusetts, Amherst.
- Precup, D., Sutton, R. S., Dasgupta, S. (2001). Off-policy temporal-difference learning with function approximation. In *Proceedings of the 18th International Conference on Machine Learning (ICML 2001)*, pp. 417–424.
- Precup, D., Sutton, R. S., Paduraru, C., Koop, A., Singh, S. (2006). Off-policy learning with options and recognizers. In *Advances in Neural Information Processing Systems 18 (NIPS*

- 2005), pp. 1097–1104. MIT Press, Cambridge, MA.
- Precup, D., Sutton, R. S., Singh, S. (2000). Eligibility traces for off-policy policy evaluation. In *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pp. 759–766. Morgan Kaufmann.
- Puterman, M. L. (1994). *Markov Decision Problems*. Wiley, New York.
- Puterman, M. L., Shin, M. C. (1978). Modified policy iteration algorithms for discounted Markov decision problems. *Management Science*, 24(11):1127–1137.
- Quartz, S., Dayan, P., Montague, P. R., Sejnowski, T. J. (1992). Expectation learning in the brain using diffuse ascending connections. In *Society for Neuroscience Abstracts*, 18:1210.
- Randløv, J., Alstrøm, P. (1998). Learning to drive a bicycle using reinforcement learning and shaping. In *Proceedings of the 15th International Conference on Machine Learning (ICML 1998)*, pp. 463–471.
- Rangel, A., Camerer, C., Montague, P. R. (2008). A framework for studying the neurobiology of value-based decision making. *Nature Reviews Neuroscience*, 9(7):545–556.
- Rangel, A., Hare, T. (2010). Neural computations associated with goal-directed choice. *Current Opinion in Neurobiology*, 20(2):262–270.
- Rao, R. P., Sejnowski, T. J. (2001). Spike-timing-dependent Hebbian plasticity as temporal difference learning. *Neural Computation*, 13(10):2221–2237.
- Ratcliff, R. (1990). Connectionist models of recognition memory: Constraints imposed by learning and forgetting functions. *Psychological Review*, 97(2):285–308.
- Reddy, G., Celani, A., Sejnowski, T. J., Vergassola, M. (2016). Learning to soar in turbulent environments. *Proceedings of the National Academy of Sciences*, 113(33):E4877–E4884.
- Redish, D. A. (2004). Addiction as a computational process gone awry. *Science*, 306(5703):1944–1947.
- Reetz, D. (1977). Approximate solutions of a discounted Markovian decision process. *Bonner Mathematische Schriften*, 98:77–92.
- Rescorla, R. A., Wagner, A. R. (1972). A theory of Pavlovian conditioning: Variations in the effectiveness of reinforcement and nonreinforcement. In A. H. Black and W. F. Prokasy (Eds.), *Classical Conditioning II*, pp. 64–99. Appleton-Century-Crofts, New York.
- Revusky, S., Garcia, J. (1970). Learned associations over long delays. In G. Bower (Ed.), *The Psychology of Learning and Motivation*, v. 4, pp. 1–84. Academic Press, Inc., New York.
- Reynolds, J. N. J., Wickens, J. R. (2002). Dopamine-dependent plasticity of corticostriatal synapses. *Neural Networks*, 15(4):507–521.
- Ring, M. B. (in preparation). Representing knowledge as forecasts (and state as knowledge).
- Ripley, B. D. (2007). *Pattern Recognition and Neural Networks*. Cambridge University Press.
- Rixner, S. (2004). Memory controller optimizations for web servers. In *Proceedings of the 37th annual IEEE/ACM International Symposium on Microarchitecture*, p. 355–366. IEEE Computer Society.
- Robbins, H. (1952). Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 58:527–535.
- Robertie, B. (1992). Carbon versus silicon: Matching wits with TD-Gammon. *Inside Backgammon*, 2(2):14–22.
- Romo, R., Schultz, W. (1990). Dopamine neurons of the monkey midbrain: Contingencies of responses to active touch during self-initiated arm movements. *Journal of Neurophysiology*, 63(3):592–624.
- Rosenblatt, F. (1962). *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Spartan Books, Washington, DC.

- Ross, S. (1983). *Introduction to Stochastic Dynamic Programming*. Academic Press, New York.
- Ross, T. (1933). Machines that think. *Scientific American*, 148(4):206–208.
- Rubinstein, R. Y. (1981). *Simulation and the Monte Carlo Method*. Wiley, New York.
- Rumelhart, D. E., Hinton, G. E., Williams, R. J. (1986). Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland (Eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, vol. I, *Foundations*. Bradford/MIT Press, Cambridge, MA.
- Rummery, G. A. (1995). *Problem Solving with Reinforcement Learning*. Ph.D. thesis, University of Cambridge.
- Rummery, G. A., Niranjan, M. (1994). On-line Q-learning using connectionist systems. Technical Report CUED/F-INFENG/TR 166. Engineering Department, Cambridge University.
- Ruppert, D. (1988). Efficient estimations from a slowly convergent Robbins-Monro process. Cornell University Operations Research and Industrial Engineering Technical Report No. 781.
- Russell, S., Norvig, P. (2009). *Artificial Intelligence: A Modern Approach*, 3rd edition. Prentice-Hall, Englewood Cliffs, NJ.
- Russo, D. J., Van Roy, B., Kazerouni, A., Osband, I., Wen, Z. (2018). A tutorial on Thompson sampling, *Foundations and Trends in Machine Learning*. ArXiv:1707.02038.
- Rust, J. (1996). Numerical dynamic programming in economics. In H. Amman, D. Kendrick, and J. Rust (Eds.), *Handbook of Computational Economics*, pp. 614–722. Elsevier, Amsterdam.
- Saddoris, M. P., Cacciapaglia, F., Wightman, R. M., Carelli, R. M. (2015). Differential dopamine release dynamics in the nucleus accumbens core and shell reveal complementary signals for error prediction and incentive motivation. *The Journal of Neuroscience*, 35(33):11572–11582.
- Saksida, L. M., Raymond, S. M., Touretzky, D. S. (1997). Shaping robot behavior using principles from instrumental conditioning. *Robotics and Autonomous Systems*, 22(3):231–249.
- Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal on Research and Development*, 3(3), 210–229.
- Samuel, A. L. (1967). Some studies in machine learning using the game of checkers. II—Recent progress. *IBM Journal on Research and Development*, 11(6):601–617.
- Schaal, S., Atkeson, C. G. (1994). Robot juggling: Implementation of memory-based learning. *IEEE Control Systems*, 14(1):57–71.
- Schmajuk, N. A. (2008). Computational models of classical conditioning. *Scholarpedia*, 3(3):1664.
- Schmidhuber, J. (1991a). Curious model-building control systems. In *Proceedings of the IEEE International Joint Conference on Neural Networks*, pp. 1458–1463. IEEE.
- Schmidhuber, J. (1991b). A possibility for implementing curiosity and boredom in model-building neural controllers. In *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior*, pp. 222–227. MIT Press, Cambridge, MA.
- Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks*, 6:85–117.
- Schmidhuber, J., Storck, J., Hochreiter, S. (1994). Reinforcement driven information acquisition in nondeterministic environments. Technical report, Fakultät für Informatik, Technische Universität München, München, Germany.
- Schraudolph, N. N. (1999). Local gain adaptation in stochastic gradient descent. In *Proceedings of the International Conference on Artificial Neural Networks*, pp. 569–574. IEEE, London.
- Schraudolph, N. N. (2002). Fast curvature matrix-vector products for second-order gradient descent. *Neural Computation*, 14(7):1723–1738.

- Schraudolph, N. N., Yu, J., Aberdeen, D. (2006). Fast online policy gradient learning with SMD gain vector adaptation. In *Advances in Neural Information Processing Systems*, pp. 1185–1192.
- Schulman, J., Chen, X., Abbeel, P. (2017). Equivalence between policy gradients and soft Q-Learning. ArXiv:1704.06440.
- Schultz, D. G., Melsa, J. L. (1967). *State Functions and Linear Control Systems*. McGraw-Hill, New York.
- Schultz, W. (1998). Predictive reward signal of dopamine neurons. *Journal of Neurophysiology*, 80(1):1–27.
- Schultz, W., Apicella, P., Ljungberg, T. (1993). Responses of monkey dopamine neurons to reward and conditioned stimuli during successive steps of learning a delayed response task. *The Journal of Neuroscience*, 13(3):900–913.
- Schultz, W., Dayan, P., Montague, P. R. (1997). A neural substrate of prediction and reward. *Science*, 275(5306):1593–1598.
- Schultz, W., Romo, R. (1990). Dopamine neurons of the monkey midbrain: contingencies of responses to stimuli eliciting immediate behavioral reactions. *Journal of Neurophysiology*, 63(3):607–624.
- Schultz, W., Romo, R., Ljungberg, T., Mirenowicz, J., Hollerman, J. R., Dickinson, A. (1995). Reward-related signals carried by dopamine neurons. In J. C. Houk, J. L. Davis, and D. G. Beiser (Eds.), *Models of Information Processing in the Basal Ganglia*, pp. 233–248. MIT Press, Cambridge, MA.
- Schwartz, A. (1993). A reinforcement learning method for maximizing undiscounted rewards. In *Proceedings of the 10th International Conference on Machine Learning (ICML 1993)*, pp. 298–305. Morgan Kaufmann.
- Schweitzer, P. J., Seidmann, A. (1985). Generalized polynomial approximations in Markovian decision processes. *Journal of Mathematical Analysis and Applications*, 110(2):568–582.
- Selfridge, O. G. (1978). Tracking and trailing: Adaptation in movement strategies. Technical report, Bolt Beranek and Newman, Inc. Unpublished report.
- Selfridge, O. G. (1984). Some themes and primitives in ill-defined systems. In O. G. Selfridge, E. L. Risland, and M. A. Arbib (Eds.), *Adaptive Control of Ill-Defined Systems*, pp. 21–26. Plenum Press, NY. Proceedings of the NATO Advanced Research Institute on Adaptive Control of Ill-defined Systems, NATO Conference Series II, Systems Science, Vol. 16.
- Selfridge, O. J., Sutton, R. S., Barto, A. G. (1985). Training and tracking in robotics. In A. Joshi (Ed.), *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pp. 670–672. Morgan Kaufmann.
- Seo, H., Barraclough, D., Lee, D. (2007). Dynamic signals related to choices and outcomes in the dorsolateral prefrontal cortex. *Cerebral Cortex*, 17(suppl 1):110–117.
- Seung, H. S. (2003). Learning in spiking neural networks by reinforcement of stochastic synaptic transmission. *Neuron*, 40(6):1063–1073.
- Shah, A. (2012). Psychological and neuroscientific connections with reinforcement learning. In M. Wiering and M. van Otterlo (Eds.), *Reinforcement Learning: State-of-the-Art*, pp. 507–537. Springer-Verlag Berlin Heidelberg.
- Shannon, C. E. (1950). Programming a computer for playing chess. *Philosophical Magazine and Journal of Science*, 41(314):256–275.
- Shannon, C. E. (1951). Presentation of a maze-solving machine. In H. V. Forester (Ed.), *Cybernetics. Transactions of the Eighth Conference*, pp. 173–180. Josiah Macy Jr. Foundation.
- Shannon, C. E. (1952). “Theseus” maze-solving mouse. <http://cyberneticzoo.com/mazesolvers/1952->

- theseus-maze-solving-mouse--claude-shannon-american/.
- Shelton, C. R. (2001). *Importance Sampling for Reinforcement Learning with Multiple Objectives*. Ph.D. thesis, Massachusetts Institute of Technology, Cambridge MA.
- Shepard, D. (1968). A two-dimensional interpolation function for irregularly-spaced data. In *Proceedings of the 23rd ACM National Conference*, pp. 517–524. ACM, New York.
- Sherman, J., Morrison, W. J. (1949). Adjustment of an inverse matrix corresponding to changes in the elements of a given column or a given row of the original matrix (abstract). *Annals of Mathematical Statistics*, 20(4):621.
- Shewchuk, J., Dean, T. (1990). Towards learning time-varying functions with high input dimensionality. In *Proceedings of the Fifth IEEE International Symposium on Intelligent Control*, pp. 383–388. IEEE Computer Society Press, Los Alamitos, CA.
- Shimansky, Y. P. (2009). Biologically plausible learning in neural networks: a lesson from bacterial chemotaxis. *Biological Cybernetics*, 101(5-6):379–385.
- Si, J., Barto, A., Powell, W., Wunsch, D. (Eds.) (2004). *Handbook of Learning and Approximate Dynamic Programming*. John Wiley and Sons.
- Silver, D. (2009). *Reinforcement Learning and Simulation Based Search in the Game of Go*. Ph.D. thesis, University of Alberta, Edmonton.
- Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489.
- Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., Riedmiller, M. (2014). Deterministic policy gradient algorithms. In *Proceedings of the 31st International Conference on Machine Learning (ICML 2014)*, pp. 387–395.
- Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, L., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., Hassabis, D. (2017a). Mastering the game of Go without human knowledge. *Nature*, 550(7676):354–359.
- Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simoyan, K., Hassabis, D. (2017b). Mastering chess and shogi by self-play with a general reinforcement learning algorithm. ArXiv:1712.01815.
- Simsek, Ö., Algorta, S., Kothiyal, A. (2016). Why most decisions are easy in tetris—And perhaps in other sequential decision problems, as well. In *Proceedings of the 33rd International Conference on Machine Learning (ICML 2016)*, pp. 1757–1765.
- Simon, H. (2000). Lecture at the Earthware Symposium, Carnegie Mellon University. <https://www.youtube.com/8DBjc>.
- Singh, S. P. (1992a). Reinforcement learning with a hierarchy of abstract models. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, pp. 202–207. AAAI/MIT Press, Menlo Park, CA.
- Singh, S. P. (1992b). Scaling reinforcement learning algorithms by learning variable temporal resolution models. In *Proceedings of the 9th International Workshop on Machine Learning*, pp. 406–415. Morgan Kaufmann.
- Singh, S. P. (1993). *Learning to Solve Markovian Decision Processes*. Ph.D. thesis, University of Massachusetts, Amherst.
- Singh, S. P. (Ed.) (2002). Special double issue on reinforcement learning, *Machine Learning*, 49(2-3).

- Singh, S., Barto, A. G., Chentanez, N. (2005). Intrinsically motivated reinforcement learning. In *Advances in Neural Information Processing Systems 17 (NIPS 2004)*, pp. 1281–1288. MIT Press, Cambridge, MA.
- Singh, S. P., Bertsekas, D. (1997). Reinforcement learning for dynamic channel allocation in cellular telephone systems. In *Advances in Neural Information Processing Systems 9 (NIPS 1996)*, pp. 974–980. MIT Press, Cambridge, MA.
- Singh, S. P., Jaakkola, T., Jordan, M. I. (1994). Learning without state-estimation in partially observable Markovian decision problems. In *Proceedings of the 11th International Conference on Machine Learning (ICML 1994)*, pp. 284–292. Morgan Kaufmann.
- Singh, S., Jaakkola, T., Littman, M. L., Szepesvri, C. (2000). Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning*, 38(3):287–308.
- Singh, S. P., Jaakkola, T., Jordan, M. I. (1995). Reinforcement learning with soft state aggregation. In *Advances in Neural Information Processing Systems 7 (NIPS 1994)*, pp. 359–368. MIT Press, Cambridge, MA.
- Singh, S., Lewis, R. L., Barto, A. G. (2009). Where do rewards come from? In N. Taatgen and H. van Rijn (Eds.), *Proceedings of the 31st Annual Conference of the Cognitive Science Society*, pp. 2601–2606. Cognitive Science Society.
- Singh, S., Lewis, R. L., Barto, A. G., Sorg, J. (2010). Intrinsically motivated reinforcement learning: An evolutionary perspective. *IEEE Transactions on Autonomous Mental Development*, 2(2):70–82. Special issue on Active Learning and Intrinsically Motivated Exploration in Robots: Advances and Challenges.
- Singh, S. P., Sutton, R. S. (1996). Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22(1-3):123–158.
- Skinner, B. F. (1938). *The Behavior of Organisms: An Experimental Analysis*. Appleton-Century, New York.
- Skinner, B. F. (1958). Reinforcement today. *American Psychologist*, 13(3):94–99.
- Skinner, B. F. (1963). Operant behavior. *American Psychologist*, 18(8):503–515.
- Sofge, D. A., White, D. A. (1992). Applied learning: Optimal control for manufacturing. In D. A. White and D. A. Sofge (Eds.), *Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches*, pp. 259–281. Van Nostrand Reinhold, New York.
- Sorg, J. D. (2011). *The Optimal Reward Problem: Designing Effective Reward for Bounded Agents*. Ph.D. thesis, University of Michigan, Ann Arbor.
- Sorg, J., Lewis, R. L., Singh, S. P. (2010). Reward design via online gradient ascent. In *Advances in Neural Information Processing Systems 23 (NIPS 2010)*, pp. 2190–2198. Curran Associates, Inc.
- Sorg, J., Singh, S. (2010). Linear options. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems*, pp. 31–38.
- Sorg, J., Singh, S., Lewis, R. (2010). Internal rewards mitigate agent boundedness. In *Proceedings of the 27th International Conference on Machine Learning (ICML 2010)*, pp. 1007–1014.
- Spence, K. W. (1947). The role of secondary reinforcement in delayed reward learning. *Psychological Review*, 54(1):1–8.
- Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958.
- Staddon, J. E. R. (1983). *Adaptive Behavior and Learning*. Cambridge University Press.
- Stanfill, C., Waltz, D. (1986). Toward memory-based reasoning. *Communications of the ACM*, 29(12):1213–1228.

- Steinberg, E. E., Keiflin, R., Boivin, J. R., Witten, I. B., Deisseroth, K., Janak, P. H. (2013). A causal link between prediction errors, dopamine neurons and learning. *Nature Neuroscience*, 16(7):966–973.
- Sterling, P., Laughlin, S. (2015). *Principles of Neural Design*. MIT Press, Cambridge, MA.
- Sternberg, S. (1963). Stochastic learning theory. In: *Handbook of Mathematical Psychology*, Volume II, R. D. Luce, R. R. Bush, and E. Galanter (Eds.). John Wiley & Sons.
- Sugiyama, M., Hachiya, H., Morimura, T. (2013). *Statistical Reinforcement Learning: Modern Machine Learning Approaches*. Chapman & Hall/CRC.
- Suri, R. E., Bargas, J., Arbib, M. A. (2001). Modeling functions of striatal dopamine modulation in learning and planning. *Neuroscience*, 103(1):65–85.
- Suri, R. E., Schultz, W. (1998). Learning of sequential movements by neural network model with dopamine-like reinforcement signal. *Experimental Brain Research*, 121(3):350–354.
- Suri, R. E., Schultz, W. (1999). A neural network model with dopamine-like reinforcement signal that learns a spatial delayed response task. *Neuroscience*, 91(3):871–890.
- Sutton, R. S. (1978a). Learning theory support for a single channel theory of the brain. Unpublished report.
- Sutton, R. S. (1978b). Single channel theory: A neuronal theory of learning. *Brain Theory Newsletter*, 4:72–75. Center for Systems Neuroscience, University of Massachusetts, Amherst, MA.
- Sutton, R. S. (1978c). *A unified theory of expectation in classical and instrumental conditioning*. Bachelors thesis, Stanford University.
- Sutton, R. S. (1984). *Temporal Credit Assignment in Reinforcement Learning*. Ph.D. thesis, University of Massachusetts, Amherst.
- Sutton, R. S. (1988). Learning to predict by the method of temporal differences. *Machine Learning*, 3(1):9–44 (important erratum p. 377).
- Sutton, R. S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the 7th International Workshop on Machine Learning*, pp. 216–224. Morgan Kaufmann.
- Sutton, R. S. (1991a). Dyna, an integrated architecture for learning, planning, and reacting. *SIGART Bulletin*, 2(4):160–163. ACM, New York.
- Sutton, R. S. (1991b). Planning by incremental dynamic programming. In *Proceedings of the 8th International Workshop on Machine Learning*, pp. 353–357. Morgan Kaufmann.
- Sutton, R. S. (Ed.) (1992a). *Reinforcement Learning*. Kluwer Academic Press. Reprinting of a special double issue on reinforcement learning, *Machine Learning*, 8(3-4).
- Sutton, R.S. (1992b). Adapting bias by gradient descent: An incremental version of delta-bar-delta. *Proceedings of the Tenth National Conference on Artificial Intelligence*, pp. 171–176, MIT Press.
- Sutton, R.S. (1992c). Gain adaptation beats least squares? *Proceedings of the Seventh Yale Workshop on Adaptive and Learning Systems*, pp. 161–166, Yale University, New Haven, CT.
- Sutton, R. S. (1995a). TD models: Modeling the world at a mixture of time scales. In *Proceedings of the 12th International Conference on Machine Learning (ICML 1995)*, pp. 531–539. Morgan Kaufmann.
- Sutton, R. S. (1995b). On the virtues of linear learning and trajectory distributions. In *Proceedings of the Workshop on Value Function Approximation at The 12th International Conference on Machine Learning (ICML 1995)*.
- Sutton, R. S. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems 8 (NIPS 1995)*,

- pp. 1038–1044. MIT Press, Cambridge, MA.
- Sutton, R. S. (2009). The grand challenge of predictive empirical abstract knowledge. *Working Notes of the IJCAI-09 Workshop on Grand Challenges for Reasoning from Experiences*.
- Sutton, R. S. (2015a) Introduction to reinforcement learning with function approximation. Tutorial at the Conference on Neural Information Processing Systems (NIPS), Montreal, December 7, 2015.
- Sutton, R. S. (2015b) True online Emphatic TD(λ): Quick reference and implementation guide. ArXiv:1507.07147. Code is available in Python and C++ by downloading the source files of this arXiv paper as a zip archive.
- Sutton, R. S., Barto, A. G. (1981a). Toward a modern theory of adaptive networks: Expectation and prediction. *Psychological Review*, 88(2):135–170.
- Sutton, R. S., Barto, A. G. (1981b). An adaptive network that constructs and uses an internal model of its world. *Cognition and Brain Theory*, 3:217–246.
- Sutton, R. S., Barto, A. G. (1987). A temporal-difference model of classical conditioning. In *Proceedings of the Ninth Annual Conference of the Cognitive Science Society*, pp. 355–378. Erlbaum, Hillsdale, NJ.
- Sutton, R. S., Barto, A. G. (1990). Time-derivative models of Pavlovian reinforcement. In M. Gabriel and J. Moore (Eds.), *Learning and Computational Neuroscience: Foundations of Adaptive Networks*, pp. 497–537. MIT Press, Cambridge, MA.
- Sutton, R. S., Maei, H. R., Precup, D., Bhatnagar, S., Silver, D., Szepesvári, Cs., Wiewiora, E. (2009a). Fast gradient-descent methods for temporal-difference learning with linear function approximation. In *Proceedings of the 26th International Conference on Machine Learning (ICML 2009)*, pp. 993–1000. ACM, New York.
- Sutton, R. S., Szepesvári, Cs., Maei, H. R. (2009b). A convergent $O(d^2)$ temporal-difference algorithm for off-policy learning with linear function approximation. In *Advances in Neural Information Processing Systems 21 (NIPS 2008)*, pp. 1609–1616. Curran Associates, Inc.
- Sutton, R. S., Mahmood, A. R., Precup, D., van Hasselt, H. (2014). A new Q(λ) with interim forward view and Monte Carlo equivalence. In *Proceedings of the International Conference on Machine Learning*, 31. JMLR W&CP 32(2).
- Sutton, R. S., Mahmood, A. R., White, M. (2016). An emphatic approach to the problem of off-policy temporal-difference learning. *Journal of Machine Learning Research*, 17(73):1–29.
- Sutton, R. S., McAllester, D. A., Singh, S. P., Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems 12 (NIPS 1999)*, pp. 1057–1063. MIT Press, Cambridge, MA.
- Sutton, R. S., Modayil, J., Delp, M., Degris, T., Pilarski, P. M., White, A., Precup, D. (2011). Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *Proceedings of the Tenth International Conference on Autonomous Agents and Multiagent Systems*, pp. 761–768, Taipei, Taiwan.
- Sutton, R. S., Pinette, B. (1985). The learning of world models by connectionist networks. In *Proceedings of the Seventh Annual Conference of the Cognitive Science Society*, pp. 54–64.
- Sutton, R. S., Precup, D., Singh, S. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2):181–211.
- Sutton, R. S., Rafols, E., Koop, A. (2006). Temporal abstraction in temporal-difference networks. In *Advances in neural information processing systems*, pp. 1313–1320.
- Sutton, R. S., Singh, S. P., McAllester, D. A. (2000). Comparing policy-gradient algorithms. Unpublished manuscript.
- Sutton, R. S., Szepesvári, Cs., Geramifard, A., Bowling, M., (2008). Dyna-style planning with

- linear function approximation and prioritized sweeping. In *Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence*, pp. 528–536.
- Sutton, R. S., Tanner, B. (2005). Temporal-difference networks. In *Advances in Neural Information Processing Systems 17*, p. 1377–1384.
- Szepesvári, Cs. (2010). Algorithms for reinforcement learning. In *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 4(1):1–103. Morgan and Claypool.
- Szita, I. (2012). Reinforcement learning in games. In M. Wiering and M. van Otterlo (Eds.), *Reinforcement Learning: State-of-the-Art*, pp. 539–577. Springer-Verlag Berlin Heidelberg.
- Tadepalli, P., Ok, D. (1994). H-learning: A reinforcement learning method to optimize undiscounted average reward. Technical Report 94-30-01. Oregon State University, Computer Science Department, Corvallis.
- Tadepalli, P., Ok, D. (1996). Scaling up average reward reinforcement learning by approximating the domain models and the value function. In *Proceedings of the 13th International Conference on Machine Learning (ICML 1996)*, pp. 471–479.
- Takahashi, Y., Schoenbaum, G., and Niv, Y. (2008). Silencing the critics: Understanding the effects of cocaine sensitization on dorsolateral and ventral striatum in the context of an actor/critic model. *Frontiers in Neuroscience*, 2(1):86–99.
- Tambe, M., Newell, A., Rosenbloom, P. S. (1990). The problem of expensive chunks and its solution by restricting expressiveness. *Machine Learning*, 5(3):299–348.
- Tan, M. (1991). Learning a cost-sensitive internal representation for reinforcement learning. In L. A. Birnbaum and G. C. Collins (Eds.), *Proceedings of the 8th International Workshop on Machine Learning*, pp. 358–362. Morgan Kaufmann.
- Tanner, B. (2006). *Temporal-Difference Networks*. MSc thesis, University of Alberta.
- Taylor, G., Parr, R. (2009). Kernelized value function approximation for reinforcement learning. In *Proceedings of the 26th International Conference on Machine Learning (ICML 2009)*, pp. 1017–1024. ACM, New York.
- Taylor, M. E., Stone, P. (2009). Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10:1633–1685.
- Tesauro, G. (1986). Simple neural models of classical conditioning. *Biological Cybernetics*, 55(2-3):187–200.
- Tesauro, G. (1992). Practical issues in temporal difference learning. *Machine Learning*, 8(3-4):257–277.
- Tesauro, G. (1994). TD-Gammon, a self-teaching backgammon program, achieves master-level play. *Neural Computation*, 6(2):215–219.
- Tesauro, G. (1995). Temporal difference learning and TD-Gammon. *Communications of the ACM*, 38(3):58–68.
- Tesauro, G. (2002). Programming backgammon using self-teaching neural nets. *Artificial Intelligence*, 134(1-2):181–199.
- Tesauro, G., Galperin, G. R. (1997). On-line policy improvement using Monte-Carlo search. In *Advances in Neural Information Processing Systems 9 (NIPS 1996)*, pp. 1068–1074. MIT Press, Cambridge, MA.
- Tesauro, G., Gondek, D. C., Lechner, J., Fan, J., Prager, J. M. (2012). Simulation, learning, and optimization techniques in Watson’s game strategies. *IBM Journal of Research and Development*, 56(3-4):16–1–16–11.
- Tesauro, G., Gondek, D. C., Lenchner, J., Fan, J., Prager, J. M. (2013). Analysis of WATSON’s strategies for playing Jeopardy! *Journal of Artificial Intelligence Research*, 47:205–251.
- Tham, C. K. (1994). *Modular On-Line Function Approximation for Scaling up Reinforcement*

- Learning*. Ph.D. thesis, University of Cambridge.
- Thathachar, M. A. L., Sastry, P. S. (1985). A new approach to the design of reinforcement schemes for learning automata. *IEEE Transactions on Systems, Man, and Cybernetics*, 15(1):168–175.
- Thathachar, M., Sastry, P. S. (2002). Varieties of learning automata: an overview. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 36(6):711–722.
- Thathachar, M., Sastry, P. S. (2011). *Networks of Learning Automata: Techniques for Online Stochastic Optimization*. Springer Science & Business Media.
- Theocharous, G., Thomas, P. S., Ghavamzadeh, M. (2015). Personalized ad recommendation for life-time value optimization guarantees. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence*. AAAI Press, Palo Alto, CA.
- Thistlthwaite, D. (1951). A critical review of latent learning and related experiments. *Psychological Bulletin*, 48(2):97–129.
- Thomas, P. S. (2014). Bias in natural actor–critic algorithms. In *Proceedings of the 31st International Conference on Machine Learning, JMLR W&CP* 32(1), pp. 441–448.
- Thomas, P. S. (2015). *Safe Reinforcement Learning*. Ph.D. thesis, University of Massachusetts, Amherst.
- Thomas, P. S., Brunskill, E. (2017). Policy gradient methods for reinforcement learning with function approximation and action-dependent baselines. ArXiv:1706.06643.
- Thomas, P. S., Theocharous, G., Ghavamzadeh, M. (2015). High-confidence off-policy evaluation. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, pp. 3000–3006. AAAI Press, Menlo Park, CA.
- Thompson, W. R. (1933). On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3-4):285–294.
- Thompson, W. R. (1934). On the theory of apportionment. *American Journal of Mathematics*, 57: 450–457.
- Thon, M. (2017). *Spectral Learning of Sequential Systems*. Ph.D. thesis, Jacobs University Bremen.
- Thon, M., Jaeger, H. (2015). Links between multiplicity automata, observable operator models and predictive state representations: a unified learning framework. *The Journal of Machine Learning Research*, 16(1):103–147.
- Thorndike, E. L. (1898). Animal intelligence: An experimental study of the associative processes in animals. *The Psychological Review, Series of Monograph Supplements*, II(4).
- Thorndike, E. L. (1911). *Animal Intelligence*. Hafner, Darien, CT.
- Thorp, E. O. (1966). *Beat the Dealer: A Winning Strategy for the Game of Twenty-One*. Random House, New York.
- Tian, T. (in preparation) *An Empirical Study of Sliding-Step Methods in Temporal Difference Learning*. M.Sc thesis, University of Alberta, Edmonton.
- Tieleman, T., Hinton, G. (2012). Lecture 6.5–RMSProp. COURSERA: Neural networks for machine learning 4.2:26–31.
- Tolman, E. C. (1932). *Purposive Behavior in Animals and Men*. Century, New York.
- Tolman, E. C. (1948). Cognitive maps in rats and men. *Psychological Review*, 55(4):189–208.
- Tsai, H.-S., Zhang, F., Adamantidis, A., Stuber, G. D., Bonci, A., de Lecea, L., Deisseroth, K. (2009). Phasic firing in dopaminergic neurons is sufficient for behavioral conditioning. *Science*, 324(5930):1080–1084.
- Tsetlin, M. L. (1973). *Automaton Theory and Modeling of Biological Systems*. Academic Press, New York.

- Tsitsiklis, J. N. (1994). Asynchronous stochastic approximation and Q-learning. *Machine Learning*, 16(3):185–202.
- Tsitsiklis, J. N. (2002). On the convergence of optimistic policy iteration. *Journal of Machine Learning Research*, 3:59–72.
- Tsitsiklis, J. N., Van Roy, B. (1996). Feature-based methods for large scale dynamic programming. *Machine Learning*, 22(1-3):59–94.
- Tsitsiklis, J. N., Van Roy, B. (1997). An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42(5):674–690.
- Tsitsiklis, J. N., Van Roy, B. (1999). Average cost temporal-difference learning. *Automatica*, 35(11):1799–1808.
- Turing, A. M. (1948). Intelligent machinery. In B. Jack Copeland (Ed.) (2004), *The Essential Turing*, pp. 410–432. Oxford University Press, Oxford.
- Ungar, L. H. (1990). A bioreactor benchmark for adaptive network-based process control. In W. T. Miller, R. S. Sutton, and P. J. Werbos (Eds.), *Neural Networks for Control*, pp. 387–402. MIT Press, Cambridge, MA.
- Unnikrishnan, K. P., Venugopal, K. P. (1994). Alopex: A correlation-based learning algorithm for feedforward and recurrent neural networks. *Neural Computation*, 6(3): 469–490.
- Urbanczik, R., Senn, W. (2009). Reinforcement learning in populations of spiking neurons. *Nature neuroscience*, 12(3):250–252.
- Urbanowicz, R. J., Moore, J. H. (2009). Learning classifier systems: A complete introduction, review, and roadmap. *Journal of Artificial Evolution and Applications*. 10.1155/2009/736398.
- Valentin, V. V., Dickinson, A., O'Doherty, J. P. (2007). Determining the neural substrates of goal-directed learning in the human brain. *The Journal of Neuroscience*, 27(15):4019–4026.
- van Hasselt, H. (2010). Double Q-learning. In *Advances in Neural Information Processing Systems 23 (NIPS 2010)*, pp. 2613–2621. Curran Associates, Inc.
- van Hasselt, H. (2011). *Insights in Reinforcement Learning: Formal Analysis and Empirical Evaluation of Temporal-difference Learning*. SIKS dissertation series number 2011-04.
- van Hasselt, H. (2012). Reinforcement learning in continuous state and action spaces. In M. Wiering and M. van Otterlo (Eds.), *Reinforcement Learning: State-of-the-Art*, pp. 207–251. Springer-Verlag Berlin Heidelberg.
- van Hasselt, H., Sutton, R. S. (2015). Learning to predict independent of span. ArXiv:1508.04582.
- Van Roy, B., Bertsekas, D. P., Lee, Y., Tsitsiklis, J. N. (1997). A neuro-dynamic programming approach to retailer inventory management. In *Proceedings of the 36th IEEE Conference on Decision and Control*, Vol. 4, pp. 4052–4057.
- van Seijen, H. (2011). Reinforcement Learning under Space and Time Constraints. University of Amsterdam PhD thesis. Hague: TNO.
- van Seijen, H. (2016). Effective multi-step temporal-difference learning for non-linear function approximation. ArXiv:1608.05151.
- van Seijen, H., Sutton, R. S. (2013). Efficient planning in MDPs by small backups. In: *Proceedings of the 30th International Conference on Machine Learning (ICML 2013)*, pp. 361–369.
- van Seijen, H., Sutton, R. S. (2014). True online TD(λ). In *Proceedings of the 31st International Conference on Machine Learning (ICML 2014)*, pp. 692–700. JMLR W&CP 32(1),
- van Seijen, H., Mahmood, A. R., Pilarski, P. M., Machado, M. C., Sutton, R. S. (2016). True online temporal-difference learning. *Journal of Machine Learning Research*, 17(145):1–40.
- van Seijen, H., Van Hasselt, H., Whiteson, S., Wiering, M. (2009). A theoretical and empirical analysis of Expected Sarsa. In *IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, pp. 177–184.

- van Seijen, H., Whiteson, S., van Hasselt, H., Wiering, M. (2011). Exploiting best-match equations for efficient reinforcement learning. *Journal of Machine Learning Research* 12:2045–2094.
- Varga, R. S. (1962). *Matrix Iterative Analysis*. Englewood Cliffs, NJ: Prentice-Hall.
- Vasilaki, E., Frémaux, N., Urbanczik, R., Senn, W., Gerstner, W. (2009). Spike-based reinforcement learning in continuous state and action space: when policy gradient methods fail. *PLoS Computational Biology*, 5(12).
- Viswanathan, R., Narendra, K. S. (1974). Games of stochastic automata. *IEEE Transactions on Systems, Man, and Cybernetics*, 4(1):131–135.
- Wagner, A. R. (2008). Evolution of an elemental theory of Pavlovian conditioning. *Learning & Behavior*, 36(3):253–265.
- Walter, W. G. (1950). An imitation of life. *Scientific American*, 182(5):42–45.
- Walter, W. G. (1951). A machine that learns. *Scientific American*, 185(2):60–63.
- Waltz, M. D., Fu, K. S. (1965). A heuristic approach to reinforcement learning control systems. *IEEE Transactions on Automatic Control*, 10(4):390–398.
- Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards*. Ph.D. thesis, University of Cambridge.
- Watkins, C. J. C. H., Dayan, P. (1992). Q-learning. *Machine Learning*, 8(3-4):279–292.
- Werbos, P. J. (1977). Advanced forecasting methods for global crisis warning and models of intelligence. *General Systems Yearbook*, 22(12):25–38.
- Werbos, P. J. (1982). Applications of advances in nonlinear sensitivity analysis. In R. F. Drenick and F. Kozin (Eds.), *System Modeling and Optimization*, pp. 762–770. Springer-Verlag.
- Werbos, P. J. (1987). Building and understanding adaptive systems: A statistical/numerical approach to factory automation and brain research. *IEEE Transactions on Systems, Man, and Cybernetics*, 17(1):7–20.
- Werbos, P. J. (1988). Generalization of back propagation with applications to a recurrent gas market model. *Neural Networks*, 1(4):339–356.
- Werbos, P. J. (1989). Neural networks for control and system identification. In *Proceedings of the 28th Conference on Decision and Control*, pp. 260–265. IEEE Control Systems Society.
- Werbos, P. J. (1992). Approximate dynamic programming for real-time control and neural modeling. In D. A. White and D. A. Sofge (Eds.), *Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches*, pp. 493–525. Van Nostrand Reinhold, New York.
- Werbos, P. J. (1994). *The Roots of Backpropagation: From Ordered Derivatives to Neural Networks and Political Forecasting* (Vol. 1). John Wiley and Sons.
- Wiering, M., Van Otterlo, M. (2012). *Reinforcement Learning: State-of-the-Art*. Springer-Verlag Berlin Heidelberg.
- White, A. (2015). *Developing a Predictive Approach to Knowledge*. Ph.D. thesis, University of Alberta, Edmonton.
- White, D. J. (1969). *Dynamic Programming*. Holden-Day, San Francisco.
- White, D. J. (1985). Real applications of Markov decision processes. *Interfaces*, 15(6):73–83.
- White, D. J. (1988). Further real applications of Markov decision processes. *Interfaces*, 18(5):55–61.
- White, D. J. (1993). A survey of applications of Markov decision processes. *Journal of the Operational Research Society*, 44(11):1073–1096.
- White, A., White, M. (2016). Investigating practical linear temporal difference learning. In *Proceedings of the 2016 International Conference on Autonomous Agents and Multiagent Systems*, pp. 494–502.

- Whitehead, S. D., Ballard, D. H. (1991). Learning to perceive and act by trial and error. *Machine Learning*, 7(1):45–83.
- Whitt, W. (1978). Approximations of dynamic programs I. *Mathematics of Operations Research*, 3(3):231–243.
- Whittle, P. (1982). *Optimization over Time*, vol. 1. Wiley, New York.
- Whittle, P. (1983). *Optimization over Time*, vol. 2. Wiley, New York.
- Wickens, J., Kötter, R. (1995). Cellular models of reinforcement. In J. C. Houk, J. L. Davis and D. G. Beiser (Eds.), *Models of Information Processing in the Basal Ganglia*, pp. 187–214. MIT Press, Cambridge, MA.
- Widrow, B., Gupta, N. K., Maitra, S. (1973). Punish/reward: Learning with a critic in adaptive threshold systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 3(5):455–465.
- Widrow, B., Hoff, M. E. (1960). Adaptive switching circuits. In *1960 WESCON Convention Record Part IV*, pp. 96–104. Institute of Radio Engineers, New York. Reprinted in J. A. Anderson and E. Rosenfeld, *Neurocomputing: Foundations of Research*, pp. 126–134. MIT Press, Cambridge, MA, 1988.
- Widrow, B., Smith, F. W. (1964). Pattern-recognizing control systems. In J. T. Tou and R. H. Wilcox (Eds.), *Computer and Information Sciences*, pp. 288–317. Spartan, Washington, DC.
- Widrow, B., Stearns, S. D. (1985). *Adaptive Signal Processing*. Prentice-Hall, Englewood Cliffs, NJ.
- Wiener, N. (1964). *God and Golem, Inc: A Comment on Certain Points where Cybernetics Impinges on Religion*. MIT Press, Cambridge, MA.
- Wiewiora, E. (2003). Potential-based shaping and Q-value initialization are equivalent. *Journal of Artificial Intelligence Research*, 19:205–208.
- Williams, R. J. (1986). Reinforcement learning in connectionist networks: A mathematical analysis. Technical Report ICS 8605. Institute for Cognitive Science, University of California at San Diego, La Jolla.
- Williams, R. J. (1987). Reinforcement-learning connectionist systems. Technical Report NU-CCS-87-3. College of Computer Science, Northeastern University, Boston.
- Williams, R. J. (1988). On the use of backpropagation in associative reinforcement learning. In *Proceedings of the IEEE International Conference on Neural Networks*, pp. I263–I270. IEEE San Diego section and IEEE TAB Neural Network Committee.
- Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256.
- Williams, R. J., Baird, L. C. (1990). A mathematical analysis of actor–critic architectures for learning optimal controls through incremental dynamic programming. In *Proceedings of the Sixth Yale Workshop on Adaptive and Learning Systems*, pp. 96–101. Center for Systems Science, Dunham Laboratory, Yale University, New Haven.
- Wilson, R. C., Takahashi, Y. K., Schoenbaum, G., Niv, Y. (2014). Orbitofrontal cortex as a cognitive map of task space. *Neuron*, 81(2):267–279.
- Wilson, S. W. (1994). ZCS, A zeroth order classifier system. *Evolutionary Computation*, 2(1):1–18.
- Wise, R. A. (2004). Dopamine, learning, and motivation. *Nature Reviews Neuroscience*, 5(6):1–12.
- Witten, I. H. (1976a). Learning to Control. University of Essex PhD thesis.
- Witten, I. H. (1976b). The apparent conflict between estimation and control—A survey of the two-armed problem. *Journal of the Franklin Institute*, 301(1-2):161–189.

- Witten, I. H. (1977). An adaptive optimal controller for discrete-time Markov environments. *Information and Control*, 34(4):286–295.
- Witten, I. H., Corbin, M. J. (1973). Human operators and automatic adaptive controllers: A comparative study on a particular control task. *International Journal of Man-Machine Studies*, 5(1):75–104.
- Woodbury, T., Dunn, C., and Valasek, J. (2014). Autonomous soaring using reinforcement learning for trajectory generation. In *52nd Aerospace Sciences Meeting*, p. 0990.
- Woodworth, R. S. (1938). *Experimental Psychology*. New York: Henry Holt and Company.
- Xie, X., Seung, H. S. (2004). Learning in neural networks by reinforcement of irregular spiking. *Physical Review E*, 69(4):041909.
- Xu, X., Xie, T., Hu, D., Lu, X. (2005). Kernel least-squares temporal difference learning. *International Journal of Information Technology*, 11(9):54–63.
- Yagishita, S., Hayashi-Takagi, A., Ellis-Davies, G. C. R., Urakubo, H., Ishii, S., Kasai, H. (2014). A critical time window for dopamine actions on the structural plasticity of dendritic spines. *Science*, 345(6204):1616–1619.
- Yee, R. C., Saxena, S., Utgoff, P. E., Barto, A. G. (1990). Explaining temporal differences to create useful concepts for evaluating states. In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90)*, pp. 882–888. AAAI Press, Menlo Park, CA.
- Yin, H. H., Knowlton, B. J. (2006). The role of the basal ganglia in habit formation. *Nature Reviews Neuroscience*, 7(6):464–476.
- Young, P. (1984). *Recursive Estimation and Time-Series Analysis*. Springer-Verlag, Berlin.
- Yu, H. (2010). Convergence of least squares temporal difference methods under general conditions. *International Conference on Machine Learning* 27, pp. 1207–1214.
- Yu, H. (2012). Least squares temporal difference methods: An analysis under general conditions. *SIAM Journal on Control and Optimization*, 50(6):3310–3343.
- Yu, H. (2015). On convergence of emphatic temporal-difference learning. In *Proceedings of the 28th Annual Conference on Learning Theory, JMLR W&CP 40*. Also ArXiv:1506.02582.
- Yu, H. (2016). Weak convergence properties of constrained emphatic temporal-difference learning with constant and slowly diminishing stepsize. *Journal of Machine Learning Research*, 17(220):1–58.
- Yu, H. (2017). On convergence of some gradient-based temporal-differences algorithms for off-policy learning. Arxiv:1712.09652.
- Yu, H., Mahmood, A. R., Sutton, R. S. (2017). On generalized bellman equations and temporal-difference learning. ArXiv:17041.04463. A summary appeared in *Proceedings of the Canadian Conference on Artificial Intelligence*, pp. 3–14. Springer.

Index

Page numbers in *italics* are recommended to be consulted first. Page numbers in **bold** contain boxed algorithms.

- k*-armed bandits, 25–45
- absorbing state, 57
- access-control queuing example, 256
- action preferences, 322, 329, 336, 455
 - in bandit problems, 37, 42
- action-value function, *see* value function, action
- action-value methods, 321
 - for bandit problems, 27
- actor–critic, 21, 239, 321, 331–332, 338, 406
 - one-step (episodic), **332**
 - with eligibility traces (episodic), **332**
 - with eligibility traces (continuing), **333**
 - neural, 395–415
- addiction, 409–410
- advantage actor–critic methods, 338
- afterstates, 137, 140, 181, 182, 191, 424, 430
- agent–environment interface, 47–57, 467
- all-actions algorithm, 326
- AlphaGo, AlphaGo Zero, AlphaZero, 441–450
- Andreae, John, 17, 21, 69, 89
- ANN, *see* artificial neural networks
- applications and case studies, 421–457
- approximate dynamic programming, 15
- artificial intelligence, xvii, 1, 472, 475–478
- artificial neural networks, 223–228, 238–239, 395–398, 423, 430, 436–450, 472
- associative reinforcement learning, 45, 418
- associative search, 41
- asynchronous dynamic programming, 85, 88
- Atari video game play, 436–441
- auxiliary tasks, 460–461, 468, 474
- average reward setting, 249–255, 258, 464
- averagers, 264
- backgammon, 11, 21, 182, 184, 421–426
- backpropagation, 21, 225–227, 239, 407, 424, 436, 439
- backup diagram, 60, 139
 - for dynamic programming, 59, 61, 64, **172**
 - for Monte Carlo methods, 94
 - for Q-learning, 134
 - for TD(0), 121
- for Sarsa, 129
- for Expected Sarsa, 134
- for Sarsa(λ), 304
- for TD(λ), 289
- for Q(λ), 313
- for Tree Backup(λ), 314
- for truncated TD(λ), 296
- for n -step $Q(\sigma)$, 155
- for n -step Expected Sarsa, 146
- for n -step Sarsa, 146
- for n -step TD, 142
- for n -step Tree Backup, 152
- for Samuel’s Checker Player, 428
- compound, 288
- half backups, 62
- backward view of eligibility traces, 288, 293
- Baird’s counterexample, 261–264, 280, 283, 285
- bandit algorithm, simple, **32**
- bandit problems, 25–45
- basal ganglia, 386
- baseline, 37–40, 329, 330, 331, 338
- behavior policy, 103, 110, *see* off-policy learning
- Bellman equation, 14
 - for v_π , 59
 - for q_π , 78
 - for optimal value functions: v_* and q_* , 63
 - differential, 250
 - for options, 463
- Bellman error, 268, 270, 272, 273
 - learnability of, 274–277
 - vector, 267–269
- Bellman operator, 267–269, 286
- Bellman residual, 286, *see* Bellman error
- Bellman, Richard, 14, 71, 89, 241
- binary features, 215, 222, 245, 304, 305
- bioreactor example, 51
- blackjack example, 93–94, 99, 106
- blocking maze example, 166
- bootstrapping, 89, 189, 308, 331
 - n -step, 141–158, 255
 - and dynamic programming, 89
 - and function approximation, 208, 264–274

- and Monte Carlo methods, 95
 and stability, 263–265
 and TD learning, 120
 assessment of, 124–128, 248, 264, 291, 318
 in psychology, 345, 349, 354, 355
 parameter (λ or n), 291, 307, 399
- BOXES**, 18, 71, 237
- branching factor, 173–177, 422
- breakfast example, 5, 22
- bucket-brigade algorithm, 19, 21, 139
- catastrophic interference, 472
- certainty-equivalence estimate, 128
- chess, 4, 20, 54, 182, 450
- classical conditioning, 20, 343–357
 blocking, 371
 and higher-order conditioning, 345–355
 delay and trace conditioning, 344
 Rescorla-Wagner model, 346–349
 TD model, 349–357
- classifier systems, 19, 21
- cliff walking example, 132, 133
- CMAC, *see* tile coding
- coarse coding, 215–220, 237
- cognitive maps, 363–364
- collective reinforcement learning, 404–407
- complex backups, *see* compound update
- compound stimulus, 345, 346–356, 371, 382
- compound update/backup, 288, 319
- conditioned/unconditioned stimulus, conditioned response (CS/US, CR), 343
- constant- α MC, 120
- contextual bandits, 41
- continuing tasks, 54, 57, 70, 124, 249, 294
- continuous action, 73, 244, 335–336
- continuous state, 73, 223, 237
- continuous time, 11, 71
- control and prediction, 342
- control theory, 4, 70
- control variates, 150–152, 155, 281
 and eligibility traces, 309–312
- credit assignment, 11, 17, 19, 47, 294, 401
 in psychology, 346, 361
 structural, 385, 405, 407
- critic, 18, 238, 346, 417, *see also* actor-critic
- cumulant, 459
- curiosity, 474
- curse of dimensionality, 4, 14, 221, 231
- cybernetics, xvii, 477
- deadly triad, 264
- deep learning, 11, 223, 441, 472–474, 480
- deep reinforcement learning, 236
- deep residual learning, 226
- delayed reinforcement, 361–363
- delayed reward, 2, 47, 249
- dimensions of reinforcement learning methods, 189–191
- direct and indirect RL, 162, 164, 192
- discounting, 55, 199, 243, 249, 282, 324, 328, 427, 459
 in pole balancing, 56
 state dependent, 307
 deprecated, 253, 256
- distribution models, 159, 185
- dopamine, 377, 381–387, 413–419
 and addiction, 409–410
- double learning, 134–136, 140
- DP, *see* dynamic programming
- driving-home example, 122–123
- Dyna architecture, 164, 161–170
- dynamic programming, 13–15, 73–90, 174, 262
 and artificial intelligence, 89
 and function approximation, 241
 and options, 463
 and the deadly triad, 264
 computational efficiency of, 87
- eligibility traces, 287–320, 350, 362, 398–403
 accumulating, 301, 306, 310
 replacing, 301, 306
 dutch, 300–303
 contingent/non-contingent, 399–403, 411
 off-policy, 309–316
 with state-dependent λ and γ , 309–316
- Emphatic-TD methods, 234–235, 315
 off-policy, 281–282
- environment, 47–57
- episodes, episodic tasks, 11, 54–57, 91
- error reduction property, 144, 288
- evaluative feedback, 17, 25, 47
- evolution, 7, 359, 374, 471
- evolutionary methods, 7, 8–10, 19
- expected approximate value, 148, 155
- Expected Sarsa, 133, *see also* Sarsa, Expected
- expected update, 75, 172–181, 189
- experience replay, 440–441
- explore/exploit dilemma, 3, 103, 472
- exploring starts, 96, 98–100, 178

- feature construction, 210–223
final time step (T), 54
Fourier basis, 211–214
function approximation, 195–200
- gambler’s example, 84
game theory, 19
gazelle calf example, 5
general value functions (GVFs), 459–463, 474
generalized policy iteration (GPI), 86–87, 92, 97, 138, 189
genetic algorithms, 19
Gittins index, 43
gliding/soaring case study, 453–457
goal, *see* reward signal
golf example, 61, 63, 66
gradient, 201
gradient descent, *see* stochastic gradient descent
Gradient-TD methods, 278–281, 314–315
greedy or ε -greedy
as exploiting, 26–28
as shortsighted, 64
 ε -greedy policies, 100
gridworld examples, 60, 65, 76, 147
cliff walking, 132
Dyna blocking maze, 166
Dyna maze, 164
Dyna shortcut maze, 167
windy, 130, 131
- habitual and goal-directed control, 364–368
hedonistic neurons, 402–404
heuristic search, 181–183, 190
as sequences of backups, 183
in Samuel’s checkers player, 426
in TD-Gammon, 425
history of reinforcement learning, 13–21
Holland, John, 19, 21, 44, 139, 241
Hull, Clark, 16, 359, 360, 362–363
- importance sampling, 103–117, 151, 257
ratio, 104, 148, 258
weighted and ordinary, 105, 106
and eligibility traces, 309–312
and infinite variance, 106
discounting aware, 112–113
incremental implementation, 109
per-decision, 114–115
- n -step, 148–156
incremental implementation
of averages, 30–33
of weighted averages, 109
instrumental conditioning, 357–361, *see also*
Law of Effect
and motivation, 360–361
Thorndike’s puzzle boxes, 358
interest and emphasis, 234–235, 282, 316
inverse reinforcement learning, 470
- Jack’s car rental example, 81–82, 137, 210
- kernel-based function approximation, 232–233
Klopf, A. Harry, xv, xvii, 19–21, 402–404, 411
- latent learning, 192, 363, 366
Law of Effect, 15–16, 45, 342, 358–361, 417
learning automata, 18
Least Mean Square (LMS) algorithm, 279, 301
Least-Squares TD (LSTD), 228–229
linear function approx., 204–209, 266–269
linear programming, 87, 90
local and global optima, 200
- Markov decision process (MDP), 2, 14, 47–71
Markov property, 49, 115, 465–468
Markov reward process (MRP), 125
maximization bias, 134–136
maximum-likelihood estimate, 128
MC, *see* Monte Carlo methods
Mean Squared
Bellman Error, \overline{BE} , 268
Projected Bellman Error, \overline{PBE} , 269
Return Error, \overline{RE} , 275
TD Error, \overline{TDE} , 270
Value Error, \overline{VE} , 199–200
- memory-based function approx., 230–232
Michie, Donald, 17, 71, 116
Minsky, Marvin, 16, 17, 20, 89
model of the environment, 7, 159
model-based and model-free methods, 7, 159
in animal learning, 363–368
model-based reinforcement learning, 159–193
in neuroscience, 407–409
Monte Carlo methods, 91–117
first- and every-visit MC, 92
first-visit MC control, 101
first-visit MC prediction, 92

- gradient method for v_π , **202**
 Monte Carlo ES (Exploring Starts), **99**
 off-policy control, **111**, 110–112
 off-policy prediction, 103–109, **110**
 Monte Carlo Tree Search (MCTS), 185–188
 motivation, 360–361
 mountain car example, 244–248, 305, 306
 multi-armed bandits, 25–45
- n*-step methods, *141–158*
 $Q(\sigma)$, **156**
 Sarsa, **147**, **247**
 differential, **255**
 off-policy, **149**
 TD, **144**
 Tree Backup, **154**
 truncated λ -return, 295
- naughts and crosses, *see* tic-tac-toe
- neural networks, *see* artificial neural networks
- neurodynamic programming, 15
- neuroeconomics, 413, 419
- neuroscience, 4, 21, *377–419*
- nonstationarity, 30, *32–36*, 41, 44, 255
 inherent, 91, 198
- notation, *xiii*, *xix*
- observations, 464
- off-policy methods, *257–286*
 vs on-policy methods, 100, 103
 Monte Carlo, 103–115
 Q-learning, **131**
 Expected Sarsa, 133–134
 n-step, 148–156
 n-step $Q(\sigma)$, **156**
 n-step Sarsa, **149**
 n-step Tree Backup, **154**
 and eligibility traces, 309–316
 Emphatic-TD(λ), 315
 GQ(λ), 315
 GTD(λ), 314
 HTD(λ), 315
 Q(λ), 312–314
 Tree Backup(λ), 312–314
 reducing variance, 283–284
- on-policy distribution, *175*, 199, 208, 258, 262, 281, 282
 vs uniform distribution, 176
- on-policy methods, 100
 actor-critic, **332**, **333**
- approximate
 control, **244**, **247**, **251**, **255**
 prediction, **202**, **203**, **209**
 Monte Carlo, 101, 100–103, **328**, **330**
n-step, **144**, **147**
 Sarsa, **130**, 129–131
 TD(0), **120**, 119–128
 with eligibility traces, **293**, **300**, **305**, **307**
- operant conditioning, *see* instrumental learning
- optimal control, 2, *14–15*, 21
- optimistic initial values, *34–35*, 192
- optimizing memory control, 432–436
- options, 461–464
 models of, 462
- pain and pleasure, 6, 16, 413
- Partially Observable MDPs (POMDPs), *466*
- Pavlov, Ivan, 16, *343–345*, 362
- Pavlovian
 conditioning, *see* classical conditioning
 control, 343, 371, 373, 479
- personalizing web services, 450–453
- planning, 3, 5, 7, 11, 138, *159–193*
 in psychology, 363, 364, 366
 with learned models, *161–168*, 473
 with options, 461, 463
- policy, 6, 41, **58**
 hierarchical, 462
 soft and ε -soft, 100–103, 110
- policy approximation, 321–324
- policy evaluation, *74–76*, *see also* prediction
 iterative, **75**
- policy gradient methods, 321–338
 REINFORCE, **328**, **330**
 actor-critic, **332**, **333**
- policy gradient theorem, 324–326
 proof, episodic case, 325
 proof, continuing case, 334
- policy improvement, 76–80
 theorem, 78, 101
- policy iteration, 14, **80**, *80–82*
- polynomial basis, 210–211
- prediction, *74–76*, *see also* policy evaluation
 and control, 342
 Monte Carlo, 92–97
 off-policy, 103–108
 TD, 119–126
 with approximation, 197–241
- prior knowledge, 11, 34, 54, 137, 236, 324, 470

- prioritized sweeping, **170**, 168–171
 projected Bellman error, 285
 vector, 267, **269**
 proximal TD methods, 286
 pseudo termination, 282, **308**
 psychology, 4, 13, 18, 20, **341–376**
- $Q(\lambda)$, Watkins's, 312–314
 Q -function, *see* action-value function
 Q -learning, 21, **131**, *131–135*
 double, **136**
 Q -planning, **161**
 $Q(\sigma)$, **156**, 154–156
 queuing example, 251
- R-learning, 256
 racetrack exercise, 111
 radial basis functions (RBFs), 221–222
 random walk, 95
 5-state, **125**, 126, 127
 19-state, **144**, 291
 TD(λ) results on, **294**, **295**, 299
 1000-state, **203–209**, 217, 218
 Fourier and polynomial bases, 214
 real-time dynamic programming, 177–180
 recycling robot example, 52
 REINFORCE, **328**, 326–331
 with baseline, **330**
 reinforcement learning, 1–21
 reinforcement signal, **380**
 representation learning, 473
 residual-gradient algorithm, **272–274**, 277
 naive, 270, 271
 return, 54–57
 n-step, **143**
 for $Q(\sigma)$, **155**
 for action values, **146**
 for Expected Sarsa, **148**
 for Tree Backup, **153**
 with control variates, **150**, **151**
 with function approximation, **209**
 differential, **250**, **255**, 334
 flat partial, 113
 with state-dependent termination, **308**
 λ -return, **288–291**
 truncated, 296
 reward prediction error hypothesis, **381–383**, **387–395**
 reward signal, 1, 6, **48**, **53**, 361, 380, 383, 397
- and reinforcement, 373–375, 380–381
 design of, **469–472**, 476
 intrinsic, 474
 sparse, 469–470
 rod maneuvering example, 171
 rollout algorithms, 183–185
 root mean-squared (RMS) error, 125
- safety, **434**, **475–478**
 sample and expected updates, **121**, 170–174
 sample or simulation model, 115
 sample-average method, 27
 Samuel's checkers player, 20, 241, **426–429**
 Sarsa, **130**, *129–131*, **244**
 vs Q-learning, 132
 differential, one-step, **251**
 Expected, *133–134*, 140
 n-step, 148
 n-step off-policy, 150
 double, 136
 n-step, **147**, *145–148*, **247**
 differential, **255**
 off-policy, **149**
 Sarsa(λ), **305**, 303–307
 true online, **307**
 Schultz, Wolfram, 387–395, **410**
 search control, 163
 secondary reinforcement, 20, 346, 354, 369
 selective bootstrap adaptation, 239
 semi-gradient methods, 202, **258–259**
 SGD, *see* stochastic gradient descent
 Shannon, Claude, 16, 20, 70, 71, 426
 shaping, **360**, 470
 Skinner, B. F., **359–360**, 375, 470, 480
 soap bubble example, 95
 soft and ε -soft policies, 100–103, 110
 soft-max, **322–323**, 329, 336, 400, 445, 455
 for bandits, **37**, 45
 spike-timing-dependent plasticity (STDP), 401
 state, 7, 48, 49
 *k*th-order history approach, 468
 and observations, 464–468
 Markov property, 465–468
 belief, 466
 latent, 466
 observable operator models (OOMs), 467
 partially observable MDPs, 14, 466
 predictive state representations, 466
 state-update function, 466

- state aggregation, 203–204
 state-update function, 466
 step-size parameter, 10, 31–33, 120, 125, 126
 automatic adaptation, 238
 in DQN, 439, 440
 in psychological models, 347, 348
 selecting manually, 222–223
 with coarse coding, 216
 with Fourier features, 213
 with tile coding, 217, 223
 stochastic approx. convergence conditions, 33
 stochastic gradient descent (SGD), 200–204
 in the Bellman error, 269–277
 strong and weak methods, 4
 supervised learning, xvii, 2, 16–19, 198
 sweeps, 75, 160, *see also* prioritized sweeping
 synaptic plasticity, 379
 Hebbian, 400
 two-factor and three factor, 400
 system identification, 364
- tabular solution methods, 23
 target
 policy, 103, 110
 of update, 31, 143, 198
 TD, *see* temporal-difference learning
 TD error, 121
 n-step, 255
 differential, 250
 with function approximation, 270
 TD(λ), **293**, 292–295
 truncated, 295–297
 true online, **300**, 299–301
 TD-Gammon, 21, 421–426
 temporal abstraction, 461–464
 temporal-difference learning, 10, 119–140
 history of, 20–21
 advantages of, 124–126
 optimality of, 126–128
 TD(0), **120**, **203**
 TD(1), 294
 TD(λ), **293**, 292–295
 true online, **300**, 299–301
 λ -return methods
 offline, 290
 online, 297–299
 n-step, **144**, 141–158, **209**
 termination function, 307, 459
 Thompson sampling, 43, 45
- Thorndike, Edward, *see* Law of Effect
 tic-tac-toe, 8–13, 17, 137
 tile coding, 217–221, 223, 238, 246, 434, 435
 Tolman, Edward, 364, 408
 trace-decay parameter (λ), 287, 289, 290, 292
 state dependent, 307
 trajectory sampling, 174–177
 transition probabilities, 49
 Tree Backup
 n-step, 152–153, **154**
 Tree-Backup(λ), 312–314
 trial-and-error, 2, 7, 15–21, 403, 404, *see also*
 instrumental conditioning
 true online TD(λ), **300**, 299–301
 Tsitsiklis and Van Roy's Counterexample, 263
- undiscounted continuing tasks, *see* average reward setting
 unsupervised learning, 2, 226
- value, 6, 26, 47
 value function, 6, 58–67
 for a given policy: v_π and q_π , 58
 for an optimal policy: v_* and q_* , 62
 action, 58, 63, 64, 71, 129, 131
 approximate action values: $\hat{q}(s, a, \mathbf{w})$, 243
 approximate state values: $\hat{v}(s, \mathbf{w})$, 197
 differential, 243
 vs evolutionary methods, 10
 value iteration, **83**, 82–85
 value-function approximation, 198
- Watkins, Chris, 14, 21, 89, 320
 Watson (*Jeopardy!* player), 429–432
 Werbos, Paul, 14, 21, 69, 89, 139, 238
 Witten, Ian, 21, 69

Adaptive Computation and Machine Learning

Francis Bach, Editor

Bioinformatics: The Machine Learning Approach, Pierre Baldi and Søren Brunak

Reinforcement Learning: An Introduction, Richard S. Sutton and Andrew G. Barto

Graphical Models for Machine Learning and Digital Communication, Brendan J. Frey

Learning in Graphical Models, Michael I. Jordan

Causation, Prediction, and Search, second edition, Peter Spirtes, Clark Glymour, and Richard Scheines

Principles of Data Mining, David Hand, Heikki Mannila, and Padhraic Smyth

Bioinformatics: The Machine Learning Approach, second edition, Pierre Baldi and Søren Brunak

Learning Kernel Classifiers: Theory and Algorithms, Ralf Herbrich

Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond, Bernhard Schölkopf and Alexander J. Smola

Introduction to Machine Learning, Ethem Alpaydin

Gaussian Processes for Machine Learning, Carl Edward Rasmussen and Christopher K.I. Williams

Semi-Supervised Learning, Olivier Chapelle, Bernhard Schölkopf, and Alexander Zien, Eds.

The Minimum Description Length Principle, Peter D. Grünwald

Introduction to Statistical Relational Learning, Lise Getoor and Ben Taskar, Eds.

Probabilistic Graphical Models: Principles and Techniques, Daphne Koller and Nir Friedman

Introduction to Machine Learning, second edition, Ethem Alpaydin

Machine Learning in Non-Stationary Environments: Introduction to Covariate Shift Adaptation, Masashi Sugiyama and Motoaki Kawanabe

Boosting: Foundations and Algorithms, Robert E. Schapire and Yoav Freund

Machine Learning: A Probabilistic Perspective, Kevin P. Murphy

Foundations of Machine Learning, Mehryar Mohri, Afshin Rostami, and Ameet Talwalkar

Introduction to Machine Learning, third edition, Ethem Alpaydin

Deep Learning, Ian Goodfellow, Yoshua Bengio, and Aaron Courville

Elements of Causal Inference, Jonas Peters, Dominik Janzing, and Bernhard Schölkopf

Machine Learning for Data Streams, with Practical Examples in MOA, Albert Bifet, Ricard Gavaldà, Geoffrey Holmes, Bernhard Pfahringer