Springer Theses
Recognizing Outstanding Ph.D. Research

Christopher Gatti

Design of Experiments for Reinforcement Learning

Springer

Springer Theses

Recognizing Outstanding Ph.D. Research

Springer Theses-the “best of the best”

Internationally top-ranked research institutes select their best thesis annually for publication in this series. Nominated and endorsed by two recognized specialists, each thesis is chosen for its scientific excellence and impact on research. For greater accessibility to non-specialists, the published versions include an extended introduction, as well as a foreword by the student’s supervisor explaining the special relevance of the work for the field. As a whole, the series provides a valuable resource both for newcomers to the relevant field, and for other scientists seeking detailed background information on special questions. Finally, it provides an accredited documentation of the valuable contributions made by today’s younger generation of scientists.

The content of the series is available to millions of readers worldwide and, in addition to profiting from this broad dissemination, the author of each thesis is rewarded with a cash prize equivalent to € 500.

More information about this series at http://www.springer.com/series/8790

Christopher Gatti

Design of Experiments for Reinforcement Learning

Springer

Christopher Gatti
Industrial and Systems Engineering
Rensselaer Polytechnic Institute
Troy
New York
USA

ISSN 2190-5053 ISSN 2190-5061 (electronic)
ISBN 978-3-319-12196-3 ISBN 978-3-319-12197-0 (eBook)
DOI 10.1007/978-3-319-12197-0
Springer Cham Heidelberg New York Dordrecht London

Library of Congress Control Number: 2014954825

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Foreword

This monograph addresses reinforcement learning with neural networks. Reinforcement learning is a machine learning technique that is used for solving sequential decision making problems. In essence, reinforcement learning is based on an iterative process of trial-and-error, very similar to how humans learn how to perform a task. A typical example of such a problem is game playing where, over many games, players learn moves that contribute to winning the game. A challenging caveat here is that during the game the utility of each individual move is unknown, and it is only at the end of the game that a winner is declared and the utility of all moves can be determined.

One way to implement reinforcement learning is by using neural networks, which can be trained with the temporal difference algorithm, a variation of the popular backpropagation algorithm. This approach implicitly solves the Bellman equation, allows for very large state spaces, and requires little *a priori* knowledge about the environment, which can plague other approaches such as models based on hidden Markov Models. However, there is one serious drawback to using neural networks for reinforcement learning: learning may not always convergence and a solution may not be found. For that reason, its extension to real-world problems is often not successful, and thus there are relatively few noteworthy applications.

The purpose of this monograph is to gain a better understanding of reinforcement learning in a variety of problems. This work contributes fundamental knowledge, basic science if you will, of what affects reinforcement learning and what contributes to a successful implementation. This work takes an empirical approach to understanding of the behavior and interactions between the two main components of a neural network based implementation of reinforcement learning: the learning algorithm and the functional representation of the learned knowledge.

This monograph serves as a textbook introduction to reinforcement learning with neural networks. A key original contribution of this work is the utilization of contemporary design of experiments methods, including a novel sequential experimentation procedure that finds convergent learning algorithm parameter subregions, and stochastic kriging for response surface metamodeling. Three single-agent problems are considered in this work: the mountain car problem, the truck backer-upper problem, and the tandem truck backer-upper problem. In the appendix, a similar approach to a game playing problem is illustrated as well.

The knowledge gained from this work provides insight as to what enables and what has an effect on successful reinforcement learning implementations so that this learning method can be applied routinely to more challenging problems.

Department of Industrial
and Systems Engineering
Rensselaer Polytechnic Institute
Clifton Park, New York—August 2014

Mark J. Embrechts, Professor

Acknowledgment

First and foremost, I would like to acknowledge and thank my longtime mentor and friend Dr. Richard Hughes from the University of Michigan. After graduating in 2005 and having no work experience, he took a chance on me, for which I am extremely grateful. He taught me how to do research, he gave me time to get lost in research and fall in love with the process of inquiry and discovery, and he instilled a great sense of academic integrity in me. I would not have gone into research and be where I am today if it were not for Richard.

The gym has always been by home away from home (though it is arguably my first home) wherever I am. It's always given me a place to shut out the rest of the world and to focus on having fun. I'd like to thank World Class Gymnastics Academy, the kids, the coaches, and the parents, for welcoming me when I came to town and for making me a part of their family. The coaches have become my best friends and my family away from home and I cannot thank them enough for their sincere friendship. I'd also like to thank the kids at the gym for always putting a smile on my face and for reminding me that there is more to life than work.

I'd like to thank my advisor Dr. Embrechts for his mentorship over the past four years, for his motivation, for his challenging questions and discussions, and for his patience in allowing me the freedom to explore my research interests. I'd also like to thank my committee for their interest in my work, and for all of their thoughtful discussions, about both academic and non-academic matters. I'd like to thank those who I have had research appointments with during my time in graduate school, which provided me with financial support, including Dr. Kristin Bennett, Dr. William Wallace, Dr. Malik Magdon-Ismail, and Dr. Mark Goldberg. I cannot forget to thank our department secretary Mary Wager for all of her administrative help and for being a friend to chat with to break up the days.

I'd like to thank my parents and brother and sister for their love and support, for always being just a phone call away, for having people that you know always care about what's going on. I'd like to thank James Brooks and Sarah Nurre for our weekly breakfasts at Manory's, for giving some routine to my hectic life, for their thoughtful discussions, both academic and non-academic. Finally, I'd like to thank the countless individuals who have gotten me to this point, for all things large and small.

Book Note

Parts of this Thesis Have Been Published As

Gatti, C. J. & Embrechts, M. J. (2012). Reinforcement learning with neural networks: Tricks of the trade. In Georgieva, P., Mihayolva, L., & Jain, L. (Eds.), *Advances in Intelligent Signal Processing and Data Mining* (pp. 275–310). New York, NY: Springer-Verlag.

Gatti, C. J., Embrechts, M. J., & Linton, J. D. (2011). Parameter settings of reinforcement learning for the game of Chung Toi. In *Proceedings of the 2011 IEEE International Conference on Systems, Man, and Cybernetics (SMC 2011), Anchorage, AK, 9–12 October* (pp. 3530–3535). doi: 10.1109/ICSMC.2011.6084216

Gatti, C. J., Embrechts, M. J., & Linton, J. D. (2013). An empirical analysis of reinforcement learning using design of experiments. In *Proceedings of the 21st European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN), Bruges, Belgium, 24–26 April* (pp. 221–226). Bruges, Belgium: ESANN.

Contents

Chapter 1
Introduction

Reinforcement learning is a machine learning method that enables the learning of optimal behavior in challenging or uncertain environments. Optimal behavior in this case can be defined as the set of sequential decisions that result in the achievement of a goal or the best possible outcome. This learning process can be regarded as a process of trial-and-error, which is coupled with feedback provided from the environment that indicates the utility of the outcome. This learning method ultimately attempts to learn a mapping between actions and outcomes.

To make this learning paradigm more concrete, consider board games, which are classic examples of reinforcement learning. Players take turns assessing the state, or the configuration, of the game, and then select the action that they believe will most likely result in an outcome that is in their best interest. Game play therefore consists of making a sequence of decisions in order to achieve a goal, which is most often to win the game. During the game however, the true utility and benefit of each action is not known, and only at the end of the game is there certain evidence of the utility of all of the actions played during the game. Board games have been a prominent domain for the development of computational implementations of reinforcement learning, including applications to Tic-Tac-Toe (Wiering 1995), checkers/draughts (Wiering et al. 1995), Chess (Veness et al. 2009), Othello (van Eck and van Wezel 2008), 9×9 Go (Silver et al. 2012), and the most notable application of reinforcement learning to the game of backgammon (Tesauro 1992, 1995). Board games are widely used applications of reinforcement learning, especially for algorithm development and assessment, because they have well-defined rules and actions and their dynamics are relatively well understood.

There are relatively few significant real-world applications of reinforcement learning, though some of these applications include robot control (Smart and Kaelbling 2002), autonomous helicopters flight (Ng et al. 2004), and optimizing industrial manufacturing processes (Mahadevan and Theocharous 1998). In comparison to games, real-world environments have complex underlying processes that may not be fully understood, thus complicating the process of learning how to interact within such environments. Furthermore, while the current methods for implementing reinforcement learning are rather elegant and simple, their performance across many types of environment characteristics are not well-understood. Further complicating

C. Gatti, *Design of Experiments for Reinforcement Learning,*
Springer Theses, DOI 10.1007/978-3-319-12197-0_1

this learning method is that reinforcement learning relies on a form of knowledge representation; that is, a functional or numerical representation of the learned knowledge about how to behave in an environment. Consequently, we do not currently fully understand the effects of parameters related to the algorithms and different types of representations.

One of the fundamental limitations of applying reinforcement learning to more challenging and significant applications is our lack of knowledge regarding the behavior and performance of the current methods, including the learning algorithms and the functional representation of the knowledge. While the field of reinforcement learning has been active for nearly three decades, there has been relatively little work that attempts to understand how these learning methods behave in different scenarios, though a number of recent works have begun to explore and understand these algorithms at a fundamental level (Kalyanakrishnan and Stone 2009; Whiteson et al. 2009; Kalyanakrishnan and Stone 2011; Whiteson et al. 2011). However, the studies performed in these works consist of rather basic parameter studies and do not use designed experiments, which limits the information that can be learned from these studies.

This dissertation explores reinforcement learning using a design of experiments approach in order to understand how reinforcement learning algorithms perform under a variety of conditions. We are primarily interested in how learning algorithm and representation parameters affect learning algorithm performance under different problem domains. Specifically, the learning algorithm we focus on is the temporal difference algorithm TD(λ), and the form of knowledge representation is a neural network. Although the use of design of experiments (DoE) to study machine learning (ML) is relatively rare, this is a well-established field that allows for the discovery of objective information using formal methods and statistical analysis. The use of design of experiments for studying reinforcement learning is a novel approach that can yield significantly more information than basic parameter studies that have been used in the past.

We should acknowledge that the use of the term *design of experiments* often conjures the use of classical experimental designs with rather rigid restrictions and/or assumptions. However, in this work, we take a looser definition of this concept, and yet we still consider the approach taken herein to fall under design of experiments because it is still based on a structured approach, with an experimental design and modeling approach, to analyze a complex system of interest. The methods used in this work, however, are a slightly more contemporary and adaptable to the challenges of modern problems.

In this work, we use two types of experimentation, each of which serves a specific purpose. The use of the temporal difference algorithm with neural networks may not always converge. Consequently, finding combinations of parameters of the learning algorithm and representation that allow for learning convergence is a requisite first task to a successful reinforcement learning implementation. We develop a novel sequential experimentation procedure that is able to find small subregions of the parameter space in which reinforcement learning converges with a high probability. After finding convergent parameter subregions, we use a second

experiment to explore these regions by creating metamodels, or surrogate models, of the performance of reinforcement learning with respect to the learning algorithm and representation parameters. These metamodels allow for further exploration of the response surface, such as identifying the most influential parameters of the learning algorithm and representation, thus providing us with a better understanding of what affects this learning method. We use these experimentation and modeling techniques in a series of problem domains: the mountain car problem, the single trailer truck backer-upper problem, and the tandem trailer truck backer-upper problem.

While understanding the behavior of reinforcement learning is the major aim of this work, we stress that this work is as much about design of experiments as it is about reinforcement learning due to the unique characteristics of the data generating process. In fact, we consider this work to be *basic science*, if you will, for reinforcement learning. That is, we attempt to understand the fundamental reinforcement learning algorithm and the effects of a number of key parameters that are essential to learning convergence. Previous work by Sutton and Barto (1998), Gatti et al. (2011a), and Gatti et al. (2013) has scratched the surface of this problem using one-factor-at-a-time (OFAT) studies in simple domains, or using slightly more complex problems and experimental designs. Additionally, even recent works have been interested in understanding reinforcement learning algorithms that were developed years ago at more than a superficial level (Kalyanakrishnan and Stone 2011;Dann et al. 2014). However, these works are limited in the number of parameters they explore and in the domains that are studied. The current work uses experimental methodologies that are extensible to many more parameters and to any type of learning algorithm.

Considering the novelty of this work, that is, exploring reinforcement learning using a structured approach, it is therefore essential to keep in mind that the work presented and investigated herein is merely a start, or a beginning, to potentially many different avenues for further investigation. From a reinforcement learning perspective, of primary interest would be how additional domain characteristics affect learning. Additionally, selecting parameter combinations that allow for optimal reinforcement learning is another potential direction. From a design of experiments perspective, we would be interested in exploring our novel sequential experimentation procedure in detail and how it behaves in different problems, or using this experimentation procedure in other problem domains outside of machine learning that are also plagued with potentially non-convergent computer simulations.

This dissertation is structured as follows. Chapter 2 presents and reviews literature related to the fundamental concepts on which the proposed work is based. We provide an overview of the field of reinforcement learning including a discussion of the three major components of reinforcement learning: the domain or environment, the learning algorithm, and the form of knowledge representation. Chapter 3 then introduces and discusses relevant topics from the field of design of experiments. We detail our novel sequential experimentation approach for finding convergent parameter subregions in Chap. 4, and we also provides a 2-dimensional example problem to make the sequential algorithm more concrete. This chapter also describes our approach for creating and analyzing metamodels of response surfaces.

In Chaps. 5–7, we explore the mountain car problem, the single trailer truck backer-upper problem, and the tandem trailer truck backer-upper problem, respectively. We conclude this work with a discussion of our findings, our innovations, and possible directions for future work in Chap. 8. Appendices A and B present foundational work in exploring the effects of parameters in reinforcement learning applied a two player board game and a benchmark control problem, from which this work grew.

References

Dann, C., Neumann, G., & Peters, J. (2014). Policy evaluation with temporal differences: A survey and comparison. *Journal of Machine Learning Research*, 15(1), 809–883.

Gatti, C. J., Embrechts, M. J., & Linton, J. D. (2011a). Parameter settings of reinforcement learning for the game of Chung Toi. In *Proceedings of the 2011 IEEE International Conference on Systems, Man, and Cybernetics (SMC 2011), Anchorage, AK, 9–12 October* (pp. 3530–3535). doi: 10.1109/ICSMC.2011.6084216

Gatti, C. J., Embrechts, M. J., & Linton, J. D. (2013). An empirical analysis of reinforcement learning using design of experiments. In *Proceedings of the 21st European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN), Bruges, Belgium, 24–26 April* (pp. 221–226). Bruges, Belgium: ESANN.

Kalyanakrishnan, S. & Stone, P. (2009). An empirical analysis of value function-based and policy search reinforcement learning. In *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS '09), Budapest, Hungary, 10–15 May* (Vol. 2, pp. 749–756). Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Kalyanakrishnan, S. & Stone, P. (2011). Characterizing reinforcement learning methods through parameterized learning problems. *Machine Learning*, 84(1–2), 205–247.

Mahadevan, S. & Theocharous, G. (1998). Optimizing production manufacturing using reinforcement learning. In Cook, D. J. (Ed.) *Proceedings of the 11th International Florida Artificial Intelligence Research Society Conference, Sanibel Island, Florida, 18–20 May* (pp. 372–377). AAAI Press.

Ng, A. Y., Coates, A., Diel, M., Ganapathi, V., Schulte, J., Tse, B., Berger, E. & Liang, E. (2004). Autonomous inverted helicopter flight via reinforcement learning. In *International Symposium on Experimental Robotics (ISER-2004), Singapore, 18–21 June* (pp. 363–372). Cambridge, MA: MIT Press.

Silver, D., Sutton, R. S., & Müller, M. (2012). Temporal-difference search in computer Go. *Machine Learning*, 87(2), 183–219.

Smart, W. D. & Kaelbling, L. P. (2002). Effective reinforcement learning for mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Washington, D.C., 11–15 May* (Vol. 4, pp. 3404–3410). doi: 10.1109/ROBOT.2002.1014237

Sutton, R. S. & Barto, A. G. (1998). *Reinforcement Learning*. Cambridge, MA: MIT Press.

Tesauro, G. (1992). Practical issues in temporal difference learning. *Machine Learning*, 8(3–4), 257–277.

Tesauro, G. (1995). Temporal difference learning and TD-Gammon. *Communications of the ACM*, 38(3), 58–68.

van Eck, N. J. & van Wezel, M. (2008). Application of reinforcement learning to the game of othello. *Computers & Operations Research*, 35(6), 1999–2017.

Veness, J., Silver, D., Uther, W., & Blair, A. (2009). Bootstrapping from game tree search. In Bengio, Y., Schuurmans, D., Lafferty, J. D., Williams, C. K. I., & Culotta, A. (Eds.), *Advances in Neural Information Processing Systems 22* (pp. 1937–1945). Red Hook, NY: Curran Associates, Inc.

Whiteson, S., Tanner, B., Taylor, M. E., & Stone, P. (2009). Generalized domains for empirical evaluations in reinforcement learning. In *Proceedings of the 26th International Conference on Machine Learning: Workshop on Evaluation Methods for Machine Learning, Montreal, Canada, 14–18 June*. Retrieved from http://www.site.uottawa.ca/ICML09WS/papers/w8.pdf

Whiteson, S., Tanner, B., Taylor, M. E., & Stone, P. (2011). Protecting against evaluation overfitting in empirical reinforcement learning. In *Proceedings of the IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL), Paris, France, 11-15 April* (pp. 120–127). doi: 10.1109/ADPRL.2011.5967363

Wiering, M. A. (1995). *TD learning of game evaluation functions with hierarchical neural architectures*. Unpublished masters thesis, Department of Computer Science, University of Amsterdam, Amsterdam, Netherlands.

Wiering, M. A., Patist, J. P., & Mannen, H. (2007). *Learning to play board games using temporal difference methods* (Technical Report UU–CS–2005–048, Institute of Information and Computing Sciences, Utrecht University). Retrieved from http://www.ai.rug.nl/~mwiering/GROUP/ARTICLES/learning_games_TR.pdf.

Chapter 2
Reinforcement Learning

This chapter provides an overview of the field of reinforcement learning and concepts that are relevant to the proposed work. The field of reinforcement learning is not very well-known and although the learning paradigm is easily understandable, some of the more detailed concepts can be difficult to grasp. Accordingly, reinforcement learning is presented beginning with a review of the the fundamental concepts and methods. This introduction to reinforcement learning is followed by a review of the three major components of the reinforcement learning method: the environment, the learning algorithm, and the representation of the learned knowledge. Some of the terminology used herein may be slightly different from other fields, though this is done to be consistent with the reinforcement learning literature.

Note that in this work reinforcement learning is considered from the artificial intelligence or computer science perspective on solving sequential decision making problems. Sequential decision making problems, however, are also the focus of other fields with difference perspectives, including control theory and operations research (Kappen 2007; Powell 2008). Each of these fields uses slightly different methods that have been developed for, or have been successful in, solving types of problems with unique characteristics that are specific to each field, though there may be considerable overlap in the types of problems solved by each community. The operations research community uses approaches such as simulation-optimization, forecasting approaches for rolling-horizon problems, and dynamic programming methods (Powell 2008), whereas the control theory community uses integral control and related methods based on plant models (Kappen 2007). The field of reinforcement learning (from the artificial intelligence perspective) is not only related to other computational and mathematical approaches for solving similar problems, but it is also a well-accepted and fundamental physiological model of learning in the neuroscience community (Rescorla and Wagner 1972; Dayan and Niv 2008) with conceptual intersections between the two fields (Maia 2009; Niv 2009).

Portions of this chapter previously appeared as: Gatti & Embrechts (2012).

C. Gatti, *Design of Experiments for Reinforcement Learning,*
Springer Theses, DOI 10.1007/978-3-319-12197-0_2

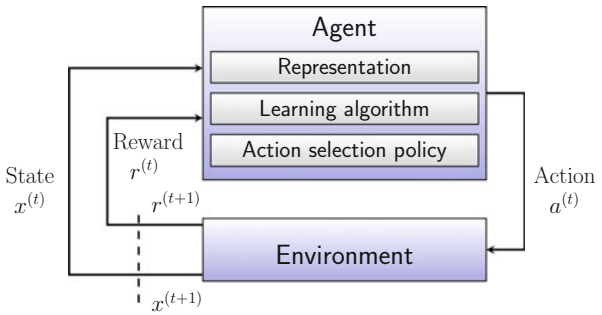


Fig. 2.1 The reinforcement learning paradigm consists of an agent interacting with an environment. The agent observes the state of the environment $x^{(t)}$ at time t, selects an action $a^{(t)}$ based on its action selection policy, and transitions to state $x^{(t+1)}$ at time $t+1$, at which point the environment issues a reward $r^{(t+1)}$ to the agent. Over time, the agent learns to pursue actions that lead to the greatest cumulative reward.

Reinforcement learning is based on an *agent*, an entity that is capable of learning and making decisions, that repeatedly interacts with an *environment* (or *domain*) (Fig. 2.1). Interactions between the agent and the environment proceed by the agent observing the state of the environment, selecting an action which it believes is likely to be beneficial, and then receiving *feedback*, or a *reward*, from the environment that provides an indication of the utility of the action (i.e., whether the action was good or bad). More specifically, in the most basic sense, the agent selects actions based on its perception of the value of the subsequent state. The feedback provided to the agent is used to improve its estimation of the value of being in each state. In the general reinforcement learning paradigm, rewards may be provided after each and every action. After repeated interaction with the environment, the agent's estimation of the true state values slowly improves, which enables the selection of more optimal actions in future interactions.

This method was largely developed as a machine learning technique by Sutton and Barto (1998). Many formulations of reinforcement learning have been developed in order to either accommodate various types of environments or to utilize slightly different learning mechanisms. One of the fundamental reinforcement learning methods is the temporal difference algorithm. This algorithm uses information from future states, and effectively propagates this information back through time in order to improve the agent's valuation of each state that was visited during an episode of agent-environment interactions. Improvements to the estimation of the state values therefore have a direct impact on the action selection processes as well. This algorithm is particularly useful in scenarios that do not necessarily follow the schema shown in Fig. 2.1, in which rewards are provided following every action. In some scenarios, with board games being a prime example, feedback is only provided at the end of the game in the form of a win, loss, or draw. The temporal difference algorithm can be used to determine the utility of actions played early in the game based on the outcome of the game, which results in a refined action selection procedure.

Reinforcement learning is more formally described as a method to determine optimal action selection policies in sequential decision making processes. The general framework is based on sequential interactions between an agent and its environment, where the environment is characterized by a set of states X, and the agent can pursue actions from the set of admissible actions A. The agent interacts with the environment and transitions from state $x^{(t)} = x$ to state $x^{(t+1)} = x'$ by selecting an action $a^{(t)} = a$. Transitions between states are often based on some defined probability $\mathcal{P}^{a}_{xx'}$. In this particular section, the time index of variables is denoted using a parenthetic-superscript or using prime notation (i.e., apostrophe), whichever is clearest or notationally cleanest, and subscripts are reserved for denoting other entities. The time index denotes the particular time step at which states are visited, actions are pursued, or rewards are received, and thus the sequences of states and actions can then be thought of as a progression through time.

The sequential decision making process therefore consists of a sequence of states $x = \{x^{(0)}, x^{(1)}, \ldots, x^{(T)}\}$ and a sequence of actions $a = \{a^{(0)}, a^{(1)}, \ldots, a^{(T)}\}$ for time steps $t = 0, 1, \ldots, T$, where state $x^{(0)}$ is the initial state and where $x^{(T)}$ is the terminal state. Feedback may be provided to the agent at each time step t in the form of a reward $r^{(t)}$ based on the particular action pursued. This feedback may be either rewarding (positive) for pursuing actions that lead to beneficial outcomes, or they may be aversive (negative) for pursuing actions that lead to worse outcomes. The terms *feedback* and *reward* will be used interchangeably, and it is important to note that a reward may be positive or negative, despite its conventional positive connotation. Processes that provide feedback at every time step t have an associated reward sequence $r = \{r^{(1)}, r^{(2)}, \ldots, r^{(T)}\}$. A complete process of agent-environment interaction from $t = 0, 1, \ldots, T$ is referred to as an *episode* consisting of the set of states visited, actions pursued, and rewards received. The total reward received for a single episode is the sum of the rewards received during the entire decision making process, $\mathcal{R} = \sum_{t=1}^{T} r^{(t)}$.

For the general sequential decision making process define above, the transition probabilities between states $x^{(t)} = x$ and $x^{(t+1)} = x'$ may be dependent on the complete sequences of previous states, actions, and rewards:

$$Pr\left\{x^{(t+1)}{=}x', r^{(t+1)}{=}r \mid x^{(t)}, a^{(t)}, r^{(t)}, x^{(t-1)}, a^{(t-1)}, r^{(t-1)}, \ldots, x^{(1)}, a^{(1)}, r^{(1)}, x^{(0)}, a^{(0)}\right\} \tag{2.1}$$

Processes in which the state transition probabilities depend only the most recent state information $(x^{(t)}, a^{(t)})$ satisfy the Markovian assumption. Under this assumption, the state transition probabilities can be expressed as:

$$Pr\left\{x^{(t+1)} = x', r^{(t+1)} = r \mid x^{(t)}, a^{(t)}\right\}$$

which is equivalent to the expression in Eq. 2.1 because all information from $t = 0, 1, \ldots, t-1$ has no effect on the state transition at time t.

If a process can be assumed to be Markovian and can be posed as a problem following the general formulation defined above, this process is amenable to both

modeling and analysis. The Markov decision process (MDP) is a specific process which entails sequential decisions and consists of a set of states X, a set of actions A, and a reward function $R(x)$. Each state $x \in X$ has an associated true state value $V^*(x)$. The decision process consists of repeated agent-environment interactions, during which the agent learns to associate states visited during the process with outcomes of entire process, thus attempting to learn the true value of each state. However, the true state values are unobservable to the agent, and thus the agents' estimates of the state values $V(x)$ are merely approximations of the true state values.

There are many cases where the Markovian assumption may not be valid. Such cases include those where the agent either cannot perfectly observe the state information, in which case the problem is referred to as a partially-observable Markov decision processes (POMDPs) (Singh et al. 1994), or if there is a long temporal dependence between states and the feedback provided. Approaches to solving these types of problem often include retaining some form of the state history (Wierstra et al. 2007), such as by using recurrent neural networks or a more complex variant that relies on long short-term memory (LSTM) (Hochreiter and Schmidhuber 1997). These approaches, however, are not considered in this work.

The transition between states $x^{(t)} = x$ and $x^{(t+1)} = x'$ when pursuing action a may be represented by a transition probability $\mathcal{P}^a_{xx'} = Pr\{x^{(t+1)} = x \mid x^{(t)} = x,\, a^{(t)} = a\}$. For problems in which the transition probabilities between all states are known, the transition probabilities can be used to formulate an explicit model of the agent-environment interaction process. A policy π is defined to be the particular sequence of actions $a = \{a^{(0)},\, a^{(1)},\, \ldots,\, a^{(T)}\}$ that is selected throughout the decision making process. Similarly, the expected reward for transitioning from state x to state x' when pursuing action a may be represented by $\mathcal{R}^a_{xx'} = \mathbb{E}\{r^{(t+1)} \mid x^{(t)} = x,\, a^{(t)} = a,\, x^{(t+1)} = x'\}$. An optimal policy π^* is considered to be the set of actions pursued during the process maximizes the total cumulative reward $\mathcal{R}$.

The optimal policy can be determined if the true state values $V^*(x)$ are known for every state. The true state values can be expressed using the Bellman optimality equation (Bertsekas 1987), which states that the value of being in state x and pursuing action a is a function of both the expected return of the current state and the optimal policy for all subsequent states:

$$V^*(x) = \max_{a \in A} \mathbb{E}\left\{r^{(t+1)} + \gamma\ V^*\left(x^{(t+1)}\right) \mid x^{(t)} = x,\, a^{(t)} = a\right\} \tag{2.2}$$

$$= \max_{a \in A} \sum_{x^+} \mathcal{P}^a_{xx'} \left[\mathcal{R}^a_{xx'} + \gamma\ V^*(x')\right] \tag{2.3}$$

The Bellman equation provides a conceptual solution to the sequential decision making problem in that the value of being in a state is a function of all future state values. The solution to reinforcement learning problems is often regarded as a policy π^*, or an action selection procedure, that leads to an optimal outcome. When implementing reinforcement learning however, the Bellman equation does not have to be explicitly solved. Rather, the identification of optimal policies in reinforcement learning problems is based on the notion that future information is relevant to the valuation of current states, and that accurate estimations of state values can be used to determine the optimal policy.

As previously mentioned, if the transition probabilities $\mathcal{P}_{xx'}$ between states are explicitly known, these probabilities can be used to formulate a model of the system. However, if these transition probabilities are unknown, transitions between states must proceed by what could be referred to as empirical sampling in a simulation-like manner. In this case, the structure of the environment can be used with the allowable actions to essentially generate the transitions between states and constrain the behavior of the agent, where the structure of the environment refers to anything that governs or has an effect on the dynamics of the environment. When state transition probabilities are unknown, the agent-environment system is considered to be *model-free*. This model-free type of problem is what classical reinforcement learning studies, and this is also the focus of this work.

Whereas supervised learning problems have a static mapping between the input space X and their labels Y, and sequence learning problems have a sequence-dependent mapping between the input space X and a set of labels Y, the only true mapping in reinforcement learning is between some states $x^* \subset X$, where x^* denotes a goal or terminal states, and their associated rewards R, which may be provided very sporadically. The classical reinforcement learning problem can be defined by an agent that interacts with an environment, and that attempts to learn the utility of states by the receipt of (possibly infrequent) rewards, in order to select a sequence of actions that maximizes the reward received. While there are many extensions, modifications, and special cases to this learning paradigm, this definition is sufficient for this work, and any exceptions will be noted.

2.1 Applications of Reinforcement Learning

Reinforcement learning has been successfully applied to many different problem areas. As the methods and extensibility of this learning approach are still being understood, a large majority of the literature uses benchmark problems. Beyond these benchmark problems, the application of reinforcement learning to games represents the next largest category, followed finally by a comparably small amount of works that apply reinforcement learning to real-world problems. In all of these applications, the definition of a *successful* application or implementation of reinforcement learning is subject to interpretation, and it is important to keep this in mind. Initial or incremental steps made in a particular application may be viewed as progress by some; others, however, may view such work as a failure if an optimal solution is not found or perfect performance is not achieved.

2.1.1 Benchmark Problems

Benchmark problems and domains are often used in reinforcement learning to evaluate novel approaches under very well-understood domain conditions. These domains can have either discrete or continuous state spaces that are often low dimensional ($\sim$ 2), allowing for visualization of functions based on their state space. These

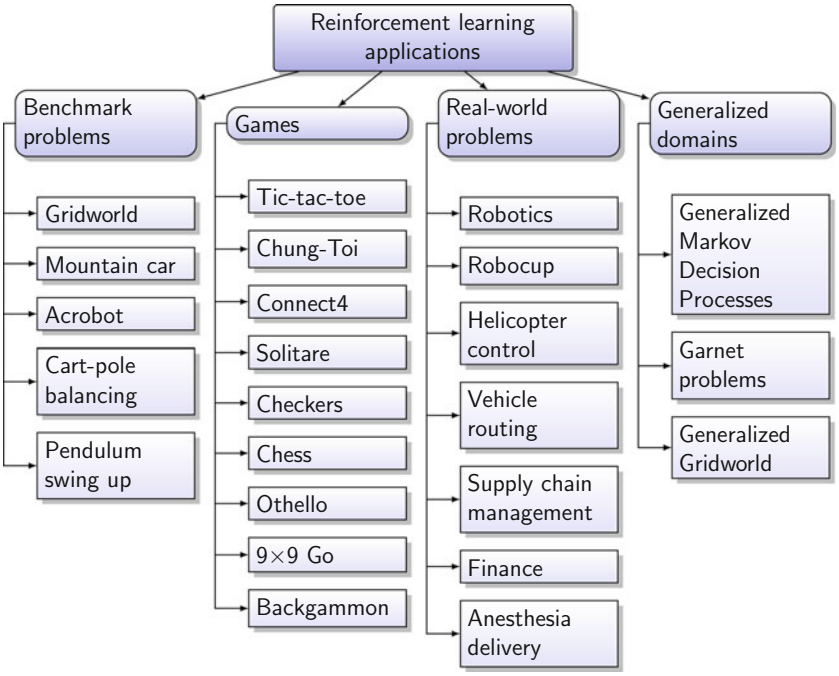


Fig. 2.2 Reinforcement learning has been applied to benchmark problems, games, real-world problems, and generalized domains. This figure shows, for each category, examples of the types of problems to which reinforcement learning has been applied.

domains are almost exclusively based on a single agent (non-adversarial domains). Though these problems are widely used and accepted, the extensibility of the learning method novelties applied to these problems is often not fully explored.

The most widely used benchmark problem in reinforcement learning is Gridworld (Sutton and Barto 1998; Fig. 2.3). In its most basic form, this domain consists of a discrete planar grid with finite dimensions. An agent is placed at some set starting location, and then selects cardinal actions (up, down, left, right) to move within the grid with the goal of reaching some specific goal grid location. Modifications to this domain include the addition of diagonal moves, the addition of penalty or hole grid locations, changes to the dimensions of the grid, and the addition of a stochastic 'wind' component that acts on the agent (Sutton and Barto 1998). Further extensions include using a large gridspace with multiple rooms, which has been used for evaluating hierarchical learning strategies through the use of subgoals (Bakker and Schmidhuber 2004; Şimşek and Barto 2004). All of these modifications are purely environmental and have an effect on the dynamics of the domain, which has downstream effects on the actions of the agent, the efficacy of the learning algorithm, and finally on the knowledge acquired by the agent and its performance in the domain.

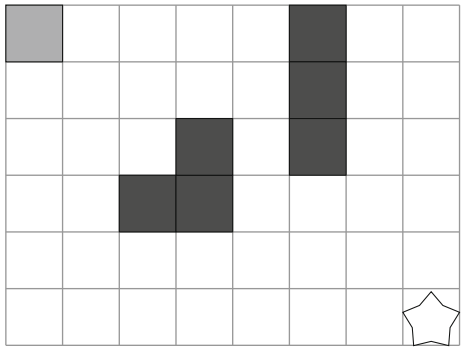


Fig. 2.3 Example Gridworld domain: The goal of this domain is to move from the starting location (*gray square*) to the goal (*star*) by taking cardinal actions and avoiding the walls (*black squares*).

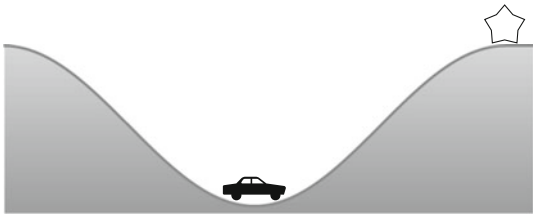


Fig. 2.4 Mountain car problem: The goal of this domain is to have the underpowered car reach the top of the mountain (*star*) by building momentum using forward or reverse actions or using no action.

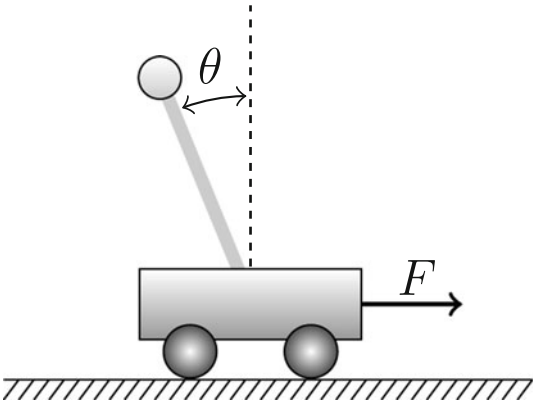


Fig. 2.5 Cart pole-balancing task: The goal of the cart-pole balancing task is to keep the pole vertically-oriented by applying left- or right-directed forces to the cart.

Additional benchmark problems include low-dimensional and continuous control problems that are based on well-defined dynamics of physical systems. Examples of these domains include the mountain car domain (Moore 1990; Singh and Sutton 1996; Riedmiller 2005; Fig. 2.4), acrobot (Sutton and Barto 1998), cart-pole balancing (Barto et al. 1983; Riedmiller 2005; Fig. 2.5), and the pendulum swing-up task (Doya 1996, 2000). The dynamics of the environment (i.e., equations of motion) are unknown to the agent, though it must learn how to behave in the environment only through selecting control actions and the receipt of rewards or penalties.

2.1.2 Games

Games represent a large proportion of the applications of reinforcement learning. In almost all of these applications, the domains are based on two-agent, adversarial, and zero-sum situations. Games in which reinforcement learning have been applied include: Tic-Tac-Toe (Wiering 1995; Ghory 2004; Patist and Wiering 2004; Konen and Beielstein 2008, 2009; Gatti and Embrechts 2012), Chung-Toi (Gatti et al. 2011a), Connect4 (Ghory 2004), Solitare (Yan et al. 2004), checkers/draughts (Schaeffer et al. 2001; Patist and Wiering 2004; Wiering et al. 2007), Chess (Thrun 1995; Baxter et al. 1998a; Mannen and Wiering 2004; Wiering et al. 2007; Veness et al. 2009), Othello (Binkley et al. 2007; van Eck and van Wezel 2008; Yoshioka et al. 1999; Skoulakis and Lagoudakis 2012), 9×9 Go (Schraudolph et al. 1994; Silver et al. 2012), and backgammon (Tesauro 1995; Wiering et al. 2007; Papahristou and Refanidis 2011). Note that these games span a large range of difficulty and that they all have unique environmental characteristics. It is also very important to note that the large majority of these applications to games have not resulted in agents that can play perfect games against any level of opponent. A truly successful application of reinforcement learning to games is often considered one that is capable of matching or beating the performance of human master players or (non-reinforcement learning trained) computer programs. Some of these successes include those to chess (Veness et al. 2009), checkers (Schaeffer et al. 2001), 9×9 Go (Silver et al. 2012), and backgammon (Tesauro 1995). In most circumstances, reinforcement learning applications to games are evaluated against computer opponents that either use a simplistic action-selection policy or are computer programs that have been developed for the same game (e.g., Schraudolph et al. 1994; Patist and Wiering 2004; Silver et al. 2012). Far fewer applications evaluate the performance of a reinforcement learning agent against human opponents, such as in Tesauro (1995) or Gatti et al. (2011b).

The most notable and most cited application of reinforcement learning is the work of Tesauro (1995) who trained a neural network to play the board game of backgammon that could challenge and win against human grandmasters in world championship play. The reasons why this applications was so powerful are not quite understood; similarly, the reasons why no other application has seen similar success is not understood as well. Some attribute the success of this application to the speed of play, representation smoothness, and stochasticity (Baxter et al. 1998b), while others question whether it is actually a true success and claim that its success is not due to reinforcement learning but rather the dynamics and co-evolution of the game (Schraudolph et al. 1994; Pollack and Blair 1996).

2.1.3 Real-World Applications

Applications of reinforcement learning to real-world problem are much less cited, though there are still numerous applications in a variety of domains. In the field of robotics, Smart and Kaelbling (2002) and Smart (2002) used Q-learning to train

mobile robots to navigate through different environments. Kohl and Stone (2004) used policy gradient reinforcement learning to train quadruped robots to walk with a fast, stable, and robust gait. Kwok and Fox (2004) used least squares reinforcement learning to train RoboCup (robot soccer) robots how to use sensors to accurately detect, gain possession, and kick a ball. Peters and Schaal (2006) used policy gradient methods for robot control with extensions to motor control strategies (Peters and Schaal 2009). Reinforcement learning for motor control has also been explored by Coulom (2002a, b).

Control is another area in which there have been successes, most notably to the control of autonomous helicopters for learning how to fly under challenging conditions (Bagnell and Schneider 2001; Ng et al. 2004). In the field of operations research, Proper and Tadepalli (2006) use reinforcement learning in a toy vehicle routing problem, and Gabel and Riedmiller (2007) use multi-agent reinforcement learning to solve a benchmark job-shop scheduling problem. A reinforcement learning algorithm named SMART has been used in a number of OR-related applications, including to optimizing industrial manufacturing processes (Mahadevan and Theocharous 1998), airline revenue management (Gosavi et al. 2002), and supply chain management (Pontrandolfo et al. 2002). Reinforcement learning has also been used for a number of problems in finance and economics. Moody and colleagues use reinforcement learning to trade financial instruments such as currencies, stocks and stock indices, and treasury bills (Moody et al. 1998; Moody and Saffell 2001), and Gorse (2011) has also applied reinforcement learning to trading stock indices. Lee (2001) applied the basic reinforcement learning algorithm TD(0) to predict stock prices. Similar work using Q-learning has been applied to stock trading and asset allocation (Nevmyvaka et al. 2006; O et al. 2006). Li and colleagues used a reinforcement learning algorithm called least-squares policy iteration (LSPI) to learn policies for exercising American stock options (Li and Schuurmans 2008; Li et al. 2009). Moore et al. (2014) applied the Q-learning reinforcement learning algorithm (with some minor adjustments) to controlling the delivery of anesthesia. In this work, an *in silico* simulation was used to develop an initial control strategy, which was subsequently refined in a real study.

The number of successful applications of reinforcement learning, however, is far fewer than one would expect since the field's inception and significant foundational works in the early 1980s (Barto et al. 1983; Sutton 1984). Success of this learning method is dependent on the *interaction* between an environment, a learning algorithm, and some form of knowledge representation (Langley 1988), and in general, the reasons why reinforcement learning works in some situations and not in others is not well understood. Successful applications are often rely on heuristics and ad-hoc implementations that require practical and theoretical knowledge about the environment, learning algorithm, and form of knowledge representation, instead of being based on formal proofs, and this approach may be the reason for such mixed results (Tsitsiklis and Roy 1996; Konen and Beielstein 2008). Formal proofs have had a place in reinforcement learning, however (see Sects. 2.2.2 and 2.2.3.2), though the assumptions on which they are based limit their applicability to very simplistic situations (Tsitsiklis and Roy 1996). Because of this, simple problem domains are still used to understand how environmental and problem characteristics affect learning.

These problems are simpler than their real-world and end-goal counterparts, but the insights that can be gained from them can be valuable in achieving the end-goal. For example, the multi-armed bandit problem is where an agent is attempts to maximize its reward from multiple single-armed bandits (i.e., slot machines), where each bandit has a different expected reward. Learning how to maximize performance in this domain requires both exploitation of knowledge and exploration of potentially better actions. This problem scenario may be used to gain an understanding of the dynamics of a problem, and such knowledge has been used to aid in the application of similar methods to problems in energy management or robotics (Galichet et al. 2013; Karnin et al. 2013).

2.1.4 Generalized Domains

There is another class of domains to which reinforcement learning has been applied that deserves attention. These domains are not based on any physical, real-world, or game-like domain. Rather they could instead be considered *generalized domains* that are either completely abstract environments or are environments that have parameterized characteristics. The term *abstract* in this sense refers to an environment which lacks a physical representation and is merely a numerical representation with defined properties, dynamics, and constraints. While all of the environments mentioned in this section have not been used with reinforcement learning specifically, they have been used with some form of sequential decision making-related algorithm.

One of the original purely abstract generalized domains was for infinite horizon problems developed by Archibald et al. (1995) out of a need for a more flexible and widely available platform for assessing the characteristics of Markov decision processes. This domain is highly parameterized, using 14 variables that determine the dynamics of the domain; one of the novelties of this work was that it allowed for the control of the mixing speed of the problem (Aldous 1983), which is related to the relative importance of the starting state of the process. However, this type of domain is based on enumerated states that are used with look-up table representations, rather than state vectors that are commonly used in present day problems.

Garnet (Generalized Average Reward Non-stationary Environment Testbed) problems are a similar type of generalized abstract domain that have been developed more recently, though these domains are for episodic learning problems (Bhatnagar et al. 2009; Castro and Mannor 2010; Awate 2009), as is considered in the present work. The domains generated by Bhatnagar et al. (2009) are based on a simpler, five parameter domain that is defined by: (1) the number of states, (2) the number of actions, (3) the branching factor, (4) the standard deviation of the reward(s), and (5) the stationarity. Two additional parameters are the dimensionality and the number of non-zero components of the state vectors. The domains created by Awate (2009) and Castro and Mannor (2010) omit the stationarity parameter, but otherwise use the same domain construction method.

Gridworld has also served as a base domain for a generalized domain by Kalyanakrishnan and Stone (2009, 2011). These works used a Gridworld-like parameterized domain to evaluate how value function and policy search learning algorithms perform under specific domain variations. These parameterized domains were developed to assess the effects of a small and specific set environment characteristics on the performance of certain learning algorithms, and thus the parameterization of the domains was not very extensive and only relied on 4 parameters: (1) domain size, (2) stochasticity, (3) reward distribution, and (4) observability. In the works cited, these domains restrict the state space to only 48 states, making it possible to benchmark the performance of the learning algorithms on each domain against an optimal action set determined using dynamic programming.

These Gridworld-based domains have a number of attractive properties, including: the smoothness of the value function due to the connectivity of the states; the 2-dimensional representation of the state space allowing for 3-dimensional visualization of the value function; and the small size of these problems allowing for an optimal solution to be found and used as a performance reference. Whiteson and colleagues also advocate for the use of generalized domains in order to guard against overfitting reinforcement learning methods to specific environments, with the goal of improving our understanding of how methods generalize to other domains (Whiteson et al. 2009, 2011). Their work uses some of the classic reinforcement learning benchmark problems such as the mountain car problem, acrobot, and puddle world as base environments. Domain characteristics, such as action perturbations or state observability, are then sampled from distributions, which produces domains with more general characteristics.

2.2 Components of Reinforcement Learning

Reinforcement learning is based on the interaction between three different components or entities (Fig. 2.1). These include: (1) the domain, or the environment in which the agent acts; (2) the learning algorithm, or the procedure by which the agent associates actions with outcomes; and (3) the representation of the learned knowledge, or the agent itself. Figure 2.6 shows each of these three main components and how these entities can be further broken down or specified to different methods. This figure is not comprehensive, especially with regards to the representations and the learning algorithms, and is only used to provide examples of these methods.

2.2.1 Domains

The domain in reinforcement learning is the environment in which the agent interacts. The domain specifies the state space, the action space, and the reward function, and all of these elements have an effect on the dynamics of the behavior of the agent. More concretely, the domain includes applications such as those cited in Sect. 2.1, including benchmark problems, games, and real-world applications. When one considers these

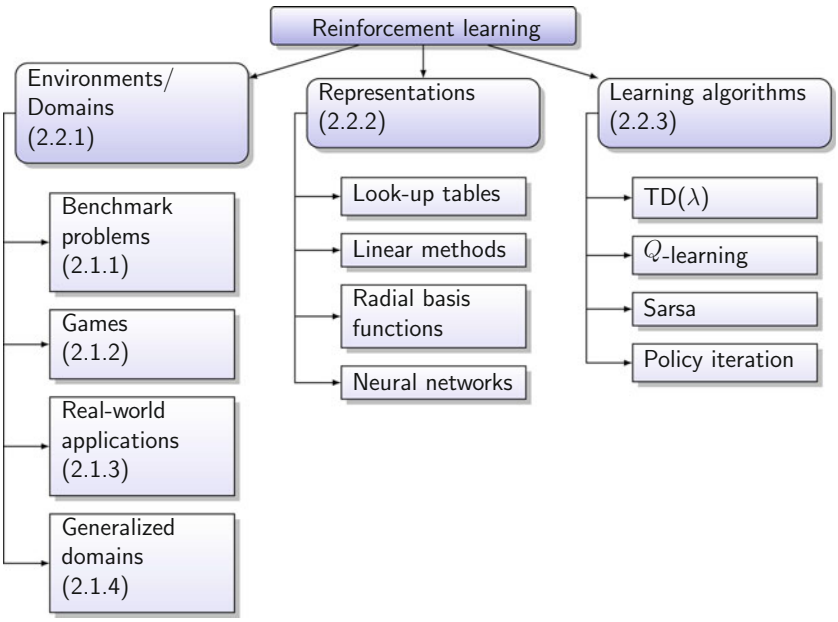


Fig. 2.6 This figure shows how reinforcement learning can be broken down into the environment or domain, the representation of the learned knowledge, and the learning algorithm, all of which are discussed in this work. The numbers specified in parentheses refer to the section number for each of these components in this work.

domains, it is apparent that they all have different characteristics: their state space is different, their action space is different, and the reward structure is different, amongst other things. It should be clear that because the behavior of the agent is dependent on the characteristics of the domain, the ability of the agent to learn and its ultimate performance is similarly dependent on domain characteristics.

Understanding the effects of domain characteristics in reinforcement learning is a fundamental step in the application and extension of these methods to novel and more challenging problems. While some of these characteristics can be plainly acknowledged when considering individual domains or groups of domains, a thorough study of these characteristics requires more formal definitions. This section defines these characteristics over four domain dimensions (Fig. 2.7): general characteristics, state space characteristics, action space characteristics, and reward/penalty function characteristics.

2.2.1.1 General Characteristics

General domain characteristics listed below are termed as *general* because they are related to the decision process at large, or because they are not associated with any of the other dimensions. These characteristics include: the time horizon, the number of agents, stationarity, and observability.

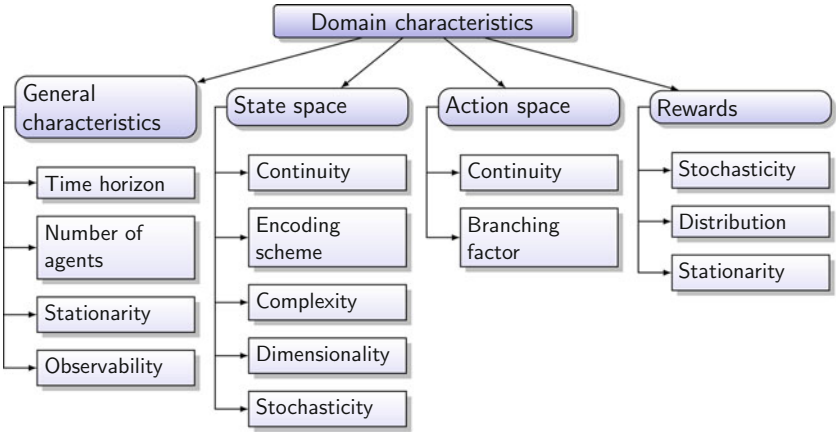


Fig. 2.7 Domains can be characterized by general, state space, action space, and reward characteristics, each of which can be considered at a finer level.

Time Horizon The time horizon refers to the length of time over which learning occurs. There are generally two types of horizons in decision processes: finite and infinite horizons. Finite horizon problems, which are considered in this work and are also known as episodic problems, have some termination criteria that ends an episode, which may be based on a finite time limit or, more often, on reaching some absorbing state. These are the types of problems that classical reinforcement learning considers almost exclusively. Infinite horizon problems have no absorbing state and instead the learning and decision making processes extend infinitely in time. These types of problems are often geared toward real-world or business-like domains, such as truck routing optimization, and are more associated with the stochastic optimization or approximate dynamic programming communities (Powell 2007).

Number of Agents This characteristic refers to the number of agents that are present in the environment. If there are multiple agents, the agents also interact with each other. In many reinforcement learning problems, especially in benchmark problems, there is a single agent (e.g., Gridworld, mountain car, etc.), though control problems also often have only a single agent. In other problems, such as in games, there are often two agents who have opposing goals. In other problems still, there could be more than two agents that compete for their own goal or a single goal, or there could be teams of agents that compete for a single goal or opposing goals (Littman 2001). As stated, this dimension is also tied to the number of goals, which will be defined later. In this work, only single-agent domains are considered.

Domain Stationarity Domain stationarity refers to how any characteristic or structural property of the domain changes over the course of agent-environment interactions. A stationary domain has characteristics that does not change, whereas a non-stationary domain has characteristics that do change with time. It is important to distinguish stationarity from stochasticity, which is used to describe other domain

characteristics. Stationarity refers to a process where the statistical properties of the distribution do not change over time. A domain may have a stochastic element to it that has some underlying distribution, for example, the distribution of sensor noise which affects its perception of the current state. If this distribution does not change over time, this characteristic is stationarity, whereas if it does change over time, this characteristic is non-stationary.

Observability Observability refers to what information is available (observable) to the agent during the learning process. If all of the relevant state information (i.e., information that is necessary to learn how to behave optimally in the domain) is available to the agent, the domain is said to be fully observable. If not all of the relevant state information is available to the agent, the domain is partially observable, and the decision process is no longer considered to be Markovian. Examples of partial observability include missing state information, sensor noise, and perceptual aliasing. Using the cart-pole balancing task as an example, missing state information could be where only the angle of the pole is available to the agent, whereas both the pole angle and angular velocity are traditionally available to the agent in this problem. Sensor noise is where the agent's observation of the state is noisy, e.g., the observed pole angle and angular velocity are different than their true values. Perceptual aliasing is where multiple states look to be the same based on current perceptual information of the agent, and only by using previous state information can these states be differentiated (Bakker et al. 2002).

A large portion of the reinforcement learning literature focuses on fully observable domains because they are simple and amenable to theoretical analysis (Jaakkola et al. 2003), however, Bakker et al. (2002) suggests that partially observable domains are more similar to real world problems. Partial observability can have a significant impact on the ability of an agent to learn. Intuitively, with partial observability, the performance of learning algorithms based on the Markovian properties (e.g., temporal difference methods) suffers due to their underlying assumptions of the environment. As a results, algorithms have been developed to handle partially-observable Markov decision processes (POMDPs) (Baxter and Bartlett 2000), but these are not considered in this work.

2.2.1.2 State Space Dimensions

This section describes characteristics that are related to the state space of the domain, or those that are most closely associated to the numerical representation of the state, the state vector.

State Space Continuity State space continuity refers to whether the state space is discrete or continuous. Many reinforcement learning problems have discrete state spaces (e.g., Gridworld, games, etc.). Those that have continuous state spaces are often based on some (potentially real-world) dynamic system (e.g., mountain car, pendulum balancing, helicopter control, etc.). It is also important to acknowledge the underlying state transition process of discrete and continuous state spaces. State

transitions in discrete state spaces have some (unknown) underlying state transition matrix. State transitions in continuous state spaces rely on system dynamics, such as equations of motion for real-world systems, and these dynamics are specific to the application of interest.

State Space Encoding State space encoding refers to how the state space is numerically represented as a state vector, which is then used with learning algorithms and functional representations. For a given domain, the states can be encoded in any number of ways. The choice or construction of a useful state encoding scheme can reduce the size of the state space and can enable efficient learning. We define a *native* or *raw* encoding scheme to be one that is intuitively obvious to any human. An example of a raw encoding scheme in a two-player board game could represent the presence of pieces on the board using a 1 for one player, -1 for the other, and 0 for when no player is present on a particular board location.

The determination of state encoding schemes, whether explicit or implicit, borders on what is analogous to feature selection in supervised learning problems. Although state encoding has been investigate by some (Mahadevan and Maggioni 2007; Günther 2008; Konen and Beielstein 2008, 2009; Osentoski 2009) and the use of hand-crafted features has been shown to be quite useful (Tesauro 1995; Tsitsiklis and Roy 1996), it has not been given nearly the attention it deserves.

State Space Complexity State space complexity refers to the size of the state space, or the number of possible states that can be visited by the agent. As this work will use a raw encoding scheme, the state space complexity will be based on the number of states when using a raw encoding scheme. While the entire state space may be numerically feasible, it may not be entirely reachable by the agent depending on the allowable actions and the dynamics and constraints of the domain.

State Space Dimensionality State space dimensionality refers to how many elements are used in the state vector to numerically represent the state. This characteristic could be thought of as how much information the agent is given from which it can learn and act, though the relevance or quality of this information is also of great importance. Note that the state space dimensionality is also dependent on the state encoding scheme.

State Transition Stochasticity State transition stochasticity refers to whether or not state transitions are deterministic. More specifically, it refers to the probability of the agent being in some state $x^{(t+1)} = x'$ at time $t+1$ after taking action $a^{(t)}$ from state $x^{(t)} = x$ at time t, or $\mathcal{P}^a_{xx'}$. Deterministic state transitions occur when $\mathcal{P}^a_{xx'} = 1$. Stochastic state transitions occur when $\mathcal{P}^a_{xx'} < 1$, where the total probability of ending up in all next-states $X^{(t+1)}$ is equal to 1. Many domains, including most games, have deterministic state transitions, though there are many domains that have stochastic state transitions. The degree of stochasticity of stochastic state transitions could also take on a range of values. In this sense, state transitions could be highly stochastic where the probability of being in any of the possible next-states are all equal. Alternatively, state transitions could be nearly deterministic when the probability of being in one particular next-state is near, but not equal to, 1, and the probability of being in all other possible next-states is very small.

2.2.1.3 Action Space Dimensions

This section describes characteristics that are related to the action space of the domain, or those that are most closely associated to the actions of the agent.

Action Space Continuity Action space continuity refers to whether the actions of the agent are discrete or continuous. Often there is some relation between the continuity of the state space and the action space, where discrete state spaces have discrete action spaces and continuous state spaces have continuous action spaces, but this is not always the case. The most common state space-action space pairing is the discrete state-discrete action paring, such as in most games. Continuous state domains are often control-type problems or have some underlying real-world dynamics, and these can use either discrete actions (e.g., mountain car domain) or continuous actions (e.g. robot control).

Branching Factor The branching factor refers to the number of actions that can be taken from any state. Some domains have constant branching factors where there is a constant number of possible actions that can be taken from any and all states. Other domains have non-constant branching factors where the number of actions from any state may increase or decrease depending on the state, and the branching factor therefore takes on a distribution over the state space. The branching factor can also be thought of as a form of constraint in the sense that the state trajectory of the agent is somewhat guided or constrained, rather than allowing for the entire state space to be reached from any other state.

2.2.1.4 Reward Dimension

Reward characteristics are specifically related to the reward function. As mentioned, while the term *reward* has a positive connotation, this term is used for any type of feedback provided to the agent and could therefore be aversive (i.e., negative). Loosely, a reward is any concrete information that is provided to the agent that is indicative of the true value or quality of a being in a state or following a trajectory.

Reward Stochasticity Reward stochasticity refers to the whether or not rewards for any particular state are deterministic or stochastic. More specifically, this characteristic specifies if rewards are provided every time the agent visits a particular state or if rewards are provided with some probability. The vast majority of domains use deterministic reward functions where the reward is fixed to a particular state or a group of states. However, domains may also provide rewards a fraction of the time, thus providing relatively less feedback to the agent. Note that this characteristics does not refer to how many states have rewards nor the magnitude of the reward(s), and these characteristics will be defined next.

Reward Distribution The reward distribution refers to how the rewards are distributed over the state space, as well as the magnitude of these rewards. Some domains have a single reward state (e.g., the mountain car problem), whereas other domains

have multiple states that provide rewards of equal magnitudes, such as in Tic-tac-toe where there are many ways to win the game. Still, other domains may have multiple reward states where the magnitude of the rewards may depend on the state, as in the game of backgammon where there are different types of wins.

Though there could be countless types of reward distributions, we define a number of cases that represent some of those found in common domains. (1) There is a single state with a positively-valued reward. (2) There is a single state with a positively-valued reward and a single state with a negatively-valued reward, each of the same magnitude. (3) There are multiple states with either positively-value or negative-valued rewards, where all rewards are of the same magnitude. (4) There are multiple states with either positively-value or negative-valued distributions of rewards.

A special case of a reward distribution is that of a reward hierarchy. In this case rewards are provided en route to the primary goal state in order to shape or guide the agent's trajectory in large and complex domains. Essentially, relatively smaller rewards are placed within the domain acting as subgoals, and these subgoals are used to lead the agent to the main goal of the domain where there is some large positively-valued reward. One example of this is in the hierarchical office domain by Bakker and Schmidhuber (2004) for which a specific learning algorithm was developed.

Reward Stationarity Reward stationarity refers to how the reward distribution changes over time, thus adding a temporal dimension to the reward distribution. In most domains, rewards are stationary and do not change over time. However, it is possible that rewards could be non-stationary, such as when rewards decrease over time when attempting to encourage the agent to reach a goal more efficiently.

2.2.1.5 State Encoding-Dependent Characteristics

As noted, some of the characteristics defined above are dependent on the state encoding scheme. Most evidently, the state encoding scheme could affect the state space dimensionality and the state space complexity. This may or may not change the branching factor depending on whether or not actions are derived from the raw state encoding or from a novel state encoding. While novel state encoding schemes will not be considered in this work, it is important to acknowledge that the state encoding scheme has an effect on a number of characteristics, and this changes the dynamics of the learning problem.

2.2.2 Representations

Reinforcement learning requires the use of some form of *memory device* in order to retain and update state values. The type of memory device, or *representation* as it will be called here, is often chosen based on qualities of the environment that seem to pair well with the representation. There have been many different representations used in reinforcement learning, with some more dominant than others, and this section briefly reviews some of the different representations used, including look-up tables, linear methods, and neural networks.

2.2.2.1 Look-up Tables

Look-up tables are the most basic form of representation for reinforcement learning. These tables store informative (i.e., state values) for each and every state. The states used in the look-up table may use either the raw state space or derived/hand-coded features as look-up indices. This type of representation could be considered the most localized type of representation because of the unique state-value mapping, and thus state value updates only influence individual states that are visited at a time. State values of all unvisited states are not affected, which makes this representation immune to updates of neighboring or nearby states. The properties of look-up tables can be rigorously analyzed and it can be proved that some learning algorithms converge under various conditions and assumptions, such as each state being sampled infinitely many times and the learning rates be set appropriately (Watkins 1989; Watkins and Dayan 1992; Jaakkola et al. 2003; Tsitsiklis and Roy 1996).

Despite the attractive properties of look-up tables, they do have some disadvantages. The primary disadvantage of look-up tables is that they are limited to relatively small and discrete state spaces due to memory constraints of computers as well as due to the computationally intensive task of having to search through all entires in the table (Baird 1995; Atkeson et al., 1997). In some cases, a simpler, and more compact representation (e.g., neural network) can accurately model the value function instead, and it can do so with far fewer parameters than unique parameters required for a look up table (Moody and Tresp 1994; Ollington et al. 2009). For reinforcement learning applications to be scaled to real-world problems, more compact representations are required than look-up tables (Singh et al. 1995). However, large state spaces can be handled by look-up tables using some approximation methods, one of which includes state aggregation, in which similar states are assumed to have similar values (Tsitsiklis and Roy 1996). Look-up tables can also be use for continuous state spaces, although states must be discretized in some manner and this may not be straightforward. Another disadvantage of look-up tables is that learning can be slow due to the shear number of states and the fact that state values are updated individually; in other words, look-up tables have no ability to generalize the state value updates (Tesauro 1992; Ollington et al. 2009).

Perhaps the earliest use of look-up tables for reinforcement learning might be the work of Michie and Chambers (1968) in which 162 'boxes' (i.e., entries in the look-up tables) are used to approximate the value function for the cart-pole balancing task. While look-up tables could be viewed as a relatively primitive representation, they have not been completely replaced by other representations and they are still used in contemporary research. Whiteson and Stone (2006) used a look-up table for adaptive job routing/scheduling, and Nevmyvaka et al. (2006) used this representation for optimizing trade execution. A more recent application proved that look-up tables can converge to the optimal policy for best-match learning, a hybrid sparse model-based and model-free Q-learning approach (van Seijen et al. 2011). Additionally, Nissen (2007) used look-up tables for playing blackjack using multiple learning algorithms including $Q(\lambda)$, Sarsa(λ) and Q-Sarsa(λ). Another use of look-up tables is for comparative purposes for benchmarking other representations. Wiering (1995)

used look-up tables with Q-learning to learn the game of Tic-tac-toe, and although this representation required $\sim$ 4600 parameters, it also achieved the best performance out of multiple representations evaluated. Wiering et al. (2007) and Wiering and van Hasselt (2008) used a look-up table to benchmark the performance of neural networks for a maze navigation problem.

2.2.2.2 Linear Methods

When the state space is either continuous or grows to the size where a look-up table is neither practical nor effective, some form of function approximator with a relatively small number of parameters may be used instead. This representation serves the same purpose as a look-up table, that is, to learn the true state values, but it is able to generalize between states and thus all states do not have to be visited to estimate their state value. The most widely used approach is the linear function approximator, in which the representation takes the general form that the value function is based on a linear combination of the state features and learned weights. The simplest linear approximators are based on hand-crafted features of the state space, and this can be successful in very simple problems.

One of the prominent linear methods used in reinforcement learning is tile coding, which consists of multiple (and possibly irregular) tilings that overlap and cover the entire state space (Albus 1975; Barto 1990; Sutton and Barto 1998). This method can be thought of as a discretized radial basis function (RBF) network, and it therefore has the ability to generalize to small and localized regions of the state space (Skelly 2004). This method has been used in many situations, including with benchmark problems (Sutton 1996; Atkeson and Santamaría 1997; Kretchmar and Anderson 1997; Kalyanakrishnan and Stone 2007; Främling 2008) as well as for algorithm/representation comparison and benchmarking (Lin 1992; Kretchmar and Anderson 1997; Whiteson and Stone 2006; Främling 2008; Sutton et al. 2009b). Despite the success of this approach, the construction of the coding scheme requires *a priori* domain-specific knowledge and is not often straightforward (Ollington et al. 2009; Whiteson et al. 2011), and this approach becomes intractable with high-dimensional state spaces (Szepesvári 2010). A similar approach to tile coding is state aggregation in which similar states are grouped using a variety of methods, such as fuzzy clustering (Singh et al. 1995) or adaptive partitioning (Bonarini et al. 2007).

Though radial basis function networks (RBFs) (Orr 1996) could be considered a neural network, they are included here as a linear method because the state values are a linear combination of the 'features' computed by the basis functions. The traditional use of RBFs in reinforcement learning distributes the centers of the basis functions over the entire state space. Basic implementations use fixed basic centers, though this approach often requires manual tuning (Sutton and Barto 1998), however, adaptive schemes can be used to update the locations or widths of the basis functions (Castro and Mannor 2010). RBFs are attractive because they have the ability to both generalized and make local approximations, and this is particularly useful

when the value function is not well-behaved and has discontinuities (Konidaris et al. 2011). RBFs have been used for benchmark problems (Thrun and Schwartz 1993; Doya 1996, 2000; Främling2008; Kalyanakrishnan and Stone 2009; Konidaris et al. 2011) and have been used for representation comparisons (Kretchmar and Anderson 1997; Mahadevan and Maggioni 2005; Främling 2008), as well for more real-world problems, such as robot control (Papierok et al. 2008).

The primary reasons why linear function approximators are attractive and so dominantly used for reinforcement learning is that they are easy to implement, the parameter updates are relatively simple, and because they are efficient from a data and computational perspective (though their efficiency does depend on the state encoding scheme). Another reason why linear methods are attractive is that they are amenable to theoretical analysis and some learning algorithms have been proven to converge under certain circumstances. Tsitsiklis and Roy (1996) proved that (on-policy) temporal difference algorithms with linear function approximators converge to an approximation of the optimal policy, and Gordon (2001) proved that Sarsa with a linear function approximator converges to a bounded region. Despite the attractiveness of theoretical analysis, the underlying assumptions are quite restrictive and often are not valid in even simple domains such as benchmark problems, much less in real-world problems. Some of these assumptions include that there is no adaptive exploration (which has been found to be extremely effective), that the learning rates be set appropriately, and a rather restrictive assumption on the choice of state features (Tsitsiklis and Roy 1996), which may be either hand-crafted or derived in some manner based on interactions with the domain.

Despite the convergence guarantees and ease of implementation, the success of linear methods with reinforcement learning is dependent on the state features or state encoding scheme. In simple domains, such as in benchmark problems and simple games, state features that are relatively basic and intuitively obvious can be sufficient. In more challenging domains that have larger state spaces or complex value functions, however, hand-crafted features developed from user experience or that exploit the domain are required (Tsitsiklis and Roy 1996), and this is not a trivial task (Powell 2008). Hand-crafting features is often a manual trial-and-error process, though there have been attempts to develop automated methods for feature construction (Günther 2008) or basis function adaptation (Menache et al. 2005). The works of Mahadevan and Maggioni (2007) and Osentoski (2009) are particularly interesting and rely on feature generation through spectral analysis of an empirically-determined state transitions matrix, which can be viewed as a network graph.

2.2.2.3 Neural Networks

Neural networks are an alternative representation that has been used in a variety of capacities for reinforcement learning. The most natural use of neural networks is for learning state value functions (which will be discussed in Sect. 2.2.3). A basic implementation uses a single feedforward neural network, where the input is the current state vector and the output is the estimated value of the current state. This

approach can be extended and modified to have multiple outputs for multiple actions, or to have multiple neural networks to represented either different actions or different regions of the state space (Wiering 1995; Kaelbling et al. 1996; Lazaric 2008; Kalyanakrishnan and Stone 2009). Out of these rather rudimentary approaches, none have been found to be superior for all applications, though many implementations have the common quality that they use networks with a single hidden layer (Ghory 2004).

The use of neural networks for reinforcement learning is attractive for a number of reasons. The first is that, unlike look-up tables but similar to linear methods, they can generalize state values to states that have not been explicitly visited. In domains with smooth and well-behaved value functions, this can be extremely useful and can reduce the amount of training required to learn the domain. The second reason is that neural networks are parameterized by a relatively small number of parameters, especially when compared to the size of the state space of some domains. Finally, neural networks do not necessarily require the use of carefully hand-crafter state features as inputs. Rather, the hidden layer(s) of the neural network is(are) able to derive implicit features that are deemed to be useful (Konen and Beielstein 2009), though these derived features cannot often be interpreted (Günther 2008).

Despite these attractive properties of neural networks, neural networks are not the dominant representation for most of the reinforcement learning community for a number of reasons. Replacing a look-up table with a complex function approximator, such as a neural network, is not trivial and is viewed by some as not being robust (Boyan and Moore 1995). The convergence proofs of reinforcement learning algorithms for linear methods have not been extended to non-linear function approximators (Tsitsiklis and Roy 1996, 1997), and these learning algorithms may find suboptimal solutions or may diverge (Boyan and Moore 1995; Bertsekas and Tsitsiklis 1996). The primary reason for this is that nonlinear methods tend to exaggerate small changes in the target function, and this exaggeration causes the value iteration algorithm to become unstable (Thrun and Schwartz 1993; Gordon 1995). It has also be suggested that the use of state discounting in reinforcement learning can potentially lead to instability with value iteration algorithms (Thrun and Schwartz 1993).

The successful use of a neural network is dependent on the properties of the underlying value function as well as on the ability of the neural network to approximate the value function (Thrun and Schwartz 1993; Dietterich 2000). Though the ability of a neural network to generalize to unvisited states is attractive, this property can be unfavorable when the value function is not smooth or is not well-behaved, which may require longer training or could result in the learning algorithm diverging (Riedmiller 2005). Such situations may arise when the numerical representations of the state vectors are close together, yet their state values are quite different (Dayan 1993; Mahadevan and Maggioni 2007; Osentoski 2009). It has been suggested that the dynamic capability of a neural network is dependent on its architecture (Loone and Irwin 2001; Igel 2003), and thus a poorly chosen network architecture may either perform poorly or may diverge during training (Hans and Udluft 2010), and others claim that the neural network architecture must be tailored for the application (Günther 2008).

A successful implementation of a neural network for reinforcement depends on far more than just the architecture, however. The practical application of using neural networks for reinforcement learning is not straightforward, and there are many cases in the literature which cite difficulties in doing so, either resulting in suboptimal and mixed results (Sutton 1996; Wiering and van Hasselt 2007), or a complete failure to learn (Chapman and Kaelbling 1991; Thrun and Schwartz 1993; Proper and Tadepalli 2006). It is important to acknowledge though, that the use of rather standard neural network parameters and settings can result in successful applications in reinforcement learning (Gatti et al. 2011a, b; Gatti and Embrechts 2012).

The most basic training algorithm for reinforcement learning using a neural network is value iteration with the temporal difference algorithm (see Sect. 2.2.3), and this learning algorithm will be the algorithm used in this work. However, other algorithms have been used or developed for reinforcement learning using a neural network. While a comprehensive review of all neural network-related training algorithms for reinforcement learning is beyond the scope of this work, we do mention a few of these methods. The value iteration algorithm, which attempts to learn a value function based only on state information, can quite easily be extended to Q-learning, which attempts to learn a value function based on paired state and action information (Watkins 1989). Neural fitted Q-iteration is a form of Q-learning that was developed for neural networks (Riedmiller 2005), though it could be used with other function approximators as well. This algorithm is novel in that it stores and reuses experiences, and that training is done in a batch manner, which makes it similar in some respects to a supervised learning approach (Lange et al. 2012). NeuroEvolution Augmenting Topologies (NEAT) is a method that evolves the structure of a neural network using a genetic algorithm, and thus alleviates some of the difficulties in selecting a network architecture *a priori* (Stanley and Miikkulainen 2002; Whiteson and Stone 2006; Whiteson et al. 2010). Additionally, neural networks can also be used with many algorithms developed for general function approximators, such as with modified value iteration algorithms, actor-critic methods, or policy gradient methods (Sutton et al. 2000).

A standard implementation of reinforcement learning with a neural network uses a feedforward neural network, and this has been successful in many cases (Tesauro 1995; Bakker and Schmidhuber 2004; Mannen and Wiering 2004; Patist and Wiering 2004; Riedmiller 2005; Wiering et al. 2007; Wiering and van Hasselt 2007; Wiering 2010; Papahristou and Refanidis 2011; Gatti and Embrechts 2012). There are also other types of neural networks or neural network methods that have been used in reinforcement learning as well. Ensembles of neural networks have been used in different capacities, such as using individual networks for separate regions of the state space (Wiering 1995), or using an ensemble of neural networks to either select actions by majority voting or estimate an average value function (Hans and Udluft 2010, 2011). Recurrent neural networks (Elman 1990) are an attractive alternative due to their ability to learn complex temporal sequences, and such networks have been used in a number of applications (Moody et al. 1998; Grüning 2007; Makino 2009). Though the long short-term memory model (Hochreiter and

Schmidhuber 1997) with recurrent neural networks has been shown to have considerable promise over basic recurrent neural networks (Gers 2001; Bakker 2001, 2007; Wierstra et al. 2010). Other types of neural networks that have been used for reinforcement learning include modular neural networks (Schraudolph et al. 1994), cascading neural networks (Nissen 2007), self-organizing maps (Touzet 1997; Smith 2002; Tan et al. 2008; Montazeri et al. 2011; Osana 2011), and explanation-based neural networks (EBNN) (Mitchell and Thrun 1992; Thrun 1995).

In spite of the implementation challenges and the lack of convergence proofs, neural networks are still a commonly used function approximator for applications of reinforcement learning. A large proportion of applications are to benchmark or toy domains, such as the inverted pendulum problem (Anderson 1987), single and double cart-pole balancing (Igel 2003; van Hasselt and Wiering 2007; Hans and Udluft 2010), the pole swing-up task (Gabel et al. 2011), keep-away (Whiteson et al. 2010), the mountain car problem (Wiering and van Hasselt 2007; Whiteson et al. 2010), and maze problems (Wiering and van Hasselt 2007). Neural networks are a common representation for learning games due to their ability to generalize, and applications include games such as Tic-tac-toe (Wiering 1995; Konen and Beielstein 2009; Gatti and Embrechts 2012), Chung-Toi (a variant of Tic-tac-toe) (Gatti et al. 2011a, b), checkers (Patist and Wiering 2004; Wiering et al. 2007), Chess (Thrun 1995; Mannen and Wiering 2004; Wiering et al. 2007), Othello (van Eck and van Wezel 2008), Go (Runarsson and Lucas 2005), and Backgrammon (Tesauro 1995; Wiering et al. 2007; Wiering 2010; Papahristou and Refanidis 2011). Finally, the use of neural networks for reinforcement learning in the real-world (including real-world conceptual problems) includes applications such as control problems (Werbos 1989; Mitchell and Thrun 1992; Yamada 2011), stock price prediction (Lee 2001) and trading (Gorse 2011), product delivery and distribution (Proper and Tadepalli 2006), resource allocation (Tesauro et al. 2007), jobshop scheduling (Gabel and Riedmiller 2007), and technical process control (Hafner and Riedmiller 2011).

2.2.3 Learning Algorithms

The goal of learning algorithms in reinforcement learning is essentially to allow the agent to learn the dynamics of the environment so that an optimal set of actions may be selected to achieve a goal or obtain the greatest total reward. While numerous conceptual approaches, and thus numerous learning algorithms, have been developed for learning this environment-action mapping (see Sutton and Barto (1998); Szepesvári (2010); Powell and Ma (2011) for comprehensive reviews of learning algorithms), in this work we focus on value function learning methods. In these methods, the agent attempts to learn a value function that approximates the value (i.e., utility) of states, though we briefly mention other reinforcement learning algorithms that have been developed.

The representation (i.e., agent, defined in Sect. 2.2.2) serves two purposes in reinforcement learning. The first purpose is to learn about the dynamics of the domain and how the selection of actions relate to feedback delivered by the domain. In other

words, the purpose of this task is to learn an input-output mapping between the state space X, the action set A, and the feedback from the domain R. The same representation can be used for the second purpose, which is to evaluate the possible actions and select a single action to pursue. The learning algorithm we focus on uses a single entity to perform both of these functions, although algorithms may use distinct components for each of these tasks.

This section describes one of the fundamental learning algorithms used in this work, the temporal difference algorithm (Sutton and Barto 1998), which is considered a value function learning algorithm. That is, in its most basic form, each state is believed to have a utility value, and the function that is learned attempts to approximate the true, underlying value of each state. States with large values are more likely to result in greater rewards or a successful outcome, whereas states with small values (potentially negative) are more likely to result in smaller (potentially negative) rewards or unsuccessful outcomes. Initially, the agents' state value estimations are erroneous because it has no knowledge of the environment. The accuracy of the state value estimates improve by the agent repeatedly interacting with the environment, during which the behavior, or the action policy, of the agent emerges from its learned knowledge of the state values.

This work uses feedfoward neural networks as the agent representation, and this sections describes the implementation details of value function learning, and specifically the temporal difference algorithm, for a feedforward neural network. The value function learning algorithm that we consider attempts to learn the value function $V(x)$ and takes the general form:

$$V(x) = \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t R_{t+1} \mid x\right], \qquad x \in X$$

where the value of state x is the expected long-term discounted reward when starting in state x. Note that this expression is quite similar to the Bellman equation (Eq. 2.3), and thus by attempting to learn the true value function of the domain, the neural network implicitly solves the Bellman equation.

The temporal difference algorithm can be thought of as an extension to the back-propagation algorithm (Rumelhart et al. 1986; Werbos 1974) that is used to train neural networks, for example, for classification or function approximation, by solving the credit-assignment problem. In other words, the network weights are individually adjusted based on how much each weight contributed to the error of the network output. The temporal difference algorithm on the other hand, solves the *temporal* credit-assignment problem, which implies that there is a temporal dimension to the credit assignment problem. In essence, this method uses feedback from past time steps to update network weights at current time steps. As there is considerable similarity between the back-propagation algorithm and the temporal difference algorithm, the back-propagation algorithm will be briefly reviewed and will then be extended to the temporal difference algorithm. Both of these algorithms will be based on the 3-layer neural network configuration shown in Fig. 2.8, but their use extends to networks will multiple hidden layers.

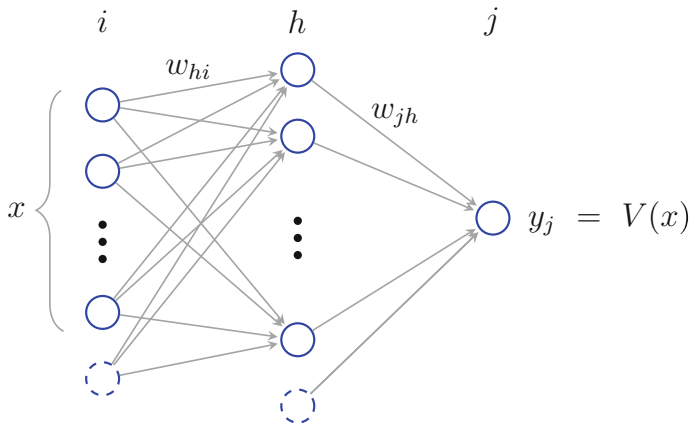


Fig. 2.8 The general structure of the feedforward neural network has an input layer i, a hidden layer h, and an output layer j, as well as bias nodes (*dashed nodes*). Weights are defined such that w_{hi} represents the weight from a node in layer i to a node in layer h, and that w_{jh} represents the weight from a node in layer h to a node in layer j. The value $V(x)$ of a state vector x is computed by forward propagation.

The weight update equation of the back-propagation algorithm for a weight w_{jh} of a neural network (a weight from a node in layer h to a node in layer j) takes the general form:

$$\Delta w_{jh} = \alpha \times E \times \delta_j \times y_h \tag{2.4}$$

where the learning parameter α modulates the magnitude of the weight adjustment, E is the prediction error, δ_j is the local gradient that is based on the derivative of the transfer function evaluated at the node in layer j, and y_h is the output of hidden node h (which is also the input to output node j) and is computed as $y_h = f(v_h)$ where the induced local field is $v_h = \sum_i w_{hi} y_i$ and $f(\cdot)$ is a transfer function. The prediction error from this network is stated as $E = (y_j^* - y_j)$ where y_j is the value of output node j and y_j^* is the corresponding target output value. The expression for Δw_{jh} can be written more explicitly using the partial derivative of the network error E with respect to the network weights:

$$\begin{aligned}
\Delta w_{jh} &= -\alpha \frac{\partial E}{\partial w_{jh}} \\
&= -\alpha \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial v_j} \frac{\partial v_j}{\partial w_{jh}} \\
&= \alpha \, (y_j^* - y_j) \, f'(v_j) \, y_h
\end{aligned}$$

where $f'(v_j)$ is the derivative of the transfer function evaluated for the induced local field v_j. This weight adjustment expression can be extended for updating the

weights Δw_{hi} connecting nodes from input layer i to nodes in hidden layer h and can be expressed as:

$$\begin{aligned}\Delta w_{hi} &= -\alpha \frac{\partial E}{\partial w_{hi}} \\ &= -\alpha \frac{\partial E}{\partial y_h} \frac{\partial y_h}{\partial v_h} \frac{\partial v_h}{\partial w_{hi}} \\ &= \alpha \, (y_j^* - y_j) \, f'(v_j) \, w_{jh} \, f'(v_h) \, x_i\end{aligned}$$

where v_h is the induced local field at hidden node h and x_i is the output of input node i (which is also the input to input node i). Note that the neural network in Fig. 2.8 has a single output node, and thus the network error E is computed from only this output node. This is a slight simplification of the general form of the neural network with multiple output nodes, in which case the back-propagation algorithm propagates all errors at the output nodes in layer j back to the hidden layer h.

The back-propagation algorithm can be extended to the temporal difference algorithm with a simple modification of the weight update equations. Using the example of a game play scenario, the basic implementation of this algorithm uses an interactive update scheme and adjusts network weights following every play during the game. This algorithm works by discounting state value prediction errors by a temporal difference factor λ, where the general form extends Eq. 2.4 and can be stated as:

$$\left(\Delta w_{jh}\right)^{(t)} = \alpha \times (E)^{(t)} \times \sum_{k=0}^{t} (\lambda)^{t-k} \times (\delta_j)^{(k)} \times (y_h)^{(k)} \tag{2.5}$$

which adds a temporal discount factor λ and a summation from the initial time step $t = 0$ up to the current time step t. The weight updates occur at every time step during an episode up to and including the terminal time step $t = T$. Due to the addition of the discount factor λ, this algorithm is referred to as TD(λ). Similar to the notation used for describing the Markov decision process, the parenthetic-superscript (e.g., $r^{(t)}$) is again used to indicate the particular time step t of the corresponding variable. The parenthetic-superscript convention is used to distinguish the values of variables at a particular time step from exponents and subscripts that also may be associated with variables. Thus, superscripts enclosed in parentheses are *not* exponents but merely refer to the value or a variable at a particular time step; however, superscripts that are not enclosed in parentheses are exponents, as in the case of λ^{t-k} for example.

In Eq. 2.5, the prediction error E is also modified from that of the back-propagation algorithm. This error, also known as the temporal difference error, is instead based on the difference between the predicted state values at $t + 1$ and t, as well as the reward received:

$$E^{(t)} = \left(r^{(t+1)} + \gamma V^{(t+1)} - V^{(t)}\right) \tag{2.6}$$

where $x^{(t)}$ is the state vector at time t, $r^{(t+1)} = R(x^{(t+1)})$ is the reward provided for transitioning to state $s^{(t+1)}$, $V^{(t)} = V(x^{(t)})$ and $V^{(t+1)} = V(x^{(t+1)})$ are the predicted

state values at the current and subsequent time steps, respectively. More explicitly, we can write $x^{(t+1)} = \pi_\epsilon(x^{(t)})$, which indicates that the state $x^{(t+1)}$ is a function of the ϵ-greedy action selection policy π_ϵ, as is used in this work. More generally though, any action selection policy π may be used, in which case we could simply write $x^{(t+1)} = \pi(x^{(t)})$. An ϵ-greedy action selection policy chooses the action that results in the next-state $x^{(t+1)}$ with the greatest value 100ϵ % of the time (where ϵ ranges over [0, 1]). In the other $100(1-\epsilon)$ % of the time, random actions are chosen regardless of the values of the next-states. Thus, this policy allows for the agent to exploit its knowledge 100ϵ % of the time, but also to explore potentially better actions $100(1-\epsilon)$ % of the time.

The values of $V^{(t)}$ and $V^{(t+1)}$ are determined by evaluating the respective state values ($x^{(t)}$ and $x^{(t+1)}$) through the neural network using forward propagation, where the next-state vector $x^{(t+1)}$ is determined based on an action selection procedure and the dynamics of the domain. This expression for the temporal difference error also discounts the subsequent state value $V^{(t+1)}$ by a factor γ, which serves to attenuate the value that the network is attempting to learn. Note that the temporal difference algorithm gets its name from this error expression, as it is based on the difference in the predicted state values at two different time steps.

The general form of the TD(λ) algorithm can be more explicitly written for updating the network weights such that $w \leftarrow w + \Delta w$. The weight updates between nodes in the output layer j and nodes in the hidden layer h ($\Delta w_{jh}^{(t)}$) at time step t can be stated as:

$$\Delta w_{jh}^{(t)} = \alpha \left(r^{(t+1)} + \gamma V^{(t+1)} - V^{(t)}\right) \sum_{k=0}^{t} \lambda^{t-k} f'\left(v_j^{(k)}\right) y_h^{(k)} \tag{2.7}$$

where $f'(v_j^{(k)})$ is the derivative of the transfer function at node j evaluated at the induced local field $v_j^{(k)}$ at time step k. Equation 2.7 can then be extended to updating the weights between nodes in the hidden layer h and nodes in the input layer i ($\Delta w_{hi}^{(t)}$) at time step t as (again with a single output node):

$$\Delta w_{hi}^{(t)} = \alpha\left(r^{(t+1)} + \gamma V^{(t+1)} - V^{(t)}\right) \sum_{k=0}^{t} \lambda^{t-k} f'\left(v_j^{(k)}\right) w_{jh}^{(t)} f'\left(v_h^{(k)}\right) x_i^{(k)} \tag{2.8}$$

A basic implementation of the TD(λ) algorithm requires only the use of Eqs. 2.7 and 2.8. Extending these equations using some relatively simple techniques however, can significantly reduce the computational cost in terms of both time and space, and this can improve the efficiency of the learning algorithm.

The use of a momentum term with coefficient η can be added to Eqs. 2.7 and 2.8 in order to incorporate a portion of the weight update from the previous time step $t-1$ into that for the current time step t. This has the effect of smoothing out the network weight changes between time steps and is often most effective when training the network in a batch manner. Batch training is where weight updates are computed during every time step, but the updates are only applied after every n time steps where

$n > 1$ and could extend across multiple episodes. After adding the momentum term, Eqs. 2.7 and 2.8 become, respectively:

$$\Delta w_{jh}^{(t)} = \eta \Delta w_{jh}^{(t-1)} + \alpha \left(r^{(t+1)} + \gamma V^{(t+1)} - V^{(t)} \right) \sum_{k=0}^{t} \lambda^{t-k} f' \left(v_j^{(k)} \right) y_h^{(k)} \tag{2.9}$$

$$\Delta w_{hi}^{(t)} = \eta \Delta w_{hi}^{(t-1)} + \alpha \left(r^{(t+1)} + \gamma V^{(t+1)} - V^{(t)} \right) \sum_{k=0}^{t} \lambda^{t-k} f' \left(v_j^{(k)} \right) w_{jh}^{(t)} f' \left(v_h^{(k)} \right) x_i^{(k)} \tag{2.10}$$

There are a few important notes about the above two equations. The error term, the difference between subsequent state values $V^{(t+1)}$ and $V^{(t)}$ (neglecting the reward term for now), is essentially the information (i.e., feedback) that is used to update network weights. Drawing from terminology of the back-propagation algorithm, $V^{(t+1)}$ can be considered the target output value y_j^*, and $V^{(t)}$ can be considered the predicted output value y_j. The network error is therefore based on the next state value $V^{(t+1)}$, in spite of the fact that this value is merely an estimate and may actually be quite different than the true state value. When weight updates become small, it is possible that numerical errors could affect the final network weights. However, it is likely that the error between the approximated value function by the neural network and the true (unobserved) value function will dominate any numerical error, and thus numerical error is not a large concern.

The general form of a sequential decision making processes proceeds over $t = 0, 1, \ldots, T$. For all intermediate (i.e., non-terminal, $t \neq T$) time steps, the next-state values $V^{(t+1)}$ are available based on their predicted value after pursuing an action; an associated reward $r^{(t+1)}$ may also be provided at these time steps as well, and the incorporation of rewards at intermediate time points is dependent on the specific problem. For example, some problems have well-defined subgoals that must be achieved en route to an ultimate goal. In such cases, non-zero reward values may be provided when these subgoals are achieved. In other problems such as board games, there are often no such subgoals, and thus the reward at all intermediate time steps $t = 0, 1, \ldots, T - 1$ is 0, and the temporal difference error is then only based on the difference between subsequent state values (i.e., $\gamma V^{(t+1)} - V^{(t)}$).

At the terminal time step $t = T$, there is no next-state value $V^{(t+1)}$, and this value is set to 0. The reward at this time step $r^{(t+1)}$ is non-zero however, and the temporal difference error is then based on the previous state value and the reward value (i.e., $r^{(t+1)} - V^{(t)}$). For problems in which there are no intermediate rewards, the reward provided at time step T is the only information that is known with complete certainty, and thus this is the only true information from which the neural network can learn the values of all previously visited states.

Equations 2.9 and 2.10 require that information (terms within the summation) from all previous states $x^{(0)}, \ldots, x^{(t-1)}$ be used to determine the appropriate weight adjustment at state $x^{(t)}$. At time step t, information from all previous states is merely discounted by λ. This can be exploited to reduce the number of computations required

for each weight update, which can be significant for episodes with many time steps. Weight updates at time t can be made based only on information from the current time step t as well as a cumulative aggregation of the information from previous time steps up to $t-1$. Let the summation in Eqs. 2.9 and 2.10 be called g. For each network weight there is a corresponding value of g, which is initially set to 0 at the beginning of each game. The value of $g^{(t)}$ at the current time step t can be computed by discounting its previous value $g^{(t-1)}$ by λ and adding it to the current state's weight adjustment. The values of g for the hidden-output weights $g_{jh}^{(t)}$ and the input-hidden weights $g_{hi}^{(t)}$ become, respectively:

$$g_{jh}^{(t)} = f'\left(v_j^{(t)}\right) y_h^{(t)} + \lambda g_{jh}^{(t-1)} \tag{2.11}$$

$$g_{hi}^{(t)} = f'\left(v_j^{(t)}\right) w_{jh}^{(t)} f'\left(v_h^{(t)}\right) x_i^{(t)} + \lambda g_{hi}^{(t-1)} \tag{2.12}$$

Replacing the summations in Eqs. 2.9 and 2.10 with g as in Eqs. 2.11 and 2.12, respectively, yields the following weight update equations:

$$\Delta w_{jh}^{(t)} = \eta \Delta w_{jh}^{(t-1)} + \alpha \left(r^{(t+1)} + \gamma V^{(t+1)} - V^{(t)}\right) g_{jh}^{(t)} \tag{2.13}$$

$$\Delta w_{hi}^{(t)} = \eta \Delta w_{hi}^{(t-1)} + \alpha \left(r^{(t+1)} + \gamma V^{(t+1)} - V^{(t)}\right) g_{hi}^{(t)} \tag{2.14}$$

The reinforcement learning framework, and the TD(λ) algorithm described above proceeds in a simulation-like fashion with the agent repeatedly interacting with the environment for a specified number of episodes N. This process is outlined in Algorithm 1.

ALGORITHM 1: The TD(λ) algorithm using an iterative weight updating scheme. Weight changes Δw are performed according to Equations 2.9 and 2.10.

Require:
Initialize neural network weights w

for N episodes **do**
  Initialize state x;
  while x is not a terminal state **do**
    $a^{(t)} \leftarrow$ action selected by policy π for $x^{(t)}$;
    Take action $a^{(t)}$, observe reward $r^{(t+1)}$ and next state $x^{(t+1)}$;
    $E \leftarrow r + \gamma V^{(t+1)} - V^{(t)}$;
    $w \leftarrow w + \Delta w$ where $\Delta w = f(E)$;
    $x^{(t)} \leftarrow x^{(t+1)}$;

Return: Neural network weights w

Value function learning is only one approach that has been developed for learning how to behave in sequential decision making problems, and this work will focus exclusively on value function learning using the temporal difference algorithm. However, it is important to acknowledge that there are other learning algorithms used in this

field that make slight modifications to temporal difference methods or that use a different or extended conceptual model of learning. The reader is directed to Sutton and Barto (1998), Szepesvári (2010), Powell and Ma (2011), and Dann et al. (2014) for a comprehensive review of many algorithms used for reinforcement learning.

2.2.3.1 Policy Evaluation Approaches

The temporal difference learning approach is a general concept that is used for learning how to perform in sequential decision making problems. Here, we describe two possible approaches to using TD(λ) that are conceptually different in what is actually being learned.

The simplest use of TD(λ) is exactly what we have previously described. The neural network would take in a state vector, and it would output a single value. That is, the input to the neural network is a state vector of the current state $x^{(t)}$, and the output of the network $V^{(t)}$ is the value of being in that state. Similarly, the value of being in state $x^{(t+1)}$ can be evaluated by inputting $x^{(t+1)}$ to the network and obtaining a value $V^{(t+1)}$. The action selection policy requires all next-states to be evaluated through the network individually, and then selecting one action from the set of possible actions using, for example, and ϵ-greedy action selection procedure. For simple problems where the state transitions are deterministic and known, and thus the next-states are known, such as in Gridworld, this approach can be used and it works.

However, in more complex problems, especially control-type problems or with continuous dynamics or some form of randomness in the environment, it is not always possible to know what the next-states are even if it is known what actions can be performed. Consequently, one cannot simply evaluate the value of each action because the next-states are unknown. An alternative formulation is to use a neural network with multiple output nodes. Each of the output nodes represent one action j out of the set possible actions, and the value at the output nodes $V(x^{(t)}, a_j^{(t)})$ represents the value of taking a specific action j when in the state $x^{(t)}$ that is input to the network. The output of the network can then be considered a state-action value, or the value of pursuing action j when in state $x^{(t)}$. In a single pass of the network, one can compute all of the state-action values for all actions. The selection of a single action can be just as in the case where the network has a single output node; that is, using an ϵ-greedy action selection procedure perhaps based on the state-action values of all possible actions.

The weight updates for state-action value learning are slightly different. Rather than having a single error term (for a single network output), we have an error vector where each value corresponds to one of the possible actions. Furthermore, only the output node j corresponding to the action that was selected has a non-zero error term, which is:

$$E^{(t)} = r^{(t+1)} + \gamma V\left(x^{(t+1)}, a^{(t+1)}\right) - V\left(x^{(t)}, a^{(t)}\right)$$

This approach is still a temporal difference approach because it is based on a difference from two time steps. However, this approach is technically more of an on-policy version of Q-learning. We use this approach in the mountain car domain and the two truck backer-upper problems, all of which have continuous dynamics.

2.2.3.2 Learning Algorithm Convergence

As previously discussed, the most common representation used with the TD(λ) algorithm is a form of linear function approximator, largely due to their simplicity, but also because temporal difference-based learning algorithms with linear function approximators can be rigorously analyzed. Jaakkola et al. (1995) and Tsitsiklis and Roy (1997) proved that temporal difference learning converges when using a linear function approximator with on-policy learning with some assumptions on the underlying Markov process, the state vectors, and learning rates. On-policy learning refers to when the value function is learned from the states that were actually visited by the agent; an alternative is off-policy learning, in which the value function is learned from states that may be different from those selected and visited by the agent. However, this rigorous analytic work has not been extended to nonlinear function approximators. Rather, counter-examples have shown that when used with nonlinear function approximators, such as neural networks, temporal difference learning can result in suboptimal solutions (Bertsekas and Tsitsiklis 1996) or can diverge (Baird 1995; Boyan and Moore 1995; Gordon 1995; Tsitsiklis and Roy 1997; Papavassiliou and Russell 1999; Fairbanks and Alonso 2012). However, as the references in Sect. 2.2.2 have shown, there are many successful examples of using temporal difference learning with neural networks.

2.2.3.3 Additional Reinforcement Learning Algorithms

Value function learning is only one approach that has been developed for learning how to behave in sequential decision making problems, and the work proposed herein will focus exclusively on value function learning. However, it is important to acknowledge other learning algorithms that are used in this field, though we will only briefly mention some of these methods and the reader is directed to Sutton and Barto (1998), Szepesvári (2010), and Powell and Ma (2011) for a comprehensive review of many algorithms used for reinforcement learning.

Extensions to the basic temporal difference algorithm include gradient-based methods that are based on the Bellman residual error (Baird 1995, 1999; Sutton et al. 2009a), which attempt to stabilize the performance of temporal difference methods, and least-squares temporal difference methods for linear approximators (Bradtke and Barto 1996; Boyan 2002). Most temporal difference algorithms are based on the estimation and learning of state values, however, in some circumstances the addition of using the action selected can aid in learning the underlying value function. These methods, called Q-learning methods, attempt to estimate a value function based

on state-action pairs, rather than just using state information alone (Watkins 1989; Watkins and Dayan 1992). One of the main differences between Q-learning and state value-based methods is that Q-learning is an off-policy learning method, i.e., it can learn the value function from states that were not explicitly visited; state value-based methods like temporal difference methods are on-policy learning methods. A further extension to Q-learning is Sarsa (state-action-reward-state-action), which learns a value function based on the quintuple $(\mathbf{x}^{(t)}, a^{(t)}, r^{(t)}, \mathbf{x}^{(t+1)}, a^{(t+1)})$ and is more suited to control problems (Rummery and Niranjan 1994; Sutton and Barto 1998).

Policy iteration methods are another approach to solving reinforcement learning problems. This approach is based on two separate, but interacting, entities which: (1) evaluates or computes the value (e.g., as in value function or Q-function) of the current policy, and (2) attempts to improve the current policy based on its value. The basic approach to this learning method is called the actor-critic method, where the critic evaluates the current policy and the critic improves the policy (Sutton and Barto 1998).

2.3 Heuristics and Performance Effectors

The inconsistency of success with reinforcement learning has led to the use of heuristic methods that modify the bare bones reinforcement learning algorithms. These methods often attempt to provide additional information during training, whether it is based on more concrete and certain state values, replaying past experiences, or other methods. The varied results of reinforcement learning has also led to the questioning of the reasons behind either their success or failure. While solid evidence regarding the true effectors of performance is rare, many authors speculate on what enabled success or contributed to failure in their or others' works. This section reviews heuristic methods and the factors that are believed to be related to reinforcement learning performance.

2.3.1 Heuristics for Reinforcement Learning

Heuristics are often used to enable or improve learning in terms of ultimate performance or learning efficiency. Many different heuristics have been used in a variety of domains, and while most of them are shown to be effective (possibly due to publication bias), the effectiveness of these approaches across domains is unknown. One training strategy that is a standard approach for training agents in adversarial domains is self-play. In this case, the agent plays for both players and selects actions which maximize next-state values for one player and select those that minimize next-state values for the other player (Tesauro 1995; Wiering et al. 2007; Gatti and Embrechts 2012). Experience replay consists of saving and reusing state sequences from previous episodes to improve the state value function (Lin 1992; Kalyanakrishnan and Stone 2007; van Seijen et al. 2011). While the implementation

by Kalyanakrishnan and Stone (2007) showed that experience replay effectively increase performance, the implementation by Lin (1992) and van Seijen et al. (2011) showed that experience replay increased the speed of learning, though the maximal performance was similar to that when not using experience replay.

Another heuristic, called database games, consists of having the agent merely observe games that have been played by human or computer players and that are stored in a database (Tesauro 1995; Thrun 1995; Patist and Wiering 2004; Mannen and Wiering 2004; Wiering et al. 2007). In this case, the agent does not select actions as in traditional reinforcement learning, but instead the agent learns from predetermined actions. The level of play of the database games can vary, but most often high-level games played by expert human players are used.

A closely related heuristic to learning a problem is that of transfer learning (Taylor and Stone 2009) or relational learning (Torrey 2009). Recall that in reinforcement learning, the agent generally begins with no knowledge about the problem domain. In transfer and relational learning, knowledge that has been learned about one task is utilized to improve the learning process and efficiency in another, related task. A 'related' task can take different forms, some of which include sharing features (Konidaris et al. 2012), altering the allowable actions, altering the reward structure, or generalizing the applicability. This approach is also very similar to that of inductive learning from an artificial intelligence perspective (Michalski 1983), for which there has been some work that leverages this learning approach to develop agents that learn provably optimal solutions (Schmidhuber 2005, 2006).

The use of specific domain or expert knowledge has also be used to improve learning efficacy. These methods exploit domain information (Hoffmann and Freier 1996), modify the representation (Schraudolph et al. 1994), or use an expertly-contrived set of state features or state encoding scheme (Tesauro 1995; Ghory 2004; Konen and Beielstein 2008; Silver et al. 2012). An alternative approach to using an explicit set of state features is to use features that are essentially 'discovered' to be useful based on spectral analysis of the agent's empirical state transition graph (Mahadevan and Maggioni 2007). Domain information has also been exploited by partitioning the state space into a small number of groups, where the states within each group have similar values, but where the groups themselves represent relatively unique scenarios (Wiering 1995). An effect of this approach is that the entire, and potentially discontinuous, state space is partitioned into groups such that each group has a smooth and continuous state subspace. A somewhat related method is to use a hierarchical approach which also essentially results in a partitioning of the state space, but where subgoals are used to achieve a single overarching goal (Bakker and Schmidhuber 2004; Şimşek and Barto 2004).

2.3.1.1 Effectors of Reinforcement Learning Performance

Despite the widely varying results of reinforcement learning in a variety of domains and circumstances, there is relatively little work explicitly investigating exactly what

affects the performance of reinforcement learning with respect to domain characteristics, learning algorithms, and representations, as well as interactions between these components. Early works that investigated performance effectors were largely focused on algorithm parameters settings in specific scenarios, whereas in recent years the reinforcement learning community has begun to advocate studies that have broader implications in reinforcement learning (Whiteson et al. 2011). The examples cited here use different learning algorithms and representations and the domains have different characteristics, and thus the results and implications of parameter effects may not be completely comparable. It is therefore difficult to draw generalizable conclusions regarding the effects of learning algorithms and their parameters, representations, and domain characteristics. This section is not meant to be comprehensive, yet it provides an overview of some of what has an effect on reinforcement learning performance. Additionally, we focus more on studies that investigate, or perhaps speculate, how any of the components of reinforcement learning affect performance, and we do not include those that perform relatively simple comparative analyses.

With respect to specific parameter settings, there are both consistencies and inconsistencies. In numerous applications, using a temporal discount factor $\lambda \approx 0.7$ with the TD(λ) algorithm has resulted in good performance in a variety of board games of varying difficulties (Tesauro 1992; Wiering 1995, 2010; Wiering et al. 2007; Gatti et al. 2011a). However, the setting and effects of the state-discount parameter γ are somewhat conflicting. When using the TD(λ) algorithm, Ghory (2004) found that γ has little effect in the game of Tic-tac-toe, and a negligible effect in the game of Connect 4. Gatti et al. (2013) used the TD(λ) algorithm with a neural network to learn the mountain car domain and found that γ has a significant effect on whether or not the network converges ($\gamma \approx 0.97$ enabled consistent convergence), on the convergence speed, as well as on the performance when the network did converge. On the other hand, Thrun and Schwartz (1993) proved that, under some circumstances, learning fails using Q-learning when $\gamma \geq 0.98$, and empirical testing in a simple robot domain showed that performance varied considerable for a wide range of γ as well as for the type of representation used. These authors also acknowledge that there are limitations to theoretical analysis and that the empirical results were not completely consistent with the theoretical analysis. In the game of chess, Thrun (1995) found that when using TD-learning with an explanation-based neural network, setting $\gamma = 1$ led to essentially no learning.

A number of domain qualities have been suggested to be beneficial to learning a domain as well. Board or value function smoothness (Ghory 2004) has been suggested to be beneficial in a number of domains, including chess (Thrun 1995) and backgammon (Tesauro 1995; Baxter et al. 1998a). Smoothness is dependent on both the domain as well as the state encoding scheme; carefully hand-crafted state features that produce a smooth value function may enable more efficient and effective learning than native encoding schemes. Thrun (1995) also suggests that the sampling nature (i.e., frequency) of states plays an important role in learning, as well as the information content of each state, which is also dependent on the state encoding. The sampling nature of the state is closely related to the exploration/exploitation

trade-off for action selection. Action exploration has been found to be very effective in improving the speed and quality of learning and is a standard training action selection policy. However, the use of a pure exploitative action selection procedure by Tesauro (1995) in backgammon also resulted in excellent performance, which may have been due to the stochasticity of the game itself, thus allowing for implicit state space exploration (Kaelbling et al. 1996).

Training for longer durations, or playing more games, has resulted in mixed results. Tesauro (1995) trained a neural network for months (Kaelbling et al. 1996), which resulted in world class performance. However, Wiering et al. (2007) found that longer training did not result in marked improvements in playing performance for their implementation of backgammon. Similarly, Gatti et al. (2011a) found that training for 100,000 games did not result in superior performance than when training for 10,000 games. The reason why the backgammon implementation of Tesauro (1995) kept increasing performance are unknown and are somewhat puzzling because temporal difference algorithms are known to 'unlearn'; that is, their performance does not necessarily monotonically increase with more training. Experience replay stores and trains on previously played games, and this method has been found to be effective in increasing performance in keep-away (Kalyanakrishnan and Stone 2007), though it did not increase maximal performance in the works by Lin (1992) for a Gridworld-type problem and by van Seijen et al. (2011) for the mountain car problem.

The training opponent has also been found to have an impact playing performance in adversarial domains. Tesauro (1995) used a self-play training scheme that eventually became an excellent player, and this approach has been successfully used in many other games (Wiering et al. 2007; Wiering 2010; Gatti et al. 2011b). As mentioned, an alternative training strategy uses database games where the agent observes games that were previously played by high level opponents. While this seems beneficial, the resulting performance of this method was the worst out of a number of different training methods in an implementation of backgammon by Wiering (2010), and it is speculated that this is due to the fact that the agent cannot test and explore actions that may potentially be better. In similar work, Wiering et al. (2007) found that performance differed very little after training against an expert player versus using self-play. Schraudolph et al. (1994) used three openly-available computer programs with differing levels of expertise to train individual neural networks to play Go, and found that the final playing ability of each network was quite different.

Generalized domains have also been used to gain an understanding of the behavior of algorithms under different domain characteristics. Bhatnagar et al. (2009) used two different Garnet problems to assess the performance of actor-critic algorithms in domains with different numbers of state, actions, and branching factors. This work found that it was easier to find parameter settings for one particular actor-critic algorithm than others, and that there are considerable differences in the convergence speed of the algorithms tested. The authors note that their parameter study was small and simplistic and the results of their study is merely suggestive of parameter and domain effects. Kalyanakrishnan and Stone (2009, 2011) compare the performance

of different learning algorithms on generalized Gridworld problems that are parameterized by the size of the domain, state transition stochasticity, function approximator coverage, and state observability. A series of basic parameter studies found that domain characteristics affect which type of learning algorithm performs best, and that there may be interactions between domain characteristics and algorithm parameters, though these results were not statistically analyzed. Whiteson et al. (2011) uses generalized benchmark problems (mountain car, acrobot, and puddle world) to evaluate state space coverage methods for function approximators, and this work found that adaptive tile coding could perform well over all of the test domains, whereas the performance of a general tile coding scheme was worse and more variable.

References

Albus, J. S. (1975). A new approach to manipulator control: The cerebellar model articulation controller (CMAC). *Journal of Dynamic Systems, Measurement, and Control*, 97(3), 220–227.

Aldous, D. (1983). Random walks on finite groups and rapidly mixing Markov chains. In *Seminar on Probability XVII, Lecture Notes in Mathematics Volume 986* (pp. 243–297). Berlin: Springer.

Anderson, C. W. (1987). Strategy learning with multilayer connectionist representations. In Langley, P. (Ed.), *Proceedings of the 4th International Workshop on Machine Learning, Irvine, CA, 22–25 June* (pp. 103–114). San Mateo, CA: Morgan Kaufmann.

Atkeson, C. G. & Santamaría, J. C. (1997). A comparison of direct and model-based reinforcement learning. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Albequerque, NM, 20–25 April* (Vol. 4, pp. 3557–3564). doi: 10.1109/ROBOT.1997.606886

Atkeson, C. G., Moore, A. W., & Schaal, S. (1997). Locally weighted learning. *Artificial Intelligence Review*, 11(1–5), 11–73.

Archibald, T. W., McKinnon, K. I. M., & Thomas, L. C. (1995). On the generation of Markov decision processes. *Journal of the Operational Research Society*, 46(3), 354–361.

Awate, Y. P. (2009). Policy-gradient based actor-critic algorithms. In *Proceedings of the Global Congress on Intelligent Systems (GCIS), Xiamen, China, 19–21 May* (pp. 505–509). doi: 10.1109/GCIS.2009.372

Bagnell, J. A. & Schneider, J. G. (2001). Autonomous helicopter control using reinforcement learning policy search methods. In *Proceedings of the International Conference on Robotics and Automation, Seoul, Korea, 21–26 May* (Vol. 2, pp. 1615–1620). doi: 10.1109/ROBOT.2001.932842

Baird, L. (1995). Residual algorithms: Reinforcement learning with function approximation. In Prieditis, A. and Russell, S. (Eds.) *Proceedings of the 12th International Conference on Machine Learning (ICML), Tahoe City, CA, 9–12 July* (pp. 30–37). San Francisco, CA: Morgan Kaufmann.

Baird, L. C. (1999). *Reinforcement learning through gradient descent*. Unpublished PhD dissertation, Carnegie Mellon University, Pittsburgh, PA.

Bakker, B. (2001). *Reinforcement learning with LSTM in non-Markovian tasks with longterm dependencies* (Technical Report, Department of Psychology, Leiden University). Retrieved from http://staff.science.uva.nl/~ bram/RLLSTM_ TR.pdf.

Bakker, B. (2007). Reinforcement learning by backpropagation through an LSTM model/critic. In *IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL), Honolulu, HI, 1–5 April* (pp. 127–134). doi: 10.1109/ADPRL.2007.368179

Bakker, B. & Schmidhuber, J. (2004). Hierarchical reinforcement learning based on subgoal discovery and subpolicy specialization. In Groen, F., Amato, N., Bonarini, A., Yoshida, E., &

Kröse, B. (Eds.), *Proceedings of the 8th Conference on Intelligent Autonomous Systems (IAS-8), Amsterdam, The Netherlands, 10–13 March* (pp. 438–445). Amsterdam, Netherlands: IOS Press.

Bakker, B., Linaker, F., & Schmidhuber, J. (2002). Reinforcement learning in partially observable mobile robot domains using unsupervised event extraction. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2002), EPFL, Switzerland, 30 September–4 October* (Vol. 1, pp. 938–943). doi: 10.1109/IRDS.2002.1041511

Barto, A. G. (1990). Connectionist learning for control: An overview. In Miller, W. T., Sutton, R. S., and Werbos, P. J. (Eds.), *Neural Networks for Control* (pp. 5–58). Cambridge, MA: MIT Press.

Barto, A. G., Sutton, R. S., & Anderson, C. (1983). Neuron-like adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics (SMC)*, 13(5), 834–846.

Baxter, J. & Bartlett, P. L. (2000). Reinforcement learning in POMDP's via direct gradient ascent. In *Proceedings of the 17th International Conference on Machine Learning (ICML)*, Stanford University, Stanford, CA, 29 June–2 July (pp. 41–48). San Francisco, CA: Morgan Kaufmann.

Baxter, J., Tridgell, A., & Weaver, L. (1998a). KnightCap: A chess program that learns by combining TD(λ) with minimax search. In *Proceedings of the 15th International Conference on Machine Learning, Madison, WI, 24–27 July* (pp. 28–36). San Francisco, CA: Morgan Kaufmann.

Baxter, J., Tridgell, A., & Weaver, L. (1998b). TDLeaf(λ): Combining temporal difference learning with game-tree search. *Australian Journal of Intelligent Information Processing Systems*, 5(1), 39–43.

Bertsekas, D. P. (1987). *Dynamic Programming: Deterministic and Stochastic Models*. Englewood Cliffs, NJ: Prentice-Hall.

Bertsekas, D. P. & Tsitsiklis, J. N. (1996). *Neuro-dynamic Programming*. Belmont, MA: Athena Scientific.

Bhatnagar, S., Sutton, R., Ghavamzadeh, M., & Lee, M. (2009). Natural actor critic algorithms. *Automatica*, 45(11), 2471–2482.

Binkley, K. J., Seehart, K., & Hagiwara, M. (2007). A study of artificial neural network architectures for Othello evaluation functions. *Information and Media Technologies*, 2(4), 1129–1139.

Bonarini, A., Lazaric, A., & Restelli, M. (2007). Reinforcement learning in complex environments through multiple adaptive partitions. In *AI*IA 2007: Artificial Intelligence and Human-Oriented Computing, Proceedings of the 10th Congress of the Italian Association for Artificial Intelligence, Rome, Italy, 10–13 September* (pp. 531–542). doi: 10.1007/978-3-540-74782-6_46

Boyan, J. A. (2002). Technical update: Least-squares temporal difference learning. *Machine Learning*, 49(2–3), 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*, (pp. 369–376). Cambridge, MA: MIT Press.

Bradtke, S. J. & Barto, A. G. (1996). Linear least-squares algorithms for temporal difference learning. *Machine Learning*, 22(1–3), 33–57.

Castro, D. D. & Mannor, S. (2010). Adaptive bases for reinforcement learning. In *Proceedings of the 2010 European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD), Barcelona, Spain, 20–24 September* (pp. 312–327). doi: 10.1007/978-3-642-15880-3_26

Chapman, D. & Kaelbling, L. P. (1991). Input generalization in delayed reinforcement learning: An algorithm and performance comparisons. In *Proceedings of 12th International Joint Conference on Artificial Intelligence (IJCAI), Sydney, Australia, 24–30 August* (Vol. 2, pp. 726–731). San Francisco, CA: Morgan Kaufmann.

Coulom, R. (2002a). Feedforward neural networks in reinforcement learning applied to high-dimensional motor control. In *Proceedings of the 13th International Conference on Algorithmic Learning Theory (ALT 2002), Lübeck, Germany, 24–26 November* (pp. 402–413). doi: 10.1007/3-540-36169-3_32

Coulom, R. (2002b). *Reinforcement learning using neural networks, with applications to motor control*. Unpublished PhD dissertation, National Polytechnic Institute of Grenoble, Grenoble, France.

Dann, C., Neumann, G., & Peters, J. (2014). Policy evaluation with temporal differences: A survey and comparison. *Journal of Machine Learning Research*, 15(1), 809–883.

Dayan, P. (1993). Improving generalization for temporal difference learning: The successor representation. *Neural Computation*, 5(4), 613–624.

Dayan, P. & Niv, Y. (2008). Reinforcement learning: The good, the bad and the ugly. *Current Opinion in Neuroscience*, 18(2), 185–196.

Dietterich, T. G. (2000). Ensemble methods in machine learning. In *Proceedings of the 1st International Workshop on Multiple Classifier Systems (MCS), Cagliari, Italy, 21–23 June* (pp. 1–15). doi: 10.1007/3-540-45014-9_1

Doya, K. (1996). Temporal difference learning in continuous time and space. In Touretzky, D. S., Mozer, M. C., & Hasselmo, M. E. (Eds.), *Advances in Neural Information Processing Systems 8* (pp. 1073–1079). Cambridge, MA: MIT Press.

Doya, K. (2000). Reinforcement learning in continuous time and space. *Neural Computation*, 12(1), 219–245.

Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, 14(2), 179–211.

Fairbanks, M. & Alonso, E. (2012). The divergence of reinforcement learning algorithms with value-iteration and function approximation. In *Proceedings of the 2012 International Joint Conference on Neural Networks (IJCNN), Brisbane, Queensland, Australia, 10–15 June* (pp. 1–8). doi: 10.1109/IJCNN.2012.6252792

Främling, K. (2008). Light-weight reinforcement learning with function approximation for real-life control tasks. In Filipe, J., Andrade-Cetto, J., & Ferrier, J.-L. (Eds.), *Proceedings of the 5th International Conference on Informatics in Control, Automation and Robotics, Intelligent Control Systems and Optimization (ICINCO-ICSO), Funchal, Madeira, Portugal, 11–15 May* (pp. 127–134). INSTICC Press.

Gabel, T. & Riedmiller, M. (2007). On a successful application of multi-agent reinforcement learning to operations research benchmarks. In *Proceedings of the 2007 IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL 2007), Honolulu, HI, 1–5 April* (pp. 69–75). doi: 10.1109/ADPRL.2007.368171

Gabel, T., Lutz, C., & Riedmiller, M. (2011). Improved neural fitted Q iteration applied to a novel computer gaming and learning benchmark. In *Proceedings of the 2011 IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL 2011), Paris, France, 11–15 April* (pp. 279–286). doi: 10.1109/ADPRL.2011.5967361

Galichet, N., Sebag, M., & Teytaud, O. (2013). Exploration vs. exploitation vs safety: Risk-aware multi-armed bandits. In *Proceedings of the Asian Conference on Machine Learning (ACML 2013), Canberra, ACT, Australia, 13–15 November* (pp. 245–260). Journal of Machine Learning Research (JMLR): Workshop and Conference Proceedings.

Gatti, C. J. & Embrechts, M. J. (2012). Reinforcement learning with neural networks: Tricks of the trade. In Georgieva, P., Mihayolva, L., & Jain, L. (Eds.), *Advances in Intelligent Signal Processing and Data Mining* (pp. 275–310). New York, NY: Springer-Verlag.

Gatti, C. J., Embrechts, M. J., & Linton, J. D. (2011a). Parameter settings of reinforcement learning for the game of Chung Toi. In *Proceedings of the 2011 IEEE International Conference on Systems, Man, and Cybernetics (SMC 2011), Anchorage, AK, 9–12 October* (pp. 3530–3535). doi: 10.1109/ICSMC.2011.6084216

Gatti, C. J., Linton, J. D., & Embrechts, M. J. (2011b). A brief tutorial on reinforcement learning: The game of Chung Toi. In *Proceedings of the 19th European Symposium on Articial Neural Networks, Computational Intelligence and Machine Learning (ESANN), Bruges, Belgium, 27–29 April* (pp. 129–134). Bruges, Belgium: ESANN.

Gatti, C. J., Embrechts, M. J., & Linton, J. D. (2013). An empirical analysis of reinforcement learning using design of experiments. In *Proceedings of the 21st European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN), Bruges, Belgium, 24–26 April* (pp. 221–226). Bruges, Belgium: ESANN.

Gers, F. (2001). *Long short-term memory in recurrent neural networks*. Unpublished PhD dissertation, École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland.

Ghory, I. (2004). *Reinforcement learning in board games* (Technical Report CSTR-04-004, Department of Computer Science, University of Bristol). Retrieved from http://www.cs.bris.ac.uk/Publications/Papers/2000100.pdf.

Gordon, G. J. (1995). Stable function approximation in dynamic programming. In *Proceedings of the 12th International Conference on Machine Learning (ICML), Tahoe City, CA, 9–12 July* (pp. 261–268). San Francisco, CA: Morgan Kaufmann.

Gordon, G. J. (2001). Reinforcement learning with function approximation converges to a region. In *Advances in Neural Information Processing Systems 13* (pp. 1040–1046). Cambridge, MA: MIT Press.

Gorse, D. (2011). Application of stochastic recurrent reinforcement learning to index trading. In *European Symposium on Artificial Neural Networks, Computational Intelligence, and Machine Learning (ESANN), Bruges, Belgium, 27–29 April* (pp. 123–128). Bruges, Belgium: ESANN.

Gosavi, A., Bandla, N., & Das, T. K. (2002). A reinforcement learning approach to a single leg airline revenue management problem with multiple fare classes and overbooking. *IIE Transactions*, 34(9), 729–742.

Grüning, A. (2007). Elman backpropagation as reinforcement for simple recurrent networks. *Neural Computation*, 19(11), 3108–3131.

Günther, M. (2008). *Automatic feature construction for general game playing*. Unpublished masters thesis, Dresden University of Technology, Dresden, Germany.

Hafner, R. & Riedmiller, M. (2011). Reinforcement learning in feedback control. *Machine Learning*, 84(1–2), 137–169.

Hans, A. & Udluft, S. (2010). Ensembles of neural networks for robust reinforcement learning. In *Proceedings of the 9th International Conference on Machine Learning and Applications (ICMLA), Washington D.C., 12–14 December* (pp. 401–406). doi: 10.1109/ICMLA.2010.66

Hans, A. & Udluft, S. (2011). Ensemble usage for more reliable policy identification in reinforcement learning. In *European Symposium on Artificial Neural Networks, Computational Intelligence, and Machine Learning (ESANN), Bruges, Belgium, 27–29 April* (pp. 165–170). Bruges, Belgium: ESANN.

Hochreiter, S. & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8), 1735–1780.

Hoffmann, A. & Freier, B. (1996). On integrating domain knowledge into reinforcement learning. In *International Conference on Neural Information Processing (ICONIP), Hong Kong, China, 24–27 September* (pp. 954–959). Singapore: Springer-Verlag.

Igel, C. (2003). Neuroevolution for reinforcement learning using evolution strategies. In *Proceedings from the 2003 Conference on Evolutionary Computing (CEC), Canberra, Australia, 8–12 December* (Vol. 4, pp. 2588–2595). doi: 10.1109/CEC.2003.1299414

Jaakkola, T., Singh, S. P., & Jordan, M. I. (1995). Reinforcement learning algorithm for partially observable Markov decision problem. In *Advances in Neural Information Processing Systems 7* (pp. 345–352). Cambridge, MA: MIT Press.

Jaakkola, T., Jordan, M. I., & Singh, S. P. (2003). On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, 6(6), 1185–1201.

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4, 237–285.

Kalyanakrishnan, S. & Stone, P. (2007). Batch reinforcement learning in a complex domain. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS07), Honolulu, HI, 14–18 May* (pp. 650–657). doi: 10.1145/1329125.1329241

Kalyanakrishnan, S. & Stone, P. (2009). An empirical analysis of value function-based and policy search reinforcement learning. In *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS '09), Budapest, Hungary, 10–15 May* (Vol. 2, pp. 749–756). Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Kalyanakrishnan, S. & Stone, P. (2011). Characterizing reinforcement learning methods through parameterized learning problems. *Machine Learning*, 84(1–2), 205–247.

Kappen, H. J. (2007). An introduction to stochastic control theory, path integrals and reinforcement learning. In Marro, J., Garrido, P. L., & Torres, J. J. (Eds.), *Cooperative Behavior in Neural Systems, American Institute of Physics Conference Series, Granada, Spain, 11–15 September* (Vol. 887, pp. 149–181). American Institute of Physics.

Karnin, Z., Koren, T., & Somekh, O. (2013). Almost optimal exploration in multi-armed bandits. In *Proceedings of the 30th International Conference on Machine Learning (ICML 2013), Atlanta, GA, 16–21 June* (Vol. 28, pp. 1238–1246). JMLR Proceedings.

Kohl, N. and Stone, P. (2004). Policy gradient reinforcement learning for fast quadrupedal locomotion. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), New Orleans, LA, 26 April 26–1 May* (pp. 2619–2624). doi: 10.1109/ROBOT.2004.1307456

Konen, W. & Beielstein, T. B. (2008). Reinforcement learning: Insights from interesting failures in parameter selection. In *Parallel Problem Solving from Nature—PPSN X, Proceedings of the 10th International Conference on Parallel Problem Solving from Nature, Dortmund, Germany, 13–17 September* (pp. 478–487). doi: 10.1007/978-3-540-87700-4_48

Konen, W. & Beielstein, T. B. (2009). Reinforcement learning for games: Failures and successes. In *Proceedings of the 11th Genetic and Evolutionary Computation Conference (GECCO), Montreal, Canada, 8–12 July* (pp. 2641–2648). doi: 10.1145/1570256.1570375

Konidaris, G., Osentoski, S., & Thomas, P. S. (2011). Value function approximation in reinforcement learning using the Fourier basis. In Burgard, W. & Roth, D. (Eds.), *Proceedings of the 25th Conference on Artificial Intelligence (AAAI 2011), San Francisco, CA, 7–11 August* (pp. 380–385). AAAI.

Konidaris, G. D., Scheidwasser, I., & Barto, A. G. (2012). Transfer in reinforcement learning via shared features. *Journal of Machine Learning Research*, 13(May), 1333–1371.

Kretchmar, R. M. & Anderson, C. W. (1997). Comparison of CMACs and radial basis functions for local function approximation in reinforcement learning. In *International Conference on Neural Networks, Houston, TX, 9–12 June* (Vol. 2, pp. 834–837). doi: 10.1109/ICNN.1997.616132

Kwok, C. & Fox, D. (2004). Reinforcement learning for sensing strategies. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS 2004), Sendai, Japan, 28 September–2 October* (Vol. 4, pp. 3158–3163). doi: 10.1109/IROS.2004.1389903

Lange, S., Gabel, T., & Riedmiller, M. (2012). Batch reinforcement learning. In Wiering, M. & van Otterlo, M. (Eds.), *Reinforcement Learning: State-of-the-Art* (pp. 45–73). New York, NY: Springer.

Langley, P. (1988). Machine learning as an experimental science. *Machine Learning*, 3(1), 5–8.

Lazaric, A. (2008). *Knowledge transfer in reinforcement learning*. Unpublished PhD dissertation, Politecnico di Milano, Milano, Italy.

Lee, J. W. (2001). Stock price prediction using reinforcement learning. In *Proceedings of the IEEE International Symposium on Industrial Electronics, Pusan, South Korea, 12–16 June* (Vol. 1, pp. 690–695). doi: 10.1109/ISIE.2001.931880

O, J., Lee, J., Lee, J. W., & Zhang, B.-T. (2006). Adaptive stock trading and dynamic asset allocation using reinforcement learning. *Information Sciences*, 176(15), 2121–2147.

Li, Y. & Schuurmans, D. (2008). Policy iteration for learning an exercise policy for American options. In Girgin, S., Loth, M., Munos, R., Preux, P., & Ryabko, D., editors, *Recent Advances in Reinforcement Learning, Proceedings of the 8th European Workshop on Recent Advances in Reinforcement Learning (EWRL 2008), Villeneuve d'Ascq, France, June 30–July 3* (pp. 165–178). doi: 10.1007/978-3-540-89722-4_13

Li, Y., Szepesvari, C., & Schuurmans, D. (2009). Learning exercise policies for American options. In Dyk, D. V. & Welling, M. (Eds.), *Proceedings of the 12th International Conference on Artificial Intelligence and Statistics (AISTATS-09), Clearwater Beach, FL, 16–18 April* (Vol. 5, pp. 352–359). JMLR: Workshop and Conference Proceedings.

Lin, L.-J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8(3–4), 293–321.

Littman, M. L. (2001). Value-function reinforcement learning in Markov games. *Journal of Cognitive Systems Research*, 2(1), 55–66.

Loone, S. M. & Irwin, G. (2001). Improving neural network training solutions using regularisation. *Neurocomputing*, 37(1–4), 71–90.

Mahadevan, S. & Maggioni, M. (2005). Value function approximation with diffusion wavelets and Laplacian eigenfunctions. In *Advances in Neural Information Processing Systems 18*. Cambridge, MA: MIT Press.

Mahadevan, S. & Maggioni, M. (2007). Proto-value functions: A Laplacian framework for learning representation and control in Markov decision processes. *Journal of Machine Learning Research*, 8, 2169–2231.

Mahadevan, S. & Theocharous, G. (1998). Optimizing production manufacturing using reinforcement learning. In Cook, D. J. (Ed.) *Proceedings of the 11th International Florida Artificial Intelligence Research Society Conference, Sanibel Island, Florida, 18–20 May* (pp. 372–377). AAAI Press.

Maia, T. V. (2009). Reinforcement learning, conditioning, and the brain: Successes and challenges. *Cognitive, Affective, & Behavioral Neuroscience*, 9(4), 343–364.

Makino, T. (2009). Proto-predictive representation of states with simple recurrent temporal-difference networks. In *Proceedings of the 26th Annual International Conference on Machine Learning (ICML), Montreal, Canada, 14–18 June* (pp. 697–704). doi: 10.1145/1553374.1553464

Mannen, H. & Wiering, M. (2004). Learning to play chess using TD(λ)-learning with database games. In Nowe, A., Lenaerts, T., & Steenhout, K. (Eds.), *Proceedings of the 13th Belgian-Dutch Conference on Machine Learning, Brussels, Belgium, 8–9 January* (pp. 72–79). Retrieved from http://www.ai.rug.nl/ mwiering/ group/articles/learning-chess.pdf

Menache, I., Mannor, S., & Shimkin, N. (2005). Basis function adaptation in temporal difference reinforcement learning. *Annals of Operations Research*, 134(1), 215–238.

Michalski, R. S. (1983). A theory and methodology of inductive learning. *Artificial Intelligence*, 20(2), 111–161.

Michie, D. & Chambers, R. A. (1968). BOXES: An experiment in adaptive control. In Dale, E. & Michie, D. (Eds.), *Machine Intelligence* (pp. 137–152). Edinburgh, Scotland: Oliver and Boyd.

Mitchell, T. M. & Thrun, S. B. (1992). Explanation-based neural network learning for robot control. In *Advances in Neural Information Processing Systems 5* (pp. 287–294). San Francisco, CA: Morgan Kaufmann.

Montazeri, H., Moradi, S., & Safabakhsh, R. (2011). Continuous state/action reinforcement learning: A growing self-organizing map approach. *Neurocomputing*, 74(7), 1069–1082.

Moody, J. & Saffell, M. (2001). Learning to trade vis direct reinforcement learning. *IEEE Transactions on Neural Networks*, 12(4), 875–889.

Moody, J. & Tresp, V. (1994). A trivial but fast reinforcement controller. *Neural Computation*, 6.

Moody, J., Wu, L., Liao, Y., & Saffell, M. (1998). Performance functions and reinforcement learning for trading systems and portfolios. *Journal of Forecasting*, 17(5–6), 441–470.

Moore, A. W. (1990). *Efficient memory-based learning for robot control*. Unpublished PhD dissertation, University of Cambridge, Cambridge, United Kingdom.

Moore, B. L., Pyeatt, L. D., Kulkarni, V., Panousis, P., Padrez, K., & Doufas, A. G. (2014). Reinforcement learning for closed-loop Propofol anesthesia: A study in human volunteers. *Journal of Machine Learning Research*, 15(Feb), 655–696.

Nevmyvaka, Y., Feng, Y., & Kearns, M. (2006). Reinforcement learning for optimized trade execution. In Cohen, W. W. and Moore, A. (Eds.), *Proceedings of the 23rd International Conference on Machine Learning (ICML), Pittsburgh, PA, 25–29 June* (pp. 673–680). New York, NY: ACM.

Ng, A. Y., Coates, A., Diel, M., Ganapathi, V., Schulte, J., Tse, B., Berger, E. & Liang, E. (2004). Autonomous inverted helicopter flight via reinforcement learning. In *International Symposium on Experimental Robotics (ISER-2004), Singapore, 18–21 June* (pp. 363–372). Cambridge, MA: MIT Press.

Nissen, S. (2007). *Large scale reinforcement learning using Q-Sarsa(λ) and cascading neural networks*. Unpublished masters thesis, Department of Computer Science, University of Copenhagen, København, Denmark.

Niv, Y. (2009). Reinforcement learning in the brain. *Journal of Mathematical Psychology*, 53(3), 139–154.

Ollington, R. B., Vamplew, P. H., & Swanson, J. (2009). Incorporating expert advice into reinforcement learning using constructive neural networks. In Franco, L., Elizondo, D. A., & Jerez, J. M. (Eds.), *Constructive Neural Networks* (pp. 207–224). Berlin: Springer.

Orr, M. J. L. (1996). *Introduction to radial basis function networks* (Technical report, Centre For Cognitive Science, University of Edinburgh). Retrieved from http://www.cc.gatech.edu/ isbell/tutorials/rbf-intro.pdf.

Osana, Y. (2011). Reinforcement learning using Kohonen feature map probabilistic associative memory based on weights distribution. In Mellouk, A. (Ed.), *Advances in Reinforcement Learning* (pp. 121–136). InTech.

Osentoski, S. (2009). *Action-based representation discovery in Markov decision processes*. Unpublished PhD dissertation, University of Massachusetts, Amherst, MA.

Papahristou, N. & Refanidis, I. (2011). Training neural networks to play backgammon variants using reinforcement learning. In *Applications of Evolutionary Computation, Proceedings of the 11th International Conference on Applications of Evolutionary Computation, Torino, Italy, 27–29 April* (pp. 113–122). Berlin: Springer-Verlag.

Papavassiliou, V. A. & Russell, S. (1999). Convergence of reinforcement learning with general function approximators. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI), Stockholm, Sweden, 31 July–6 August* (Vol. 2, pp. 748–755). San Francisco, CA: Morgan Kaufmann.

Papierok, S., Noglik, A., & Pauli, J. (2008). Application of reinforcement learning in a real environment using an RBF network. In *1st International Workshop on Evolutionary and Reinforcement Learning for Autonomous Robot Systems (ERLARS), Patras, Greece, 22 July* (pp. 17–22). Retrieved from http://www.is.uni-due.de/fileadmin/literatur/publikation/papierok08erlars.pdf

Patist, J. P. & Wiering, M. (2004). Learning to play draughts using temporal difference learning with neural networks and databases. In *Proceedings of the 13th Belgian-Dutch Conference on Machine Learning, Brussels, Belgium, 8–9 January* (pp. 87–94). doi: 10.1007/978-3-540-88190-2_13

Peters, J. & Schaal, S. (2006). Policy gradient methods for robotics. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Beijing, China, 9–15 October* (pp. 2219–2225). doi: 10.1109/IROS.2006.282564

Peters, J. & Schaal, S. (2009). Reinforcement learning of motor skills with policy gradients. *Neural Networks*, 21(4), 682–697.

Pollack, J. B. & Blair, A. D. (1996). Why did TD-Gammon work? In Mozer, M. C., Jordan, M. I., & Petsche, T. (Eds.), *Advances in Neural Information Processing Systems 9*. Cambridge, MA: MIT Press.

Pontrandolfo, P., Gosavi, A., Okogbaa, O. G., & Das, T. K. (2002). Global supply chain management: A reinforcement learning approach. *International Journal of Production Research*, 40(6), 1299–1317.

Powell, W. B. (2007). *Approximate Dynamic Programming: Solving the Curse of Dimensionality*. New York, NY: John Wiley & Sons.

Powell, W. B. (2008). What you should know about approximate dynamic programming. *Naval Research Logistics*, 56(3), 239–249.

Powell, W. B. & Ma, J. (2011). A review of stochastic algorithms with continuous value function approximation and some new approximate policy iteration algorithms for multidimensional continuous applications. *Journal of Control Theory and Applications*, 9(3), 336–352.

Proper, S. & Tadepalli, P. (2006). Scaling model-based average-reward reinforcement learning for product delivery. In *Machine Learning: European Conference on Machine Learning (ECML 2006), Berlin, Germany, 18–22 September* (pp. 735–742). doi: 10.1007/11871842_74

Rescorla, R. A. & Wagner, A. R. (1972). A theory of Pavlovian conditioning: Variations in the effectiveness of reinforcement and nonreinforcement. In Black, A. H. & Prokasy, W. F. (Eds.), *Classical Conditioning II: Current research and theory* (pp. 64–99). New York, NY: Appleton-Century-Crofts.

Riedmiller, M. (2005). Neural fitted Q iteration—First experiences with a data efficient neural reinforcement learning method. In Gama, J., Camacho, R., Brazdil, P. B., Jorge, A. M., & Torgo, L. (Eds.), *Proceedings of the 16th European Conference on Machine Learning (ECML 2005), Porto, Portugal, 3–7 October* (pp. 317–328). doi: 10.1007/11564096_32

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning internal representation by error propagation. In Rumelhart, D. E. & McClelland, J. L. (Eds.), *Parallel Distributed Processing: Exploration in the Microstructure of Cognition*. Cambridge, MA: MIT Press.

Rummery, G. A. & Niranjan, M. (1994). *On-line Q-learning using connectionist systems* (Technical Report CUED/F-INFENG/TR 166, Engineering Department, Cambridge University). Retrieved from http://mi.eng.cam.ac.uk/reports/svr-ftp/auto-pdf/rummery_tr166.pdf

Runarsson, T. P. & Lucas, S. M. (2005). Co-evolution versus self-play temporal difference learning for acquiring position evaluation in small-board Go. *IEEE Transactions on Evolutionary Computing*, 9(6), 628–640.

Schaeffer, J., Hlynka, M., & Jussila, V. (2001). Temporal difference learning applied to a high-performance game-playing program. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI), Seattle, WA, 4–10 August* (Vol. 1, pp. 529–534). San Francisco, CA: Morgan Kaufmann.

Schmidhuber, J. (2005). Completely self-referential optimal reinforcement learners. In *Proceedings of the International Conference on Artificial Neural Networks (ICANN), Warsaw, Poland, 11–15 September*, volume 3697 of *Lecture Notes in Computer Science* (pp. 223–233). Berlin: Springer.

Schmidhuber, J. (2006). G odel machines: Fully self-referential optimal universal self-improvers. In Goertzel, B. & Pennachin, C. (Eds.), *Artificial General Intelligence* (pp. 199–226). doi: 10.1007/11550907_36

Schraudolph, N. N., Dayan, P., & Sejnowski, T. J. (1994). Temporal difference learning of position evaluation in the game of Go. In Cowan, J. D. & Alspector, G. T. J. (Eds.), *Advances in Neural Information Processing Systems 6*. San Francisco, CA: Morgan Kaufmann.

Silver, D., Sutton, R. S., & Müller, M. (2012). Temporal-difference search in computer Go. *Machine Learning*, 87(2), 183–219.

Şimşek, O. & Barto, A. G. (2004). Using relative novelty to identify useful temporal abstractions in reinforcement learning. In *Proceedings of the 21st International Conference on Machine Learning, Banff, Alberta, Canada, 4–8 July* (pp. 751–758). doi: 10.1145/1015330.1015353

Singh, S. P., Jaakkola, T., & Jordan, M. I. (1994). Learning without state-estimation in partially observable Markovian decision processes. In *Proceedings of the 11th International Conference on Machine Learning (ICML), New Brunswick, NJ, 10–13 July* (pp. 284–292). San Francisco, CA: Morgan Kauffman.

Singh, S. P., Jaakkola, T., & Jordan, M. I. (1995). Reinforcement learning with soft state aggregation. In *Advances in Neural Information Processing Systems 7* (pp. 361–368). Cambridge, MA: MIT Press.

Singh, S. P. & Sutton, R. S. (1996). Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22(1–3), 123–158.

Skelly, M. M. (2004). *Hierarchical reinforcement learning with function approximation for adaptive control*. Unpublished PhD dissertation, Case Western Reserve University, Cleveland, OH.

Skoulakis, I. & Lagoudakis, M. (2012). Efficient reinforcement learning in adversarial games. In *Proceedings of the 24th IEEE International Conference on Tools with Artificial Intelligence (ICTAI), Athens, Greece, 7–9 November* (pp. 704–711). doi: 10.1109/ICTAI.2012.100

Smart, W. D. (2002). *Making reinforcement learning work on real robots*. Unpublished PhD dissertation, Brown University, Providence, RI.

Smart, W. D. & Kaelbling, L. P. (2002). Effective reinforcement learning for mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Washington, D.C., 11–15 May* (Vol. 4, pp. 3404–3410). doi: 10.1109/ROBOT.2002.1014237

Smith, A. J. (2002). Applications of the self-organising map to reinforcement learning. *Neural Networks*, 15(8–9), 1107–1124.

Stanley, K. O. & Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2), 99–127.

Sutton, R. S. (1984). *Temporal credit assignment in reinforcement learning*. Unpublished PhD dissertation, University of Massachusetts, Amherst, MA.

Sutton, R. S. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems 8* (pp. 1038–1044). Cambridge, MA: MIT Press.

Sutton, R. S. & Barto, A. G. (1998). *Reinforcement Learning*. Cambridge, MA: MIT Press.

Sutton, R. S., McAllester, D., Singh, S., & Mansour, Y. (2000). Policy gradient method for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems 12* (pp. 1057–1063). Cambridge, MA: MIT Press.

Sutton, R. S., Maei, H. R., Precup, D., Bhatnagar, S., Silver, D., Szepesvári, C., & 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, Montreal, Quebec, 14–18 June* (pp. 993–1000). doi: 10.1145/1553374.1553501

Sutton, R. S., Szepesvári, C., & Maei, H. R. (2009b). A convergent $o(n)$ algorithm for off-policy temporal-difference learning with linear function approximation. In *Advances in Neural Information Processing Systems 21* (pp. 1609–1616). Cambridge, MA: MIT Press.

Szepesvári, C. (2010). *Algorithms for Reinforcement Learning*. San Rafael, CA: Morgan & Claypool.

Tan, A.-H., Lu, N., & Xiao, D. (2008). Integrating temporal difference methods and self-organizing neural networks for reinforcement learning with delayed evaluative feedback. *IEEE Transactions on Neural Networks*, 19(2), 230–244.

Taylor, M. E. & Stone, P. (2009). Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(1), 1633–1685.

Tesauro, G. (1992). Practical issues in temporal difference learning. *Machine Learning*, 8(3–4), 257–277.

Tesauro, G. (1995). Temporal difference learning and TD-Gammon. *Communications of the ACM*, 38(3), 58–68.

Tesauro, G., Jong, N. K., Das, R., & Bennani, M. N. (2007). On the use of hybrid reinforcement learning for autonomic resource allocation. *Clustering Computing*, 10(3), 287–299.

Thrun, S. (1995). Learning to play the game of Chess. In *Advances in Neural Information Processing Systems 7* (pp. 1069–1076). Cambridge, MA: MIT Press.

Thrun, S. & Schwartz, A. (1993). Issues in using function approximation for reinforcement learning. In Mozer, M., Smokensky, P., Touretzky, D., Elman, J., & Weigand, A. (Eds.), *Proceedings of the 4th Connectionist Models Summer School, Pittsburgh, PA, 2–5 August* (pp. 255–263). Hillsdale, NJ: Lawrence Erlbaum.

Torrey, L. (2009). *Relational transfer in reinforcement learning*. Unpublished PhD dissertation, University of Wisconsin, Madison, WI.

Touzet, C. F. (1997). Neural reinforcement learning for behaviour synthesis. *Robotics and Autonomous Systems*, 22(3–4), 251–281.

Tsitsiklis, J. N. & Roy, B. V. (1996). Feature-based methods for large scale dynamic programming. *Machine Learning*, 22(1–3), 59–94.

Tsitsiklis, J. N. & Roy, B. V. (1997). An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42(5), 674–690.

van Eck, N. J. & van Wezel, M. (2008). Application of reinforcement learning to the game of othello. *Computers & Operations Research*, 35(6), 1999–2017.

van Hasselt, H. & Wiering, M. A. (2007). Reinforcement learning in continuous action spaces. In *Proceedings of the 2007 IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL), Honolulu, HI, 1–5 April* (pp. 272–279). Retrieved from http://webdocs.cs.ualberta.ca/~ vanhasse/ papers/Reinforcement_Learning_in_ Continuous_Action_Spaces.pdf

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(Jun), 2045–2094.

Veness, J., Silver, D., Uther, W., & Blair, A. (2009). Bootstrapping from game tree search. In Bengio, Y., Schuurmans, D., Lafferty, J. D., Williams, C. K. I., & Culotta, A. (Eds.), *Advances in Neural Information Processing Systems 22* (pp. 1937–1945). Red Hook, NY: Curran Associates, Inc.

Watkins, C. J. C. H. (1989). *Learning from delayed rewards*. Unpublished PhD dissertation, King's College, Cambridge, England.

Watkins, C. J. C. H. & Dayan, P. (1992). Q-learning. *Machine Learning*, 8(3–4), 279–292.

Werbos, P. J. (1974). *Beyond regression: New tools for prediction and analysis in the behavioural sciences*. Unpublished PhD dissertation, Harvard University, Cambridge, MA.

Werbos, P. J. (1989). Backpropagation and neurocontrol: A review and prospectus. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN), Washington, D.C., 18–22 June* (pp. 209–216). doi: 10.1109/IJCNN. 1989.118583

Whiteson, S. & Stone, P. (2006). Evolutionary function approximation for reinforcement learning. *Machine Learning Research*, 7, 877–917.

Whiteson, S., Tanner, B., Taylor, M. E., & Stone, P. (2009). Generalized domains for empirical evaluations in reinforcement learning. In *Proceedings of the 26th International Conference on Machine Learning: Workshop on Evaluation Methods for Machine Learning, Montreal, Canada, 14–18 June*. Retrieved from http://www.site.uottawa.ca/ICML09WS/papers/w8.pdf

Whiteson, S., Taylor, M. E., & Stone, P. (2010). Critical factors in the empirical performance of temporal difference and evolutionary methods for reinforcement learning. *Journal of Autonomous Agents and Multi-Agent Systems*, 21(1), 1–35.

Whiteson, S., Tanner, B., Taylor, M. E., & Stone, P. (2011). Protecting against evaluation overfitting in empirical reinforcement learning. In *Proceedings of the IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL), Paris, France, 11–15 April* (pp. 120–127). doi: 10.1109/ ADPRL.2011.5967363

Wiering, M. A. (1995). *TD learning of game evaluation functions with hierarchical neural architectures*. Unpublished masters thesis, Department of Computer Science, University of Amsterdam, Amsterdam, Netherlands.

Wiering, M. A. (2010). Self-play and using an expert to learn to play backgammon with temporal difference learning. *Journal of Intelligent Learning Systems & Applications*, 2(2), 57–68.

Wiering, M. A. & van Hasselt, H. (2007). Two novel on-policy reinforcement learning algorithms based on TD(λ)-methods. In *Proceedings of the IEEE International Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL), Honolulu, HI, 1–5 April* (pp. 280–287). doi: 10.1109/ADPRL.2007.368200

Wiering, M. A. & van Hasselt, H. (2008). Ensemble algorithms in reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics*, 38(4), 930–936.

Wiering, M. A., Patist, J. P., & Mannen, H. (2007). *Learning to play board games using temporal difference methods* (Technical Report UU–CS–2005-048, Institute of Information and Computing Sciences, Utrecht University). Retrieved from http://www.ai.rug.nl/~ mwiering/GROUP/ARTICLES/learning_games_TR.pdf.

Wierstra, D., Foerster, A., Peters, J., & Schmidhuber, J. (2007). Solving deep memory POMDPs with recurrent policy gradients. In *Proceedings of the 17th International Conference on Artificial Neural Networks (ICANN), Paris, France, 9–13 September* volume 4668 of *Lecture Notes in Computer Science* (pp. 697–706). doi: 10.1007/978-3-540-74690-4_71

Wierstra, D., Förster, A., Peters, J., & Schmidhuber, J. (2010). Recurrent policy gradients. *Logic Journal of the IGPL*, 18(5), 620–634.

Yamada, K. (2011). Network parameter setting for reinforcement learning approaches using neural networks. *Journal of Advanced Computational Intelligence and Intelligent Informatics*, 15(7), 822–830.

Yan, X., Diaconis, P., Rusmevichientong, P., & Roy, B. V. (2004). Solitaire: Man versus machine. In *Advances in Neural Information Processing Systems 17* (pp. 1553–1560). Cambridge, MA: MIT Press.

Yoshioka, T., Ishii, S., and Ito, M. (1999). Strategy acquisition for the game 'Othello' based on reinforcement learning. *IEICE Transactions on Information and Systems*, E82-D(12), 1618–1626.

Chapter 3
Design of Experiments

In this chapter, we review relevant concepts from the field of design of experiments, and this review assumes some basic knowledge of the field. We review both classical and contemporary design of experiments methods. Classical methods are well-established and have a long history of use in many applications; some of these include factorial designs, ANOVA (analysis of variance), and response surface modeling amongst others. The contemporary methods considered are those that are suited for design of experiments for computer simulations, which are based on some fundamental differences from classical experiments. These methods are primarily based on the experimental design and the creation of metamodels of response surfaces (i.e., surrogate models that could be use replacements for true computational models).

Design of experiments (DoE) (Fisher 1935) is the formal process of designing an experimental protocol and analyzing the empirically collected data in order to discover valid and objective information about an underlying system (Montgomery 2008). This definition is deliberately vague because of the very wide applicability of this approach. The term *design of experiments* can be interpreted as the selection of factor-level combinations to be tested or evaluated, where a factor is one variable of interest. However, it is important to note that the experimental design and the method used to analyzed the data are often tightly coupled due to various assumptions on the system under study, the design, or the analytic approach. Although the difference is subtle, in this work the term *experimental design* will refer to the selection of the factor-level combinations to be tested, and *design of experiments* will refer to the inclusive process of determining the experimental design as well as analyzing the data obtained through experimentation.

The design of experiments process must consider the desired information to be gained from the analysis. The desired information plays an important role in the design of experiments process and it can be viewed as what drives the conception of the design of experiments (Fig. 3.1). What is considered to be useful information is often subject area-specific and can take many forms, some of which include identifying differences between groups, determining effects of and interactions between factors, determining factor-level settings to optimize some measure of performance, or determining the probability of some event given factor-level settings, amongst others. The information desired, as well as characteristics of the experiment and of

C. Gatti, *Design of Experiments for Reinforcement Learning,*
Springer Theses, DOI 10.1007/978-3-319-12197-0_3

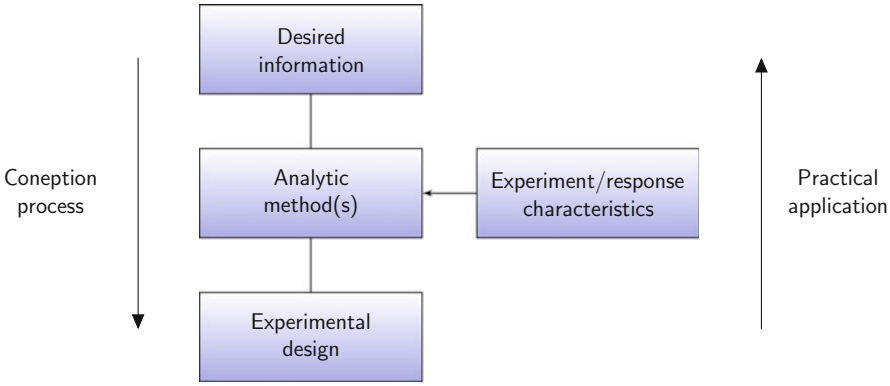


Fig. 3.1 The design and practical application processes of design of experiments. The experimental design process starts from the desired information, which, along with characteristics of the experiment and response, helps select a potential analytic method, and then directs the experimental design to be run. The practical application works in the reverse manner, beginning with testing all design points in the experimental design, analyzing the data, and finally obtaining the desired information.

the response, then limits the analytic methods that could be used. Once a particular analytic method is selected, an appropriate and well-paired experimental design can be constructed.

While the conception of an experimental approach proceeds in a top-down manner, as in Fig. 3.1, the practical application and implementation of these steps work in the reverse order. The application of the design of experiments begins by performing the experiment and collecting data. These data are then analyzed using statistical methods, which provides the desired information.

Figure 3.1 may suggest that the conception of the experimental procedure is a clear and step-wise process, consisting of identifying particular information, which leads to a single analytic method, which then leads to a single experimental design. However, this is hardly the case in practice. In ideal cases, there is a particular analytic method that is well-suited to the characteristics of the experiment and data, and this method provides all of the desired information; this intersection is represented by the overlapping region in Fig. 3.2. However, in many cases, there may not be a single analytic method that yields all of the desire information, or that is ideally suited for the peculiarities of the data collected, and there is often not a perfect overlapping region as in Fig. 3.2. Rather, each analytic method has advantages and disadvantages with regards to the information that it can provide and the characteristics of the data to which it is applied. Thus, a trade-off must therefore be made that considers all aspects of the design of experiments procedure.

In general, design of experiments techniques can be categorized as classical and contemporary (Fig. 3.3), or alternatively, as real-world (physical) experiments and computer experiments that are either deterministic or stochastic (Kleijnen 2008b).

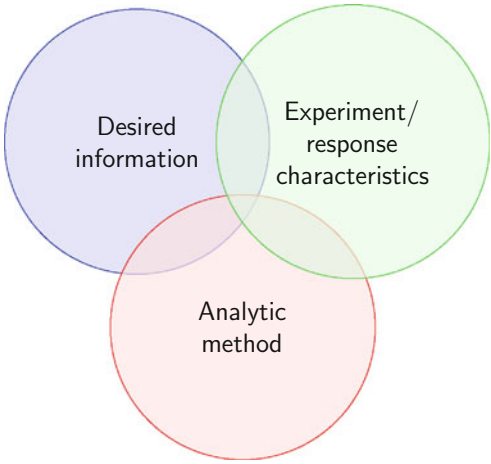


Fig. 3.2 An ideal design of experiments has overlapping regions among the desired information, analytic method, and the experiment characteristics. In real experiments, there may be no overlapping region and a trade-off must be made in one or more of these areas.

The classical and contemporary techniques take slightly different approaches to designing and analyzing experiments because the systems or processes under study often have unique and different characteristics. This section describes these two different types of experiments and some methods related to their design and analysis. We do not provide an exhaustive review, and the interested reader is directed to more comprehensive works for a more thorough review of these methods, especially Montgomery (2008) and the references in this section.

3.1 Classical Design of Experiments

In this work, we define *classical* design of experiments to be those techniques that were developed largely for discovering knowledge about physical and real-world processes or systems. Some of these methods include (fractional) factorial designs, analysis of variance (ANOVA), Taguchi methods, and response surface modeling. Though many of these methods were developed decades ago, and in spite of their relative simplicity, they have proven to be extremely useful. The types of experiments we consider are those that primarily relate to determining the effects of factors or interactions between factors. The types of experiments or situations for which these methods are most suitable include those with random experimental error, relatively few factors, and relatively few factor-levels. Additionally, classical design of experiments approaches require practices of randomization, replication, and blocking.

Experimental error is the random and uncontrollable variation of a response variable (Montgomery 2008). More specifically, for the same factor-level combination, the response will have some natural variation and it will have a non-deterministic response. Due to this experimental error, a common practice is to spread out design points to obtain more accurate factor effect estimates. These experimental techniques

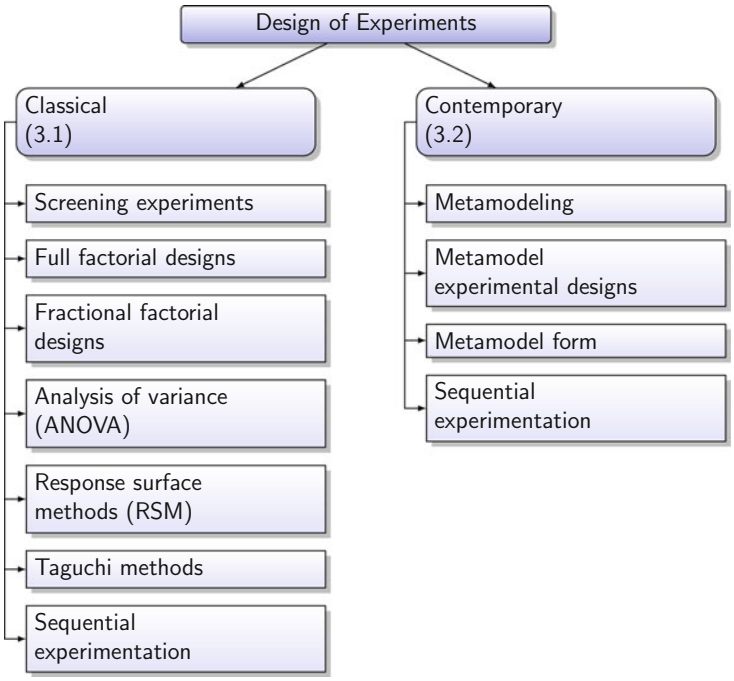


Fig. 3.3 Taxonomy of design of experiments. This figure shows some of the commonly used approaches for classical and contemporary methods, which are described in this section. Numbers in parentheses refer to the section number for the corresponding approach in this chapter.

are often limited to few factors and factor-levels because they are often used to test physical systems or processes, and testing such systems requires time, manpower, and money, and all of these things place constraints on the number of experimental runs that can be performed.

Randomization, replication, and blocking are three cornerstone practices used in classical design of experiments, though their use can be extended to contemporary design of experiments depending on the experimental characteristics. These methods are used in order to more accurately and more precisely estimate the effects of factors. Randomization includes the random selection of experimental units, the random allocation of experimental units, and the random ordering of experimental runs (Montgomery 2008). This practice aims to reduce the effects of potentially uncontrollable, but systemic, effects that are present in the system under study. Replication is the use of multiple experimental runs for the same factor-level combination, where each individual run is a single replication. This practice allows for the estimation of experimental error and for more precise estimates of fitted model parameters (Montgomery 2008). Blocking is the practice of partitioning experimental units into similar groups to reduce the effect of an uncontrollable factor (i.e., nuisance factor).

We review only a subset of classical design of experiments techniques and concepts here, which may relate to the experimental design, analysis, or a coupled experimental design and analysis technique. Additionally, the points discussed here provide but a brief overview of these methods, and the reader is directed to texts that are devoted to design of experiments, e.g., Myers and Montgomery (2002) and Montgomery (2008). The first set of methods we describe are related to experimental design, that is, the selection of which factor-level combinations to be run in an experiment.

Screening Experiments Screening experiments are used in the preliminary stages of a design of experiments. These experiments are used when there is a large set of factors that may have an influence on the response variable, though it is unknown which factors actually have significant effects. A screening experiment can be used to determine which factors, from the large set of k factors, have an influence on a response variable. Each factor is evaluated at two levels, resulting in a 2^k design. Note that at this stage of experimentation, the main interest is simply determining if there is a change in the response variable for a change in a given input variable. The relationship between the factor-levels and the response variable is therefore assumed to be linear. After identifying which factors influence the response variable, subsequent experiments can be performed to determine if there are main and/or interaction effects between factors.

Full Factorial Designs Full factorial designs evaluate multiple factors, with each factor having multiple levels. Every possible factor-level combinations is run in the experiment, and thus this design could be considered an exhaustive design. For k factors, each of which with n factor-levels, there are n^k unique factor-level combinations. While full factorial designs can be provide a wealth of information, they are frequently used in practice only for small numbers of factors and levels because of the investment (e.g., time, money, manpower) required to run all experimental combinations with replication.

Fractional Factorial Designs Fractional factorial designs are based on full factorial designs, however, they only use a subset n^{k-p} of the possible factor-level combinations, where p determines the fraction of the full factorial experiment. The use of fractional factorial designs is based on the assumption that high-order interactions are negligible. These designs leverage this assumption by aliasing main effects or interaction effects; that is, an effect may not be uniquely determined from the fractional factorial design alone, and follow-up experiments are required to precisely determine the true effects. A larger value of p results in a smaller design, though it also results in more effects being aliased to each other, and this is a trade-off that must be made by the experimenter. Additional information on fractional factorial designs can be found in Montgomery (2008).

Analysis of Variance Often an experimenter is interested in whether individual factors or interactions between factors have a significant effect on a response. The most widely-used analytic method is analysis of variance (ANOVA), which can be used to analyze data collected from many types of experimental designs, including

those previously described. Analysis of variance is used to analyze experimentally collected data to test for differences in group means for more than two groups. ANOVA works by partitioning the observed variance into that which can be explained (based on the data and an associated regression model) and that which cannot be explained. Using sum-of-squares decomposition and statistical tests comparing the explained and unexplained variance, one can determine the significance of model terms, whether they are single main effects or interaction effects. ANOVA is based on three assumptions, those being: that the response variable is normally distributed, that each group has equal variance (i.e., homoscedasticity), and that observations are independent. It should be noted, however, that truly normally distributed data are rarely seen in practice, and that ANOVA can still provide useful information with deviations from the normality assumption. Additionally, the simplest use of ANOVA requires equal numbers of observations at each factor-level and uses Type I sum-of-squares, but Type II and III sum-of-squares can be used with unequal numbers of factor-level observations.

While ANOVA is relatively independent of the experimental design, alternative design of experiments techniques, such as the response surface methodology (RSM), use experimental designs that are complementary to the analysis methods.

Response Surface Methods As the name implies, response surface methods are used to create a surface that approximates the behavior of a response variable based on a set of continuous and independent factors (Box and Wilson 1951; Box 1954; Myers and Montgomery 2002). Response surfaces can be used to identify significant effects and the form of effects, to act as a surrogate model for the system under study, or to determine a set of operating conditions that produce and optimal response. In the case of optimization, a preliminary linear model is developed using a simplistic experimental design (e.g., fractional factorial or simplex designs), and this model is used to direct and guide subsequent experimentation toward the optimal operating point. Once sufficiently close to the optimum, a second-order model, consisting of linear, quadratic, and two-way interaction effects, is developed with a more sophisticated experimental design, such as central composite designs (CCD) or Box-Behnken designs (BBD) (Box and Wilson 1951; Box 1954; Myers and Montgomery 2002). The approximate optimal operating point can then be determined through canonical analysis of this second-order model. While response surface methods have been found to be sufficient for physical and real-world experiments (Box and Draper 1987), its low-order model terms may limit its applicability to more complex response surfaces (such as from computer simulations) (Sacks et al. 1989; Vining 2008).

Taguchi Methods Taguchi design of experiment techniques were originally developed to improve the quality of manufactured goods, but their application has extended to numerous domains, including biotechnology (Antolín et al. 2002), material science (Gell et al. 2001), and supply chain optimization (Shang et al. 2004), amongst many others. This approach is based on the assumption that only single factors (and not interactions among multiple factors) have an effect on response variables. Consequently, this drastically reduces the number of experimental runs that are required to determine the effects of these factors, and this is the main attractiveness of Taguchi

methods. The assumption that only low-order effects are present can be difficult to accept, especially when studying complex systems for which there is little known, and thus Taguchi methods are often applied when the system is relatively well understood. While Taguchi methods are not used in this work, these methods represent an important and significant part of design of experiments.

While the reinforcement learning approach used in this work is assessed purely from a quantitative perspective, the knowledge that is learned and the behavior of the agent could also be assessed from a qualitative perspective. For example, the learned behavior from one set of parameters could be quite different than that from another set of parameters. Selecting which behavior is superior is dependent on one's definition of *better*. It is possible that reinforcement learning runs could be studied using Taguchi methods by assessing them in terms of quality and robustness (e.g., robust to different initial conditions or domain variations). While this would be an interesting approach, it is not developed here any further but it may be considered in future work.

Sequential Experimentation A basic design of experiments approach may be based on a single experimental design and analysis. However, it is often the case where the results of an experiment provide information that leads to additional questions, and thus additional experiments as well. Sequential experimentation is the process of iterating between experimentation and analysis, and it is often suggested over performing a single experiment due to the uncertainty in the utility of a particular experimental design. Based on the outcome of a preliminary experiment, an experimenter may want to alter the experimental design in some fashion, such as: moving or rescaling the design space, adding design points to resolve ambiguities, or adding replications to better estimate experimental error (Box 1992). As mentioned above, sequential experimentation is commonplace in the response surface methodology where low-order models direct subsequent experimental designs, and higher-order models are used when in the vicinity of the optimal operating point. Sequential experimentation for classical experiments is not a new technique and dates back to the works of Robbins (1952), Chernoff (1959), and Chernoff (1973), though its use in contemporary experiments is not as widespread as one would think given the potential utility of the approach.

3.2 Contemporary Design of Experiments

We define *contemporary* design of experiments to be those techniques that were largely developed for analyzing computer experiments. That is, experiments are run on a computer and take the form of simulations with some set of input parameters. A classic example is finite element modeling for assessing structural reliability, determining the probability of failure under different conditions (Sudret 2012), or uncertainty analysis (Wu et al. 1990). This field has adopted the names Design and Analysis of Computer Experiments (DACE) (Santner et al. 2003) or Design and Analysis of Simulation Experiments (DASE) (Kleijnen 2008a), and

is largely restricted to the design and analysis for deterministic computer models (Sacks et al. 1989; Santner et al. 2003; Chen et al. 2006). Such models produce the same response every time for the same set of input parameters, and thus the experimental error from *classical* experiments is not present. Furthermore, because of the absence of experimental error, *classical* design of experiments practices such as randomization, replication, and blocking are not necessary.

The work herein consists of simulations that are based on a learning algorithm that is not deterministic, and experimental error is present in the response variables from the simulations. The presence of experimental error in these experiments means that some practices of *classical* design of experiments techniques must be utilized to deal with experimental error, and in particular, replication. Thus, the experiments in this work have qualities of both *classical* and *contemporary* design of experiments, and techniques from both approaches will need to be employed. In this section, we describe approaches that are commonly used design of experiments techniques for computer simulations.

Metamodeling Metamodeling, also known as surrogate modeling, is the process of developing a statistical model that approximates the response of a system based on a set of input variables or parameters (Chen et al. 2006; Barton 2009; Sudret 2012). A response surface model, as described above, is one example of a metamodel. The use of metamodeling has grown in large part due to the increased use of computationally expensive computer simulations (e.g., each run takes hours to days of computation time). In these cases, the primary use of metamodeling is to develop a model that sufficiently approximates the response of the simulation but that can be evaluated 'on-demand', essentially minimizing the use of or replacing the computationally expensive simulation. Although acting as a surrogate model for the purposes of prediction, optimization, or model tuning is often the purpose of metamodels, their use is not limited to approximating simulation responses. Depending on the statistical form of the metamodel, it may provide more concrete and interpretable knowledge of the response of simulations, and such knowledge is of particular interest in this work. Similar to classical design of experiments, the development of the metamodel requires careful consideration of two tasks: determining the experimental runs that are used to build the metamodel, and specifying the form of metamodel (Sacks et al. 1989; Chen et al. 2006; Barton 2009). As previously mentioned, there may not be an experimental design or statistical model that perfectly complements the qualities of the experiment, and this holds for metamodeling as well, which often requires compromises on the design or modeling approach. An exhaustive review of metamodling is beyond the scope of this work, however, below we discus some of the experimental designs and statistical models used in metamodeling. The reader is directed to works by Chen et al. (2006) and Barton (2009) for excellent reviews that thoroughly compare these designs and models.

Experimental Designs of Metamodels Common experimental designs for metamodels include response surface method designs and Number-Theoretic Methods (NTM) (Chen et al. 2006). Design methods for response surface methods are mentioned above. Number-Theoretic Methods can be referred to as a *space-filling*

design that attempts to uniformly fill the design space by minimizing a discrepancy measure that is based on the distribution of the design points (Pronzato and Müller 2012). These designs are based on low-discrepancy random number sequences, which have the quality that the number of points falling into an arbitrary portion of the design space is proportional to a measure of this design subspace. Such sequences are often based on quasi-random numbers, which can be constructed sequentially (Pronzato and Müller 2012), and which result in a more uniformly distribution than conventional pseudo-random numbers (Tuffin 1998; Caflisch 1998). The improved sampling uniformity ultimately reduces the total computational expense of Monte Carlo simulations because quasi-random sequences converge faster than pseudo-random sequences. Examples of low-discrepancy sequences include the Halton (Halton 1960), Hammersley (Hammersley 1960), Sobol' (Sobol' 1967), Faure (Faure 1982), and Niederreiter (Niederreiter 1992) sequences. While quasi-random sequences were largely developed for Monte Carlo integration (Niederreiter 1992), their use in experimental design and statistics has been growing (Fang and Wang 1994; Chen et al. 2006).

Statistical Form of Metamodels The statistical form of the metamodel can vary greatly depending on experiment and response characteristics, the desired information from the metamodel, and the desired use of the metamodel. Here, we briefly describe some types of models that have been used for metamodeling. While conventional response surface models were largely developed for physical experiments, they have been applied to computer experiments. However, their success is rather mixed, and this may be due to their low-order model, which assumes that the response surface is relatively smooth when in fact computational models often have more complex response surfaces (Vining 2008). Nonetheless, response surface models may be of use for understanding the behavior of a few (< 4) variables under specific circumstances.

Classification and regression trees (CART) create a decision tree-like representation of the response by recursively partitioning the design space into hyper-rectangles, with each partition having its own approximation of the response surface (Breiman et al. 1984; Loh 2011). These models, however, are not robust to outliers or skewed response variables (Galimberti et al. 2011). Random forests attempt to improve the robustness of CARTs by using an ensemble of CARTs where the predicted response values are a consensus value from all of the individual CARTs (Breiman 2001). This approach, however, significantly increases the computational expense of model creation. Multi-variate adaptive regression splines (MARS) can be thought of as an extension of CARTs that use piecewise linear surfaces that are splined together to approximate a complete, and potentially complex, response surface (Friedman 1991). Each piecewise surface is parameterized by knot locations (i.e., the extent of their domain) and either single variables or interactions between variables.

Artificial neural networks are another approach to metamodeling when they are used as a regression model, and they are appealing because they can model highly nonlinear responses. Neural networks are not often used in traditional design of experiments approaches, possibly due to the disconnect between the design of

experiments and machine learning communities, however neural networks can still offer insights that are comparable to other modeling approaches. For example, variable sensitivity can be determined using the methods of Zurada et al. (1994) and Kewley et al. (2000).

Kriging, also known as spatial correlation modeling, was developed for geostatistical modeling (Matheron 1963) but has been extended to computer simulations (Sacks et al. 1989; Ankenman et al. 2010). Kriging creates interpolated surfaces using weighted combinations of surrounding points, where the weighs are based on the distance between points and their specific locations. Due to its roots being in geostatistical modeling, kriging is traditionally a deterministic modeling technique that provides an exact interpolation and does not allow for experimental error. This approach has therefore been an attractive method for deterministic computer simulations and experiments (van Beers and Kleijnen 2004). Deterministic kriging has recently been extended to stochastic kriging, which allows for responses with experimental error (Kleijnen 2009; Staum 2009; Ankenman et al. 2010), thereby also extending the appeal of this modeling technique to non-deterministic situations. Additionally, the basic kriging model can be modified to provide ANOVA-like information regarding the global sensitivity of factors (and interactions) by using different types of kernels (Durrande et al. 2012; Muehlenstaedt et al. 2012).

Sequential Experimentation in Computer Experiments In recent years, sequential experimentation based on computer models or simulations has become a topic of increasing interest (Williams et al. 2000; Lam 2008; Kleijnen 2009; Loeppky et al. 2010; Bect et al. 2012), likely due to the appeal and potential efficiency of sequential experimentation as well as the ability to perform computer experiments for which the only significant expense is computation time. As with metamodeling in general, the case for sequential computer experiments is also motivated by computer models that either require long computation times or that are very complex and have numerous input parameters. The process of sequential experimentation in computer experiments is slightly different than that of physical experiments: sequential physical experiments tend to select groups of design points or complete experimental designs for subsequent experiments, whereas sequential computer experiments select either individual or small groups of design points for subsequent experimental runs. Sequential computer experiments often proceed in the following manner: perform a pilot experiment using a small experimental design, fit a model to the pilot data, select the next experimental design point(s) based on some criterion of the model (e.g., predictor variance), run the new design point and obtain its output, repeat the model fitting and selection of subsequent design points until the model is sufficiently accurate (Williams et al. 2000; Kleijnen 2009; Pronzato and Müller 2012).

Considering that sequential computer experiments are often used to develop metamodels for computationally intense simulations, the use of the metamodels is largely focused on prediction or optimization. However, the assessment of parameter effects, as in a traditional ANOVA-sense, can also be obtained from sequential experimentation. In this case, the locations of the design points that are selected for subsequent

experimental runs are analyzed, and these points can be indicative of factor sensitivity (Kleijnen 2009). More formal spatial statistics methods can also be useful in understanding parameter effects by using variograms and its relatives (Journel and Huijbregts 1978; Cressie 1993).

3.3 Design of Experiments for Empirical Algorithm Analysis

Empirical analyses of machine learning algorithms in general is a relatively standard approach for benchmarking and testing algorithms. The case is no different with respect to reinforcement learning, where there are countless examples in which authors present figures showing the performance of the agent over the course of learning. In these cases, when multiple learning algorithms are evaluated or when parameters of a single learning algorithm are varied, multiple learning performance curves are presented in order to give the reader a sense of the relative speed of learning and the maximal performance for each algorithm or parameter variation.

The literature is relative scarce with respect to the use of a design of experiments approach to evaluating learning algorithms (not necessarily reinforcement learning algorithms) or heuristics empirically, though this scarcity and lack of rigor have been acknowledged (Hooker 1995; Eiben and Jelasity 2002). Parsons and Johnson (1997) use response surface methods with central composite and fraction factorial designs to improve the performance of genetic algorithms for DNA sequence assembly. Park and Kim (1998) use a non-linear response surface method to select parameters for simulating annealing and show the effectiveness of this approach in graph partitioning problems, flowshop scheduling problems, and production scheduling problems. Coy et al. (2000) use a design of experiments approach with response surface methods to find optimal parameters for heuristics that are commonly used in vehicle routing problems. Shilane et al. (2008) develop an approach to statistically compare evolutionary algorithms.

Perhaps the largest series of work in this area belongs to Ridge and Kudenko (2006, 2007a, b, c, 2008) with their work investigating the ant colony optimization (ACO) algorithm. Ridge and Kudenko (2006) thoroughly outlines a potential design of experiments approach for studying the ACO algorithm, including the use of screening experiments, de-aliasing effects, and response surface methods. Their work details a sequential experimentation procedure based on a screening experiment and response surface methods to gain an initial understanding of algorithm parameters. Similar work was applied to studying the ACO algorithm by using a fractional factorial design to understand the effects of 12 parameters (Ridge and Kudenko 2007b), and by using response surface methods to optimize algorithm parameters (Ridge and Kudenko 2007a, c). Ridge and Kudenko (2008) again investigated the effects of ACO parameters, this time using a hierarchical (nested) design that was analyzed using a general linear model with fixed and random effects.

References

Ankenman, B., Nelson, B. L., & Staum, J. (2010). Stochastic kriging for simulation metamodeling. *Operations Research*, 58(2), 371–382.

Antolín, G., Tinaut, F. V., Briceño, Y., Castaño, V., Pérez, C., & Ramírez, A. I. (2002). Optimisation of biodiesel production by sunflower oil transesterification. *Bioresource Technology*, 83(2), 111–114.

Barton, R. R. (2009). Simulation optimization using metamodels. In Rossetti, M. D., Hill, R. R., Johansson, B., Dunkin, A., & Ingalls, R. G. (Eds.), *Proceedings of the 2009 Winter Simulation Conference, Austin, TX, 13–16 December* (pp. 230–238). doi: 10.1109/WSC.2009.5429328

Bect, J., Ginsbourger, D., Li, L., Picheny, V., & Vazquez, E. (2012). Sequential design of computer experiments for the estimation of a probability of failure. *Statistics and Computing*, 22(3), 773–793.

Box, G. E. P. (1954). The exploration and exploitation of response surfaces: Some general considerations and examples. *Biometrics*, 10(1), 16–60.

Box, G. E. P. (1992). Sequential experimentation and sequential assembly of designs. *Quality Engineering*, 5(2), 321–330.

Box, G. E. P. & Draper, N. R. (1987). *Empirical Model-Building and Response Surfaces*. New York, NY: Wiley.

Box, G. E. P. & Wilson, K. B. (1951). On the experimental attainment of optimum conditions. *Journal of the Royal Statistical Society, Series B*, 13(1), 1–45.

Breiman, L. (2001). Random forests. *Machine Learning*, 45(1), 5–32.

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and Regression Trees*. New York, NY: Chapman & Hall.

Caflisch, R. E. (1998). Monte Carlo and quasi-Monte Carlo methods. *Acta Numerica*, 7, 1–49.

Chen, V. C. P., Tsui, K.-L., Barton, R. R., & Mechesheimer, M. (2006). A review on design, modeling and applications of computer experiments. *IIE Transactions*, 38(4), 273–291.

Chernoff, H. (1959). Sequential design of experiments. *The Annals of Mathematical Statistics*, 30(3), 755–770.

Chernoff, H. (1973). *Approaches in sequential design of experiments* (Technical Report Number 77, Department of Statistics, Stanford University). Retrieved from https://statistics.stanford.edu/sites/default/files/CHE ONR 77.pdf.

Coy, S. P., Golden, B. L., Runger, G. C., & Wasil, E. A. (2000). Using experimental design to find effective parameter settings for heuristics. *Journal of Heuristics*, 7(1), 77–97.

Cressie, N. A. C. (1993). *Statistics for Spatial Data (2nd edition)*. New York, NY: Wiley.

Durrande, N., Ginsbourger, D., Roustant, O., & Carraro, L. (2012). ANOVA kernels and RKHS of zero mean functions for model-based sensitivity analysis. *Journal of Multivariate Analysis*, 115(March), 57–67.

Eiben, A. E. & Jelasity, M. (2002). A critical note on experimental research methodology in EC. In *Proceedings of the 2002 Congress on Evolutionary Computing (CEC), Honolulu, HI, 12–17 May* (pp. 582–587). doi: 10.1109/CEC.2002. 1006991

Fang, K. T. & Wang, Y. (1994). *Number-Theoretic Methods in Statistics*. New York, NY: Chapman & Hall.

Faure, H. (1982). Discrepancy of sequences associated with a number system (in dimension s). *Acta Arithmetica*, 41(4), 337–351.

Fisher, R. A. (1935). *The design of experiments*. Edinburgh, Scotland: Oliver and Boyd.

Friedman, J. H. (1991). Multivariate adaptive regression splines (with discussion). *Annals of Statistics*, 19(1), 1–141.

Galimberti, G., Pillati, M., & Soffritti, G. (2011). Notes on the robustness of regression trees against skewed and contaminated errors. In Ingrassia, S., Rocci, R., & Vichi, M. (Eds.), *New Perspectives in Statistical Modeling and Data Analysis (Studies in Classification, Data Analysis, and Knowledge Organization)* (pp. 255–263). New York, NY: Springer-Verlag.

Gell, M., Jordan, E. H., Sohn, Y. H., Goberman, D., Shaw, L., & Xiao, T. D. (2001). Development and implementation of plasma sprayed nanostructured ceramic coatings. *Surface and Coatings Technology*, 146–147(Sept–Oct), 48–54.

Halton, J. (1960). On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. *Numerische Mathematik*, 2(1), 84–90.

Hammersley, J. M. (1960). Monte carlo methods for solving multivariate problems. *Annals of the New York Academy of Sciences*, 86, 844–874.

Hooker, J. N. (1995). Testing heuristics: We all have it wrong. *Journal of Heuristics*, 1(1), 33–42.

Journel, A. G. & Huijbregts, C. J. (1978). *Mining Geostatistics*. London, UK: Academic Press.

Kewley, R. H., Embrechts, M. J., & Breneman, C. (2000). Data strip mining for the virtual design of pharmaceuticals with neural networks. *IEEE Transactions on Neural Networks*, 11(3), 668–679.

Kleijnen, J. P. C. (2008a). *Design and Analysis of Simulation Experiments*. New York, NY: Springer.

Kleijnen, J. P. C. (2008b). Design of experiments: Overview. In Maxon, S. J., Hill, R. R., Mönch, L., Rose, O., Jefferson, T., & Fowler, J. W. (Eds.), *Proceedings of the 2008 Winter Simulation Conference (WSC 2008), Miami, Florida, 7–10 December* (pp. 479–488). doi: 10.2139/ssrn.1262179

Kleijnen, J. P. C. (2009). Kriging metamodeling in simulation: A review. *European Journal of Operational Research*, 192(3), 707–716.

Lam, C. Q. (2008). *Sequential adaptive designs in computer experiments for response surface model fit*. Unpublished PhD dissertation, The Ohio State University, Columbus, OH.

Loeppky, J. L., Moore, L. M., & Williams, B. J. (2010). Batch sequential designs for computer experiments. *Journal of Statistical Planning and Inference*, 140(6), 1452–1464.

Loh, W.-Y. (2011). Classification and regression trees. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 1(1), 14–23.

Matheron, G. (1963). Principles of geostatistics. *Economic Geology*, 58(8), 1246–1266.

Montgomery, D. C. (2008). *Design and Analysis of Experiments (7th edition)*. Hoboken, NJ: John Wiley & Sons, Inc.

Muehlenstaedt, T., Roustant, O., Carraro, L., & Kuhnt, S. (2012). Data-driven kriging models based on FANOVA-decomposition. *Statistics and Computing*, 22(3), 723–738.

Myers, R. H. & Montgomery, D. C. (2002). *Response Surface Methodology: Process and Product Optimization Using Designed Experiments (2nd edition)*. New York, NY: Wiley-Interscience.

Niederreiter, H. (1992). *Random Number Generation and Quasi-Monte Carlo Methods*. Philadelphia: SIAM.

Park, M.-W. & Kim, Y.-D. (1998). A systematic procedure for setting parameters in simulated annealing algorithms. *Computers & Operations Research*, 25(3), 207–217.

Parsons, R. and Johnson, M. (1997). A case study in experimental design applied to genetic algorithms with applications to DNA sequence assembly. *Journal of Mathematical and Management Sciences*, 17(3), 369–396.

Pronzato, L. & Müller, W. G. (2012). Design of computer experiments: Space filling and beyond. *Statistics and Computing*, 22(3), 681–701.

Ridge, E. & Kudenko, D. (2006). Sequential experiment designs for screening and tuning parameters of stochastic heuristics. In *Proceedings of the 9th International Conference on Parallel Problem Solving from Nature (PPSN), Reykjavik, Iceland, 9–13 September* (pp. 27–34). Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.64.3468&rep=rep1&type=pdf

Ridge, E. & Kudenko, D. (2007a). Analyzing heuristic performance with response surface models: Prediction, optimization and robustness. In *Genetic and Evolutionary Computation Conference (GECCO), London, England, 7–11 July* (pp. 150–157). doi: 10.1145/1276958.1276979

Ridge, E. & Kudenko, D. (2007b). Screening the parameters affecting heuristic performance. In *Genetic and Evolutionary Computation Conference (GECCO), London, England, 7–11 July* (pp. 180). doi: 10.1145/1276958.1276994

Ridge, E. & Kudenko, D. (2007c). Tuning the performance of the MMAS heuristic. In Stützle, T., Birattari, M., & Hoos, H. H. (Eds.), *Proceedings of the International Workshop on Engineering Stochastic Local Search Algorithms (SLS), Brussels, Belgium, 6–8 September* (pp. 46–60). doi: 10.1007/978-3-540-74446-7_4

Ridge, E. & Kudenko, D. (2008). Determining whether a problem characteristic affects heuristic performance. A rigorous design of experiments approach. In Cotta, C. & van Hemert, J. (Eds.), *Recent Advances in Evolutionary Computation for Combinatorial Optimization. Springer, Studies in Computational Intelligence*, volume 153 (pp. 21–35). New York, NY: Springer.

Robbins, H. (1952). Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 58(5), 527–535.

Sacks, J., Welch, W. J., Mitchell, T. J., & Wynn, H. P. (1989). Design and analysis of computer experiments. *Statistical Science*, 4(4), 409–423.

Santner, T. J., Williams, B. J., & Notz, W. (2003). *The Design and Analysis of Computer Experiments*. New York, NY: Springer.

Shang, J. S., Li, S., & Tadikamalla, P. (2004). Operational design of a supply chain system using the taguchi method, response surface methodology, simulation, and optimization. *International Journal of Production Research*, 42(18), 3823–3849.

Shilane, D., Martikainen, J., Dudoit, S., & Ovaska, S. J. (2008). A general framework for statistical performance comparison of evolutionary computation algorithms. *Information Sciences*, 178(14), 2870–2879.

Sobol', I. M. (1967). On the distribution of points in a cube and the approximate evaluation of integrals. *U.S.S.R. Computational Mathematics and Mathematical Physics*, 7(4), 86–112.

Staum, J. (2009). Better simulation metamodeling: The why, what, and how of stochastic kriging. In Rossetti, M. D., Hill, R. R., Johansson, B., Dunkin, A., & Ingalls, R. G. (Eds.), *Proceedings of the 2009 Winter Simulation Conference, Austin, TX, 13–16 December* (pp. 119–133). doi: 10.1109/WSC.2009.5429320

Sudret, B. (2012). Meta-models for structural reliability and uncertainty quantification. In Phoon, K., Beer, M., Quek, S., & Pang, S. (Eds.), *Proceedings of the* 5th *Asian-Pacific Symposium on Structural Reliability and its Applications (APSSRA 2012), Singapore, 23–25 May* (pp. 53–76). Singapore: Research Publishing Services.

Tuffin, B. (1998). On the use of low discrepancy sequences in Monte Carlo methods. *Monte Carlo Methods and Applications*, 2(4), 295–320.

van Beers, W. C. M. & Kleijnen, J. P. C. (2004). Kriging interpolation in simulation: A survey. In Ingalls, R. G., Rossetti, M. D., Smith, J. S., & Peters, B. A., editors, *Proceedings of the 2004 Winter Simulation Conference, Washington, D.C., 5–8 December* (pp. 113–121). doi: 10.1109/WSC.2004.1371308

Vining, G. G. (2008). Adapting response surface methodology for computer and simulation experiments. In Tsubaki, H., Nishina, K., & Yamada, S. (Eds.), *The Grammar of Technology Development* (pp. 127–134). Japan: Springer.

Williams, B. J., Santner, T. J., & Notz, W. I. (2000). Sequential design of computer experiments to minimize integrated response functions. *Statistica Sinica*, 10(4), 1133–1152.

Wu, Y. T., Millwater, H. R., & Cruse, T. A. (1990). Advanced probabilistic structural-analysis method for implicit performance functions. *Journal of the American Institute of Aeronautics and Astronautics*, 28(9), 1663–1669.

Zurada, J. M., Malinowski, A., & Cloete, I. (1994). Sensitivity analysis for minimization of input data dimension for feedforward neural network. In *IEEE International Symposium on Circuits and Systems (ISCAS), London, England, 30 May–2 June* (Vol. 6, pp. 447–450). doi: 10.1109/ISCAS.1994.409622

Chapter 4
Methodology

Implementing reinforcement learning requires several parameters related to the learning algorithm as well as the representation (i.e., a neural network for the implementation in this work). Additionally, the domain or problem that is to be learned can be characterized across a variety of dimensions, including the state-space, the action-space, the reward function, among many others. This work is built on the fundamental belief that there are subregions in the parameter space of the learning algorithm and the representation that allow for reinforcement learning to be successful for different domain characteristics.

The goal of this work is to investigate under what parameter conditions reinforcement learning works, and furthermore, how these parameters affect the performance. We therefore break this problem into two parts. The first part attempts to find parameter subregions, within a large parameter space, for which reinforcement learning is generally successful; we call these regions *convergent subregions* of the parameter space such that reinforcement learning runs frequently converge. The second part takes a closer look at these convergent subregions and attempts to understand how these parameters affect learning performance and what parameters are the most influential. The problem domains analyzed later in this work use very similar experimental methodologies and analysis procedures, and instead of repeating the methodology used for each problem domain, we present the methods used in this chapter.

4.1 Sequential CART

In this section we describe our novel approach for finding convergent parameter subregions for reinforcement learning. Essentially, this procedure attempts to find these convergent subregions by slowly narrowing down the parameter space using a sequential experimentation that is based on CART (classification and regression trees) models. This section first briefly reviews CART models and then describes the novel sequential CART experimentation approach. We conclude the description of this procedure and hope to make concepts more concrete by demonstrating how the sequential CART procedure works on an example 2-dimensional function.

C. Gatti, *Design of Experiments for Reinforcement Learning,*
Springer Theses, DOI 10.1007/978-3-319-12197-0_4

4.1.1 CART Modeling

Classification and regression trees (CART) (Breiman et al. 1984) are based on a decision tree that can be used in either a regression setting or in a classification setting, as the name implies. This modeling method is based on a recursive partitioning procedure that partitions a region $\mathbb{R}^d$ (where d is the dimensionality of the problem or feature vector) into non-empty, non-overlapping, hyperrectangular, axes-align subregions. Partitioning (splitting) the data involves selecting a variable to split on, as well as selecting a value of that variable at which to split the data. The model is built by sequentially selecting partitions that improve the model fit as much as possible. Consequently, the form of the model is not based on a user-specified model structure, but rather the model form is built from the data itself.

The reader is directed to the seminal work of Breiman and colleagues (Breiman et al. 1984) for a full description of this method and its inner working; however, a brief review will be provided here. As mentioned, CART partitions the data into smaller and smaller subregions, each of which can be defined by minimum and maximum values along each of the d dimensions of the data. Partitioning is done by using a splitting rule or criteria, such as the Gini impurity, and the split that achieves the greatest improvement with respect to the splitting criteria is selected. This algorithm is recursive because subregions from previous iterations are partitioned in subsequent iterations. The resulting binary tree then has interior nodes, which define the split points, and terminal nodes, which define the subregions and have an associated class label (for classification) or value (for regression).

Without some type of stopping criteria, the tree could grow very large and the subregions at the terminal nodes would become very small, resulting in a model that is overfit. Examples of stopping criteria include requiring a minimum number of data points at each interior or terminal node, or requiring a minimum improvement in the performance of the model with the addition of another interior node or split point. A very small tree is undesirable as well because such trees cannot often generalize. Creating a model with an appropriate size is often done by growing a relatively large tree, and then pruning branches such that the performance is good, yet that the data is not overfit. This is quantified by the cost-complexity trade-off $R_\xi(T) = R(T) + \xi \|T\|$ where, for example in regression, $R(T)$ is the residual sum of squares, and $\|T\|$ is the number of terminal nodes in the tree T. An appropriate value of ξ is often selected through cross-validation.

CART modeling has numerous attractive characteristics, including: that the model structure can be visualized and is intuitively interpretable, that it can handle both continuous and categorical variables, that it is relatively resistant to outliers, that it is not based on any assumptions about the data, and that it can handle interactions among many variables without the problem of becoming intractable.

However, one of the primary issues with CART models is that they are sensitive to the data from which they are built, and consequently, they can become unstable with variations in the data. If the goal is to obtain the best accuracy and performance in either regression or classification, model instability can be somewhat resolved by

using an ensemble of CART models, also known as a random forest (Breiman 2001). While random forests are attractive because they likely have better performance than single CART models, they are not suitable for the analysis of interest in this work. This is because we are interested in using the split variables and split points from the CART models in subsequent analysis, and in random forests, each CART model likely has a different structure (i.e., split variables and split points).

4.1.2 Sequential CART Modeling

The goal of sequential CART modeling, in a general sense, is to determine subregions of a parameter space that result in some type of experimental outcome or result. By *subregion*, we are referring to a region defined by minimum and maximum values of a hyperrectangle of the parameter space. With regards to the problem of interest, our goal is to determine subregions of the reinforcement learning parameter space for a specific domain that frequently result in convergent runs, where *convergent* means that learning has reached a steady state and the agent can perform acceptably in the domain of interest. An overview of this sequential experimentation procedure is described below rather informally, and a more formal description and algorithm follows.

The sequential experimentation process begins with a large parameter space defined by lower and upper bounds for each parameter. An initial large and rather coarse experiment is performed using a set of design points $X_0^{(0)}$ that are sampled from the entire parameter space, and each of these design points are run through the simulation (i.e., reinforcement learning in the domain of interest). The parenthetic superscript 0 indicates that this is the initial or original experimental design. The subscript 0 indicates that this subregion is labeled as subregion 0; a second subscript indicates a parent subregion from the experiment, and this will become clear as this process is explained. Table 4.1 presents the notation used for the sequential CART procedure. The design points in $X_0^{(0)}$ can remain in their original unscaled space as this does not have an effect on the CART model. For each design point, a numerical result $Y_0^{(0)}$ of its performance is obtain, and additionally, the type of result $\Psi_0^{(0)}$ is obtained as well. The type of result in our case is binary indicator variable that indicates whether learning converged from a reinforcement learning perspective. Note that in a more general problem, the type of results for the design runs is not limited to two types. The numerical responses are then divided into and labeled as two groups in $Z_0^{(0)}$: a *low* group that has the majority of points, and *high* group that has the remaining points. As an example, the *low* group may consist of 80 % of the points that have the lowest numerical responses, and the *high* group would consist of the remaining 20 % of the points. A CART model $T_0^{(0)}$ is then created that models the groups $Z_0^{(0)}$ as a function of the design runs $X_0^{(0)}$.

The proportions of points to consider as *high* and *low* response groups was selected based on the work of Robertson et al. (2013) who used a similar sequential CART

Table 4.1 Variables and notation used in sequential CART.

Parameter	Description
$\mathcal{L}$	Approximate level set: $\mathcal{L} = \{x \in \bigcup_i A_{q,p}^{(i)}\}$ where $A_{q,p}^{(i)}$ is the q^{th} sub-region of the CART partition on $\mathbb{R}^n$ from iteration i that has parent subregion p
$A_{q,p}^{(i)}$	q^{th} sub-region that has a parent subregion p from sampling iteration i, and thus $A_{q,p}^{(i)} \subseteq A_p^{(i-1)}$; also note that $A_0^{(0)}$ is the root sub-region and has no parent sub-region, which is defined over $[0, 1]^d$; when the parent sub-region is not important, we simply refer to sub-regions with subscript only (e.g., $A_{q,p}^{(i)}$ instead of $A_{q,p}^{(i)}$)
$B_q^{(i)}$	Sub-region boundaries of $A_q^{(i)}$ (or $A_{q,p}^{(i)}$) which has length $2d$ and specifies the minimum and maximum value of the bounding hyperrectangle in each of the d dimensions; the first d components $(1, \ldots, d)$ specify the lower bounds in each dimension, and the second d components $(d + 1, \ldots, 2d)$ specify the upper bounds in each dimension
$X_{q,p}^{(i)}$	Experimental design points scattered over $A_{q,p}^{(i)}$ based on some sampling method (i.e., Latin hypercube sampling); has size n_q and each design point has length d; $x_{q,k}$ refers to the kth design point in $X_{q,p}^{(i)}$
$Y_{q,p}^{(i)}$	Response (numerical value) of experimental design points $X_{q,p}^{(i)}$ also of size n_q (i.e., responses to design points in $A_{q,p}^{(i)}$); $y_{q,k}$ refers to the kth response of design point $x_{q,k}$
$Z_{q,p}^{(i)}$	Binary variable based on responses $Y_{q,p}^{(i)}$ indicating which responses are low points ω_L and high points ω_H; $\|\omega_L\| = \lfloor \phi n \rfloor$ where $0 < \phi < 1$ indicates the proportion of the n points considered as *low* (here, n is the number of points sampled at any time during sequential experimentation that fall into subregion $A_{q,p}^{(i)}$, and this may change as experimentation proceeds); thus $\|\omega_H\| = n - \|\omega_L\|$; currently using $\phi = 0.8$; note that $z_{q,k}$ is binary indicator corresponding to design point $x_{q,k}$ and response $y_{q,k}$
$\Psi_{q,p}^{(i)}$	Binary convergence values indicating if design runs in $X_{q,p}^{(i)}$ converged based on empirical convergence criteria for reinforcement learning (1 = converged, 0 = not converged); also of length n (i.e., responses to design points in $A_{q,p}^{(i)}$); $\psi_{q,k}$ corresponds to design point $x_{q,k}$, response $y_{q,k}$, and low/high set indicator $z_{q,k}$
$X_{q,p}^*$, $Y_{q,p}^*$, $Z_{q,p}^*$, $\Psi_{q,p}^*$	Variables X, Y, Z, and Ψ from any iteration that fall into subregion q
$T_{q,p}^{(i)}$	CART model that models $Z_{q,p}^{(i)}$ based on $X_{q,p}^{(i)}$ sampled over sub-region $A_{q,p}^{(i)}$

Table 4.1 (continued)

Parameter	Description
θ_{ψ}	Threshold value (proportion, $0 \leq \theta_{\psi} \leq 1$) for a node of the design tree to be considered converged; i.e., a design tree node is considered converged if $\beta_{q,p}^{(i)} = \frac{\sum_k \psi_{q,k}}{\|\Psi_{q,p}^{(i)}\|} \geq \theta_{\psi}$; currently using value of $\theta_{\psi} \sim 0.75$–0.9
ϕ_{prune}	Threshold proportion of convergence of points within design tree node for said node to be removed from further exploration (node is removed if $\beta_{q,p}^{(i)} \leq \phi_{prune}$, currently using $\phi_{prune} = 0.1$)
$N^{(0)}$	Number of initial design points over entire domain $A^{(0)}$
$N^{(i)}$	Number of design points over domain $A^{(i)}$; currently $N^{(i)}$ is equal for all $i \neq 0$
N_{reps}	Number of replications of each design point
N_{iter}	Maximum number of s CART modeling iterations (sequential modeling may terminate prematurely if sub-region nodes have been pruned or have converged)

approach but for the purpose of optimization. Other proportions could be used as well, but it should be noted that the algorithm will progress differently; for example, if a larger proportion of the points was considered to be in the *low* group, the algorithm would find convergent subregions slower. The exact effects of this split proportion should be evaluated under different problems, although this is left for future work.

The subregions of the CART model that are labeled as *low* are then explored in subsequent experimentation iterations. For each *low* CART model leaf q, a new set of design points $X_{q,0}^{(1)}$ are created within the subregion defined by leaf $A_{q,0}^{(1)}$, where this subregion is bounded by the CART model splits. As mentioned, the second subscript value in these variables indicates the parent subregion of the current subregion q. These new design points $X_{q,0}^{(1)}$ are then run through the simulation, and corresponding numerical results $Y_{q,0}^{(1)}$ and result types $\Psi_{q,0}^{(1)}$ are obtained. Design runs (and their numerical results and result types) from $X_0^{(0)}$ that fall with leaf q (call these entities $X_q^{(0)}$, $Y_q^{(0)}$, and $\Psi_q^{(0)}$, respectively) are appended to the runs from $X_{q,0}^{(1)}$ resulting in a set of points $X_{q,0}^*$ and associated responses $Y_{q,0}^*$, *low*/*high* indicator variables $Z_{q,0}^*$, and convergence indicators $\Psi_{q,0}^*$. Design points are appended so that design runs from previous iterations are not merely thrown away. Another CART model $T_{q,0}^{(1)}$ is then constructed that models $Z_{q,0}^*$ as a function of $X_{q,0}^*$. The next iteration proceeds by considering all q leaf nodes from iteration 1 as new parent nodes from iteration 2, sampling within these subregions, obtaining and computing responses, and creating new CART models.

The convergent subregions of the parameter space are of interest and convergent runs usually have lower numerical results Y, and this sequential CART procedure is built on this assumption. Consequently, the leaves of the CART model that are labeled as *low* are of primary interest because these subregions will likely have greater proportions of convergent reinforcement learning runs compared to outside of these subregions. As sequential experimentation proceeds, the proportion of convergent runs $\beta_{q,p}^{(i)}$ within leaves labeled as *low* is likely to increase. The sequential experimentation procedure continues finding and exploring subregions until the proportion of convergent runs with leaves labeled as *low* surpasses some threshold proportion of convergence θ_ψ. These subregions that surpass this threshold convergence proportion are then considered to be convergent subregions, which can then be analyzed or used in further and/or additional experimentation.

This algorithm is presented in Algorithm 2 and Fig. 4.1 shows an example of how this algorithm proceeds in a 2-dimensional setting. Note that the sequential CART experimentation procedure itself could also be viewed as a tree, where parameter space subregions are children of parameter space subregions from parent subregions of previous iterations, and this visualization representation of the experimental process is used in the experiments performed later in this work.

ALGORITHM 2: *Sequential CART*

Require:

1. Initial parameter space $A_0^{(0)}$ defined by bounds $B_0^{(0)}$
2. Convergence threshold θ_ψ

Create $X_0^{(0)}$ by sampling within $A_0^{(0)}$;
Run all $X_0^{(0)} \rightarrow Y_0^{(0)}, \Psi_0^{(0)}, Z_0^{(0)}$;
Create CART model: $T_0^{(0)} : Z_0^{(0)} \sim X_0^{(0)}$;
Extract leaf subregions: $A_{q,0}^{(1)}$ for $q = 1, \ldots, n_{\omega_L}$ w/ prediction ω_L from $T_0^{(0)}$;
Determine bounds $B_{q,0}^{(1)}$ for all $A_{q,0}^{(1)}$;
$i = 1$; $p = 0$; $c = n_{\omega_L}$;
while $i \leq n_{iter}$ **do**
  for all nodes q w/ parent p **do**
    Determine $\beta_{q,p}^{(i)}$ of $A_{q,p}^{(i)}$;
    if $\beta_{q,p}^{(i)} \geq \theta_\psi$ **then**
      Subregion $A_{q,p}^{(i)}$ has converged; do not explore further this subregion;
    else
      Create $X_{q,p}^{(i)}$ by sampling within $B_{q,p}^{(i)}$ from $A_{q,p}^{(i)}$;
      Run all $X_{q,p}^{(i)} \rightarrow Y_{q,p}^{(i)}, \Psi_{q,p}^{(i)}, Z_{q,p}^{(i)}$;
      Create CART model: $T_{q,p}^{(i)} : Z_{q,p}^{*} \sim X_{q,p}^{*}$;
      Extract leaf subregions: $A_{r,q}^{(i+1)}$ for $r = c+1, \ldots, n_{\omega_L}$ w/ prediction ω_L from $T_{q,p}^{(i)}$;
      Determine bounds $B_{r,q}^{(i+1)}$ for all $A_{r,q}^{(i+1)}$;
      $c = c + n_{\omega_L}$;
  $p = p + 1$;
$i = i + 1$;

Return: Convergent subregions $A_{q,p}$

Due to the stochasticity of the problems we are interested in, the responses may vary for a series of replications of the same design point, or some of these replications may not be considered to have converged. In light of this, we consider all responses of design points individually, and we do not aggregate the responses of replicates for the same design points. By *aggregate*, we refer to a summary value that is computed from all replicates of the same design point, where this summary value may be a mean value, the variance of the responses, or even a convergence probability. We do not aggregate responses because the goal of the sequential CART procedure is to find roughly convergent parameter subregions given a reasonable amount of computation time. If computation time were not an issue, aggregating responses might result in smaller convergent parameter subregions that were more accurate with respect to the convergence rate of the design points within those subregions.

The sequential CART algorithm outlined in Algorithm 2 provides a basic overview of the procedures, but there are a few additions that can be made that may improve the performance of the procedure. In particular, such additions would be used to ensure

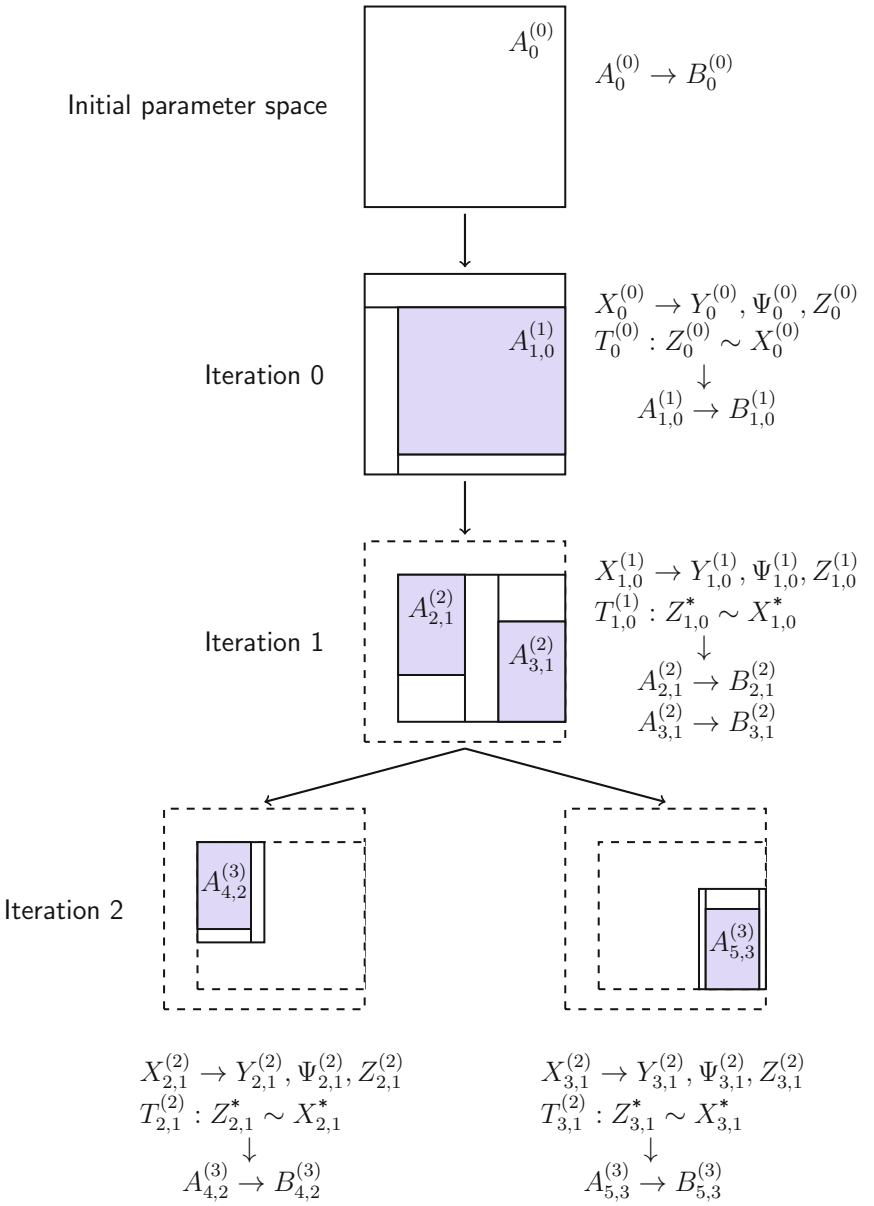


Fig. 4.1 Example of the sequential CART modeling procedure.

that the convergent subregions surpass some quantitative or qualitative metrics. These quality metrics essentially allow for pruning *low* subregions that are deemed to be poor in some respect, which prohibits potentially endless exploration of smaller and smaller subregions.

Just as subregions must surpass some threshold convergence proportion θ_ψ to be considered convergent, subregions may also be pruned for having very few design points or a very low convergence proportion β. As each problem may have a very different underlying structure and experimental designs may vary in size, the threshold values for the minimum number of points for each leaf node and the minimum convergence proportion are domain specific. The CART modeling approach has the ability to produce leaf nodes that may be oddly shaped or sized. Measures of the shape and size of the subregions could be used to further prune subregions from further exploration. Some of these measures include the subregion dimensionality aspect ratio $\alpha_{q,p}^{(i)}$ and the radius $r_{q,p}^{(i)}$ (Eqs. 4.1–4.2). In these equations, b_{j+d} and b_j are the $j + d^{th}$ and j^{th} elements of the subregion boundaries defined by $B_{q,p}^{(i)}$.

$$\alpha_{q,p}^{(i)} = \frac{\max_j \left(b_{j+d} - b_j\right)_{q,p}^{(i)}}{\min_j \left(b_{j+d} - b_j\right)_{q,p}^{(i)}} \tag{4.1}$$

$$r_{q,p}^{(i)} = \frac{1}{d} \sum_{j=1}^{d} \frac{1}{2} \left(b_{j+d} - b_j\right)_{q,p}^{(i)} \tag{4.2}$$

Unlike the design runs in X, which can remain unscaled, the measures mentioned above should be used on scaled parameters such that the original parameter space in each dimension defined by the bounds in $B_0^{(0)}$ ranges over $[0, 1]$ in order for all dimensions to be comparable. These measures would have similar threshold values to θ_ψ. A subregion $A_{q,p}^{(i)}$ may be pruned if $\alpha_{q,p}^{(i)} > \theta_\alpha$ or $r_{q,p}^{(i)} < \theta_r$. The addition of these threshold measures would be placed as if-statements in Algorithm 2 at line 13, similarly to the assessment of $\beta_{q,p}^{(i)}$ on line 11.

4.1.3 Analysis of Sequential CART

The purpose of the sequential CART procedure is to determine convergent parameter subregions in the reinforcement learning problems studied in this work. In each of the reinforcement learning problem domains we study, we use the same procedure to understand this sequential experimental procedure and to analyze the convergent subregions.

We use a novel visualization to show the range and location of the convergent subregions bounds in multiple dimensions. In these figures, groups of lines represent a single variable, where the minimum and maximum values of the original parameter space (over which the original experimental design was created) are shown on the

left and right sides of the figure, respectively. Each thick line represents the extent of a particular parameter for a particular subregion, which is numbered on the sides of the thick lines. Variables are grouped together (rather than each subregion) so that differences in the variable ranges can be easily seen. An example of this figure will be presented later in this section when a 2-dimensional example function is explored using sequential CART.

For each of the convergent subregions, we present some summary statistics, which are computed as follows. We compute the proportion of convergent points from the entire experimental design that fall into the respective subregions (p_{conv}) as well as the total number of points in each subregion. Statistics on the shape and size of the subregions include the dimensionality ratio (Eq. 4.1), the average radius, and the sum of the radii. For these metrics, the convergent parameter subregion boundaries are all normalized over $[0, 1]$ prior to computing all metrics. The dimensionality ratio provides an indication of whether the parameter ranges of the subregion are proportional or not. The average radius is the mean of all radii of the subregions, and the sum of the radii is the total of all of the subregion radii across all parameters, and these metrics are computed from Eq. 4.2.

Regional sensitivity analysis (RSA) (Hornberger and Spear 1981; Saltelli et al. 2004; Ratto et al. 2007) (also called Monte Carlo filtering) is used as a model-free method to investigate the univariate effects of parameters on the binary outcome variable of whether or not a reinforcement learning run had converged. For each parameter, RSA shows the cumulative distributions for what are called *behavioral* and *non-behavioral* groups of points, where in our case, these groups correspond to convergent and non-convergent reinforcement learning runs, respectively. Any differences in the cumulative distributions between the two groups indicates that different ranges of the respective parameter have an effect on learning convergence. This difference can be quantified using a Kolmogorov-Smirnov statistic $KS = \sup_x |F(x|B) - F(x|\bar{B})|$, which is the supremum of the set of distances between the behavioral (i.e., convergent) distribution $F(x|B)$ and the non-behavioral (i.e., non-convergent) distribution $F(x|\bar{B})$. We use this statistic as a quantitative measure to compare distributions between parameters.

4.1.4 Empirical Convergence Criteria

The description of the sequential CART algorithm is based on the notion of convergence for reinforcement learning, which is defined here. Many learning algorithms (e.g., unsupervised or supervised, and some reinforcement learning implementations) often have theoretical convergence guarantees. However, neural network-based reinforcement learning does not have such a guarantee, and learning performance must be quantified and assessed in another manner. Furthermore, reinforcement learning isn't typically analyzed using a large scale analysis as we do here, and performance is often simply assessed by visually or quantitatively assessing a handful of plots that show performance over the course of training. We use a novel

empirical convergence assessment to automatically determine if and when learning has stabilized at an acceptable level for a reinforcement learning run, and which does not require the user to manually assess learning performance. The qualifier *empirical* is used in order to distinguish our convergence assessment from true theoretical convergence, as we do not claim to say anything about theoretical convergence.

A commonly used measure of performance in reinforcement learning is the number of time steps it takes for the agent to reach a specific goal. For many domains, this value is not constant because of either the random dynamics of the environment, or because the initial state of the agent is randomized over some part of the state space of the domain. Thus, this metric, or even a moving average of this metric, may be quite variable even when learning stabilizes. We therefore use an alternative performance measure to assess empirical convergence, which is a moving proportion of the learning runs that terminate by reaching the desired goal. Each episode of a learning run may terminate due to a number of reasons, some of which may be domain-specific, and which may include reaching the goal, reaching a terminating state, using the maximum number of time steps, or having no feasible actions, amongst others. Moving proportions of each of the episode termination types can be computed as follows. Suppose that z_{ij} is a binary variable that indicates that learning episode i terminated by termination type j. The (lagging) moving proportion at learning episode i for termination type j with a moving window of size p_{win} can be computed as $\rho_{ij} = \frac{1}{p_{win}} \sum_{k}^{i} z_{ij}$ where $k = \max(0, i - p_{win} + 1)$ and where this metric is computed for all episode termination types.

The empirical convergence criteria assesses the stability of the moving proportion of reaching the goal. Assessing learning performance stability in this manner is very similar to the notion that shape recognition can be based on conjunctions of local properties (Cho and Dunn 1994), as we are interested in the shape of the learning performance curve having specific local properties. Suppose that reaching the goal is of termination type $j = 1$. For each episode of a learning run $i = p_{win}, \ldots, t_{\max}$, where $t_{\max}$ is the maximum number of time steps allowed in an episode, the following three metrics are computed as:

- Level ($conv_{val}$): which is simply the value of ρ_{i1};
- Range ($conv_{rng}$): which is the range of ρ_{k1} over $k = i - p_{win} + 1, \ldots, i$;
- Slope ($conv_{m}$): which is the slope of ρ_{k1} over $k = i - p_{win} + 1, \ldots, i$.

The size of the moving window p_{win} should be selected based on $t_{\max}$ and is likely domain specific. Reasonable values for the level, range, and slope of ρ_{i1} for most cases could be as follows: the level should be greater than 0.95; the range should be less than 0.01; and the absolute value of the slope should be less than 1×10^{-6}. If all three of these requirements are satisfied at some episode i, the learning run is considered to have converged at this episode. This criteria is used for all domains investigated herein, though a small relaxation is used in the tandem truck backer-upper problem in Chap. 7.

We should note that other metrics could be developed to quantify other aspects of a learning curve. For instance, persistence could be used to assess the stability of learning over a period of time, although the use of a moving window in the computation of the three metrics that are used (level, range, and slope) does assess persistence to some degree. A separate persistence measure was not used primarily because the learning algorithm explored in this work, TD(λ), is known to be unstable such that the agent can *unlearn* and performance can decrease after reaching a high level (Gatti et al. 2011a; Gatti and Embrechts 2012).

4.1.5 Example: 2-D 6-hump Camelback Function

A 2-dimensional function is used to illustrate the sequential CART modeling procedure. The function of interest is the 2-D 6-hump camel back function:

$$y = \left(4 - 2.1x_1^2 + \frac{x_1^4}{3}\right) x_1^2 + x_1 x_2 + \left(-4 + 4x_2^2\right) x_2^2 \tag{4.3}$$

which is normally defined over $x_1 \in [-3, 3]$, $x_2 \in [-2, 2]$. This function is augmented to fit the context for when the sequential CART procedure is applicable for the type of problem of interest. Specifically, this function should be non-deterministic and there should be some *convergence* indicator that is associated with each response value. We therefore modify this function and include an additional Gaussian-distributed zero-mean noise term. Additionally, a *convergence* indicator variable ψ was created based the response value y. The augmented camelback function and the convergence equations are:

$$y = \left(4 - 2.1x_1^2 + \frac{x_1^4}{3}\right) x_1^2 + x_1 x_2 + \left(-4 + 4x_2^2\right) x_2^2 + N(0, 0.2) \tag{4.4}$$

$$\psi = \left[U(0,1) < \left(1 - \frac{1}{1 + e^{-3.5y+2.5}}\right)\right] \tag{4.5}$$

The deterministic version of this function has six minima, four local minima and two global minima. Figure 4.2 shows a contour plot of the deterministic and noisy versions of this function, where the low regions are indicated by lighter colors. The two global minima are equal to -1.0316, which are located at $(x_1, x_2) = (-1.0898, 0.7126)$ and $(0.0898, -0.7126)$. The noisy augmented function has low regions around these locations. Convergence is most likely in these regions as well, and becomes less likely away from these locations as the function value increases. The goal of the sequential CART procedure is to find subregions, defined by variables x_1 and x_2, that have a high proportion of convergent responses (i.e., $\psi = 1$).

Table 4.2 provides a summary of the parameters and settings used in the sequential CART procedure for this example problem. The parameters and settings used for this example and for the domain problems that follow were selected based on initial

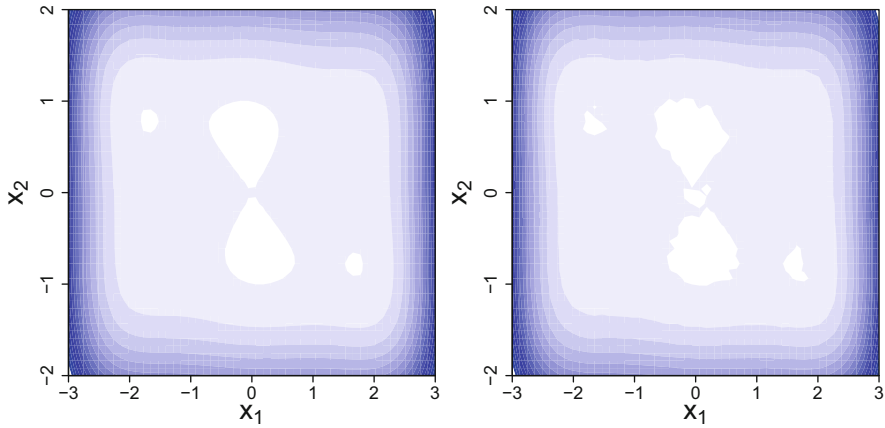


Fig. 4.2 Deterministic (**a**) and noisy (**b**) versions of the 6-hump camelback function.

Table 4.2 Parameter settings for sequential CART modeling and CART models.

Number of variables	2
Initial number of design points	40
Replications per design point	4
Number of sequential CART iterations	7
Number of design points per convergent subregion	20
Sample design method	LHS
Minimum design points per split[a]	$\lfloor 0.10\|\|y\|\| \rfloor$
Minimum design points per leaf[a]	$\lfloor 0.05\|\|y\|\| \rfloor$
Minimum complexity change per split	0.01
Maximum number of surrogates	5
Proportion of low points per leaf (ϕ)	0.8
Convergence proportion threshold (θ_ϕ)	0.75

[a] y is the response vector for the current iteration

testing and development of the sequential CART algorithm in each of the problems. At this point, we do not claim that these settings are ideal, and future work should explore the effects of each of these settings in a variety of domains. However, we do find that most of the settings used in this example problem are able to successfully find convergent parameter subregions.

An initial sample size of 40 points (20 $\times$ number of variables) were sampled over $x_1 \in [-3, 3]$ and $x_2 \in [-2, 2]$ using Latin Hypercube sampling (LHS), with 4 replications at each design point, resulting in a total of 160 initial design points. Sequential CART modeling was run for 7 iterations following the initialization design

run. At each iteration, 20 new points (10 × number of variables) were sampled within each region of the leaves of the CART models labeled as ω_L, again with 4 replications at each design point, and these points were also sampled using LHS. An alternative method for selecting subsequent design points within a subregion could be to choose points based on their proximity to previous points in subregion, though this is likely a more involved optimization problem.

The R package `rpart` (Therneau et al. 2012) was used for the CART modeling. Two CART model parameters were modified from their default values. These include the minimum number of points required for a split in the CART model and the minimum number of points required to be in a leaf node, and these values were set to 10 % and 5 %, respectively, of the total number of points used in the CART model for the current subregion being explored. For all subregions, the proportion of points labeled as ω_L was set to 80 % of the total number of points. Subregions were considered to be converged if their convergence proportion β surpassed 0.75. These settings, or settings that are very similar, seemed to be acceptable for the problems of interest in this work, however they are likely not optimal for all problems. For example, if *successful* design runs are very rare and the true subregion of interest is very small, the minimum number of points in a CART leaf would likely have to be reduced.

Figure 4.3 shows how the sequential CART procedure proceeds from the initialization at iteration 0 through iteration 7 where three convergent subregions are found. These figures show how the subregions become smaller and smaller and narrow in the lower valued and convergent regions of the parameter space.

Table 4.3 shows statistics of the convergent subregions that were found from the sequential CART approach. These tables show the bounding regions for each subregion along each dimension, the number of points that fall into each region, and some of the sizes and shapes of the subregions. Subregion 13 is more rectangular shaped than the other subregions (based on the dimensionality ratio), but it is also much larger than the other regions (based on the average radius and the sum of the radii), and this can be seen in Fig. 4.4 which shows the final convergent subregions outlined in red dotted lines.

Additionally, the parameter ranges of these convergent subregions can be represented visually and compactly as in Fig. 4.5. This figure shows, for each dimension x_1 and x_2 of the parameter space, the extent and location of the parameter ranges of each of the subregions. This plot shows that subregions 13 has the largest domain, whereas subregions 23 and 28 are much smaller. This plot also shows that the convergent subregions fall in quite different and non-adjacent regions of the parameter space.

As the sequential CART procedure is a random process due to the design run sampling, we assessed the variability of the bounds of the convergent subregions for the 2-D camelback function. The sequential CART procedure was run 20 times, and the convergent subregions were manually mapped/paired to the low regions of the of the function. For this part, we labeled the middle region of the function as subregion 1, the upper left low region as subregion 2, and the lower right low region as subregion 3. In some cases, no convergent subregions were found that appropriately match the

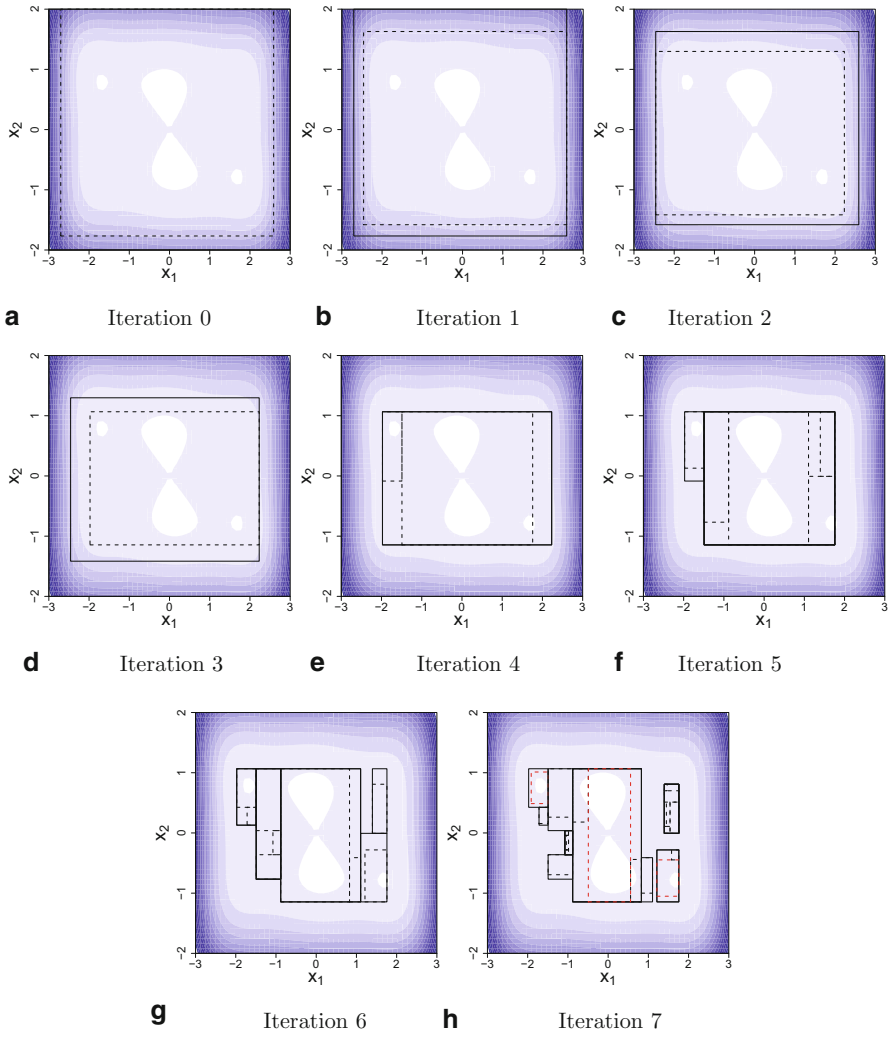


Fig. 4.3 Sequential CART modeling of the 2-D 6-hump camelback function. Each plot shows one iteration of the sequential CART process where subregions progressively become smaller. The underlying deterministic surface is shown in *blue* and *white* on each plot, where *white* indicates lower values and *blue* indicates higher values. In each subfigure, the initial bounding subregion is outlined by a *thick black line* and the subregions of the CART model that correspond to the low points ω_L are outlined in *dotted lines*. The convergent subregions are outlined with *red dotted lines*, and these regions are not explored in subsequent iterations.

low regions of the function. Table 4.4 shows the resulting parameter bounds for these subregions. The count column indicates in how many (of the 20 replications) that particular subregion was found, and we can see that the smaller subregions 2 and 3 are found less often than the large subregion 1. In general, we see that the variability,

Table 4.3 Convergent subregion domains after sequential CART modeling.

Domain #	x_1	x_2
13	[−0.700, 0.811)	[−1.106, 1.018)
23	[−1.950, −1.369)	[0.513, 1.018)
28	[1.241, 1.944)	[−0.982, −0.534)

Domain #	P_{conv}	No. points	Dim. ratio	Min. dim.	Max. dim.	Ave. radius	Sum radii
13	0.812	160	2.108	0.252	0.531	0.196	0.783
23	0.797	148	1.304	0.097	0.126	0.056	0.223
28	0.806	144	1.046	0.112	0.117	0.057	0.229

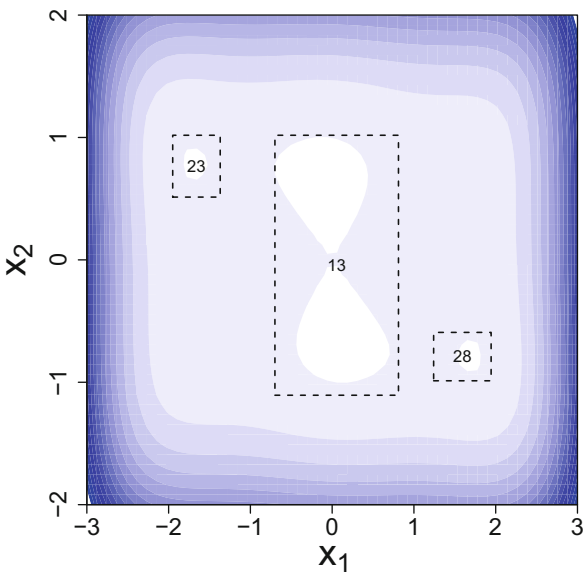


Fig. 4.4 The convergent subregions of the 2D camelback function following sequential CART modeling are enclosed in *dotted lines*. These subregions nicely surround the low regions of the function, shown as *white* in the contour plot.

which is quantified by one standard deviation of the parameter bounds, are relatively small. These numbers are clearly dependent on many parameters of the sequential CART algorithm, including the CART model settings and the experimental designs, though this provides an example of the potential variability of the parameter bounds.

4.2 Kriging Metamodeling

The use of metamodels to replace computationally expensive simulations is becoming increasingly common. Metamodels are essentially models that approximate the response surface of some input-output function. The purpose of metamodels is to

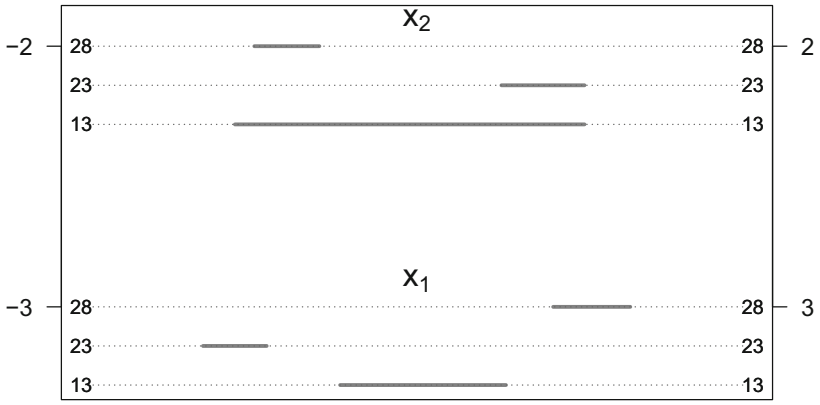


Fig. 4.5 Parameter ranges of the convergent subregions of the 2-D camelback function. For each variable or dimension of the domain, this plot shows the extent and location of each of the convergent subregions.

Table 4.4 Mean convergent subregions boundaries ($\pm$ 1 standard deviation) for the 2-D camelback function for 20 separate replications of the sequential CART procedure.

Subregion	Count	x	y
13 (1)	19	$[-0.654 \pm 0.154, 0.639 \pm 0.0978)$	$[-0.977 \pm 0.108, 1.030 \pm 0.106)$
23 (2)	4	$[-1.900 \pm 0.0591, -1.280 \pm 0.131)$	$[0.460 \pm 0.0476, 0.996 \pm 0.032)$
28 (3)	7	$[1.350 \pm 0.213, 1.870 \pm 0.0701)$	$[-0.949 \pm 0.010, -0.488 \pm 0.099)$

serve as a surrogate model that is computationally quick to evaluate and that has a similar response to the true response of a computationally intensive simulation, such as finite element models, computational fluid dynamics models, partial differential equation solvers that rely on Monte Carlo methods, or even partially converged simulations. Such models may be either deterministic or stochastic, where a deterministic model will output the same response for a set of input parameters, and where a stochastic model will have an output that is variable for a fixed set of input parameters.

In this section, we describe kriging, one type of metamodeling that has been successful in approximating both deterministic and stochastic computer simulations. We begin with an overview of deterministic kriging, and then cover stochastic kriging, which is used later in this work to approximate the response surface of reinforcement learning in two domains.

4.2.1 Kriging

Kriging, also known as spatial correlation modeling, is a type of metamodel that was initially developed for geospatial modeling by Krige (1951) for approximating

geological surfaces, where the goal is to exactly interpolate a 3-dimensional surface; also see the seminal works by Matheron (1963) and Cressie (1993). Sacks et al. (1989) developed this modeling approach specifically for deterministic computer models, which was then extended to random or stochastic models (van Beers and Kleijnen 2003). The kriging formulations herein are described in a number of works, including Qu and Fu (2013), Chen and Kim (2014), Ankenman et al. (2010), Xie et al. (2010), and Roustant et al. (2012a), and the reader is directed to these sources for additional reading, as well as Rasmussen and Williams (2006) for an authoritative review of Gaussian process models.

Let y be a response value to a d-dimensional parameter space $\mathbf{x} = (x_1, \ldots, x_d) \in D \subset \mathbb{R}^d$. Let $\mathbf{X} \in \mathbb{R}^{n \times d}$ denote a set of n (row-wise) parameter vectors $\mathbf{x}$, also called the experimental design in this context, where the ith design point is denoted by $\mathbf{x}^{(i)}$. Similarly, let $\mathbf{y} = \left(y(x^{(1)}), \ldots, y(x^{(n)})\right)'$ denote the corresponding vector of response values for the n design points, where $y(x^{(i)})$ is the response value of the ith design point.

4.2.2 Deterministic Kriging

Deterministic kriging assumes that the response at a point $\mathbf{x}$ has no intrinsic variation (Ankenman et al. 2010) and represents the response surface by:

$$Y(\mathbf{x}) = \mathbf{f}(\mathbf{x})'\boldsymbol{\beta} + M(\mathbf{x}) \tag{4.6}$$

where $\mathbf{f}(\mathbf{x})$ is a vector of trend functions and $M(\mathbf{x})$ is a centered square-integrable process that accounts for spatial correlation or dependence. Simple kriging uses a constant trend value such that $\mathbf{f}(\mathbf{x})' = \mu(\mathbf{x})$, whereas universal kriging uses trend functions that take the form of $\mu(\mathbf{x}) = \sum_{j=1}^{p} \beta_j f_j(\mathbf{x})$, where f_j is a fixed basis function (e.g., first order, second order, etc.) and β_j is an unknown trend coefficient.

The predicted mean response at design point $\mathbf{x}$ and its variance for simple kriging, respectively, are:

$$\hat{Y}(\mathbf{x}) = \mu(\mathbf{x}) + \Sigma_M(\mathbf{x}, \cdot)' \, \Sigma_M^{-1} \, (\mathbf{y} - \boldsymbol{\mu}) \tag{4.7}$$

$$\text{Var}\left(\hat{Y}(\mathbf{x})\right) = \Sigma(\mathbf{x}, \mathbf{x}) - \Sigma_M(\mathbf{x}, \cdot)' \, \Sigma_M^{-1} \, \Sigma_M(\mathbf{x}, \cdot) \tag{4.8}$$

and for universal kriging are:

$$\hat{Y}(\mathbf{x}) = \mathbf{f}(\mathbf{x})'\hat{\boldsymbol{\beta}} + \Sigma_M(\mathbf{x}, \cdot)' \, \Sigma_M^{-1} \, (\mathbf{y} - \mathbf{F}\hat{\boldsymbol{\beta}}) \tag{4.9}$$

$$\begin{aligned} \text{Var}\left(\hat{Y}(\mathbf{x})\right) = {} & \Sigma(\mathbf{x}, \mathbf{x}) - \Sigma_M(\mathbf{x}, \cdot)' \, \Sigma_M^{-1} \, \Sigma_M(\mathbf{x}, \cdot) \\ & + \left(\mathbf{f}(\mathbf{x})' - \Sigma_M(\mathbf{x}, \cdot)' \, \Sigma_M^{-1} \, \mathbf{F}\right)' \left(\mathbf{F}' \, \Sigma_M^{-1} \, \mathbf{F}\right)^{-1} \left(\mathbf{f}(\mathbf{x})' - \Sigma_M(\mathbf{x}, \cdot)' \, \Sigma_M^{-1} \, \mathbf{F}\right) \end{aligned} \tag{4.10}$$

where $\hat{\boldsymbol{\beta}}$ is a vector of trend coefficients, and where Σ_M and $\Sigma_M(\mathbf{x}, \cdot)$ take the form of:

$$\Sigma_M = \Sigma_M(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \text{Cov}(M(\mathbf{x}^{(i)}), M(\mathbf{x}^{(j)})) \quad \forall \quad i, j = 1, \ldots, k \tag{4.11}$$

$$\Sigma_M(\mathbf{x}, \cdot) = \Sigma_M(\mathbf{x}, \mathbf{x}^{(j)}) = \text{Cov}(M(\mathbf{x}), M(\mathbf{x}^{(j)})) \quad \forall \quad j = 1, \ldots, k \tag{4.12}$$

$\Sigma_M \in \mathbb{R}^{k \times k}$ accounts for the pairwise spatial variance between all points, and $\Sigma_M(\mathbf{x}, \cdot) \in \mathbb{R}^{1 \times k}$ accounts for the spatial variance between some point $\mathbf{x}$ and all other points. The covariance matrix Σ_M more explicitly takes the form of:

$$\Sigma_M(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \tau^2 R\left(\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\right) \tag{4.13}$$

where τ^2 is the spatial variance of the process, and where R is a covariance function or kernel that satisfies $\lim_{\|\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\| \to \infty} R(\mathbf{x}^{(i)} - \mathbf{x}^{(j)}) = 0$ and $R(0) = 1$. Additional information on the covariance function follows the description of stochastic kriging.

4.2.3 Stochastic Kriging

Stochastic kriging was developed for cases where the response surface cannot be assumed to be deterministic, such that the response at a design point $\mathbf{x}$ has some intrinsic variance, in addition to the Gaussian process model having built-in extrinsic variance. From above, Σ_M is thought of as extrinsic uncertainty (Ankenman et al. 2010; Qu and Fu 2013) because it is part of the specified (imposed) model and not part of the underlying process. Intrinsic variance is the variability at a design point because of the stochasticity of the underlying process. One way of thinking about extrinsic versus intrinsic uncertainty is using a 3-dimensional case, where there are two variables of interest along a plane, and the response dimension lies perpendicular to this plane. The kriging model would create a response surface connecting design points. Extrinsic uncertainty could be thought of as uncertainty along the response surface or between design points (i.e., along the dependent variable plane). Both of these types of uncertainty are affected by the experimental design and have an effect on model estimation and inference (Ankenman et al. 2010; Chen and Kim 2014). Stochastic kriging has therefore been called a smoothing approach, rather than a interpolative approach as in deterministic kriging (Xie et al. 2010).

Stochastic kriging models a response surface as:

$$Y(\mathbf{x}) = \mathbf{f}(\mathbf{x})'\boldsymbol{\beta} + M(\mathbf{x}) + \epsilon(\mathbf{x}) \tag{4.14}$$

where $\epsilon(\mathbf{x}) \sim \mathcal{N}(0, \rho^2)$ and where ρ^2 is the noise variance, which may be unique for each design point $\mathbf{x}^{(\mathbf{i})}$ for $i = 1, \ldots, k$ where each design point $\mathbf{x}^{(i)}$ has n_i replications. For the design point $\mathbf{x}^{(i)}$, let the mean response $\bar{\mathcal{Y}}(\mathbf{x}^{(i)})$ and the variance of the response $\sigma^2(\mathbf{x}^{(i)})$ be:

$$\bar{\mathcal{Y}}(\mathbf{x}^{(i)}) = \frac{1}{n_i} \sum_{j=1}^{n_i} Y_j(\mathbf{x}^{(i)}) \tag{4.15}$$

$$\sigma^2(\mathbf{x}^{(i)}) = \text{Var}_{j=1,\dots,n_i}(\mathbf{x}^{(i)}) \tag{4.16}$$

Additionally, let Σ_ϵ be a diagonal matrix with entries of the variance of the response values at each design point $\mathbf{x}^{(i)}$ (over all replications $j = 1, \dots, n_i$); i.e., $(\Sigma_\epsilon)_{ii} = \sigma^2(\mathbf{x}^{(i)})$.

The predicted mean response at design point $\mathbf{x}$ for simple kriging and universal kriging, respectively, are:

$$\hat{Y}(\mathbf{x}) = \mu(\mathbf{x}) + \Sigma_M(\mathbf{x}, \cdot)'\Sigma^{-1}(\bar{\mathcal{Y}} - \boldsymbol{\mu}) \tag{4.17}$$

$$\hat{Y}(\mathbf{x}) = \mathbf{f}(\mathbf{x})'\hat{\boldsymbol{\beta}} + \Sigma_M(\mathbf{x}, \cdot)'\Sigma^{-1}(\bar{\mathcal{Y}} - \mathbf{F}\hat{\boldsymbol{\beta}}) \tag{4.18}$$

where $\Sigma = (\Sigma_M + \Sigma_\epsilon)$, $\mathbf{F} = \left(\mathbf{f}\left(\mathbf{x}^{(1)}\right)', \dots, \mathbf{f}\left(\mathbf{x}^{(k)}\right)'\right)'$ and $\mathbf{F} \in \mathbb{R}^{k \times p}$, $\mathbf{f}\left(\mathbf{x}^{(i)}\right) \in \mathbb{R}^p$ is a vector of known functions of $\mathbf{x}^{(i)}$, and $\hat{\boldsymbol{\beta}} \in \mathbb{R}^p$ are unknown parameters to be estimated. The variance of $\hat{Y}(\mathbf{x})$ for simple and universal kriging are similar to those in Eqs. 4.8 and 4.11, respectively, except that Σ_M is replaced with $\Sigma = (\Sigma_M + \Sigma_\epsilon)$.

Unlike deterministic kriging, stochastic kriging is not an exact interpolator, and there is variability around each design point. Thus, the predicted variance at each design point does not go to zero and is inflated over the entire response surface compared to that of deterministic kriging. Furthermore, the predicted variance at each design point does not depend on the response values and only depends on the design points and their variances ρ_i^2.

4.2.4 Covariance Function

The covariance function or kernel R in Eq. 4.13 determines how the response at one design point $\mathbf{x}^{(i)}$ is affected by the responses at all other design points $\mathbf{x}^{(j)}$ for $j \neq i$, and this is the basis and hallmark of the Gaussian process model. This kernel must be positive definite, and the form of which is often selected from a set of kernels that are known to be positive definite; parameters specific to the kernel of choice are then fitted based on the data. Naturally, a Gaussian covariance function is one example that can be used. However, there are a number of other correlation functions that have also proven to be useful in kriging, and these are shown in Table 4.5 and in Figs. 4.6 and 4.7. Most of these functions are based on a single parameter θ (one for each dimension), though the power-exponential function is also based on an exponent b. where $h_q^{(i,j)} = \mathbf{x}_q^{(i)} - \mathbf{x}_q^{(j)}$, and where $\theta > 0$ and $0 < b \leq 2$; see Rasmussen and Williams (2006) for more details. The covariance matrix Σ_M is then created for all points $i, j = 1, \dots, k$ by:

$$\Sigma_M(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = R\left(\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\right) = \sigma^2 \prod_{q=1}^{d} g\left(h_q^{(i,j)}; \theta_q\right) \tag{4.19}$$

Table 4.5 Covariance functions for kriging metamodels.

Gaussian	$g(h) = \exp\left(-\frac{h^2}{2\theta^2}\right)$
Matérn $\nu = 5/2$	$g(h) = \left(1 + \frac{\sqrt{5}\lvert h\rvert}{\theta} + \frac{5\,h^2}{3\theta^2}\right)\exp\left(-\frac{\sqrt{5}\lvert h\rvert}{\theta}\right)$
Matérn $\nu = 3/2$	$g(h) = \left(1 + \frac{\sqrt{3}\lvert h\rvert}{\theta}\right)\exp\left(-\frac{\sqrt{3}\lvert h\rvert}{\theta}\right)$
Exponential	$g(h) = \exp\left(-\frac{\lvert h\rvert}{\theta}\right)$
Power-exponential	$g(h) = \exp\left(-\left(\frac{\lvert h\rvert}{\theta}\right)^b\right)$ where $0 < b \leq 2$

Figure 4.7 compares different covariance kernels for the same value of θ, also known as characteristic length scales, which generally range over $[0, 2]$. The covariance kernel parameters have a physical interpretation such that larger parameters indicate that the response surface changes slowly along the respective variable, and the surface changes quickly for smaller covariance parameters. Depending on the value of θ, the kernels have different shapes. Consequently, the spatial influence of a point at a distance h from another point is also different. There are also some finer differences of these kernels as well. The Gaussian kernel is smooth and has continuous derivatives at all orders. When the Matérn family of covariance kernels is specified by a parameter $\nu = k + 1/2$, where k is a non-negative integer, there is an analytics expression. The Matérn kernels with $\nu = 3/2$ and $\nu = 5/2$ are two popular versions; the Matérn 3/2 kernel is once differentiable and the Matérn 5/2 kernel is

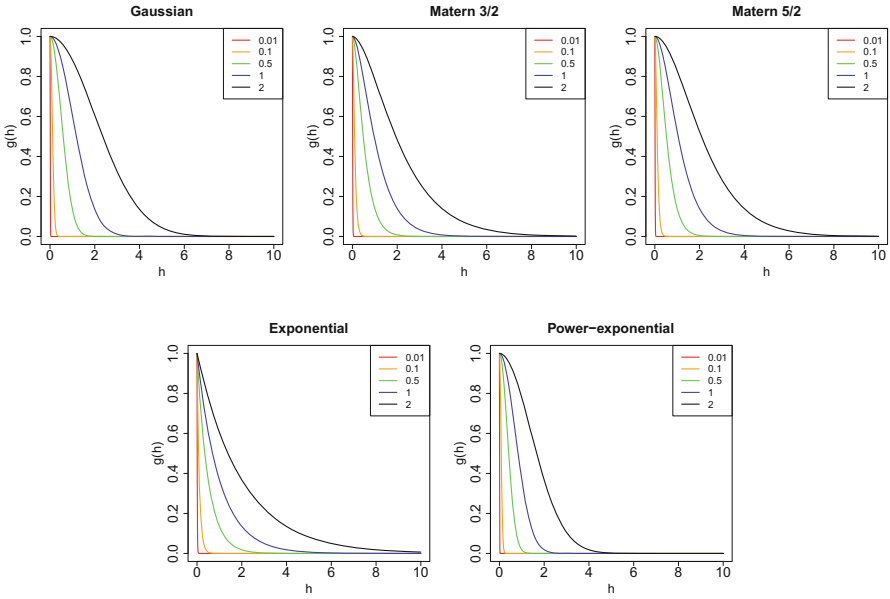


Fig. 4.6 Examples of different covariance kernels used in kriging for a series of parameter values for θ; the power-exponential function used $b = 2$. Each *curve* shows the spatial influence of a point at a distance of h.

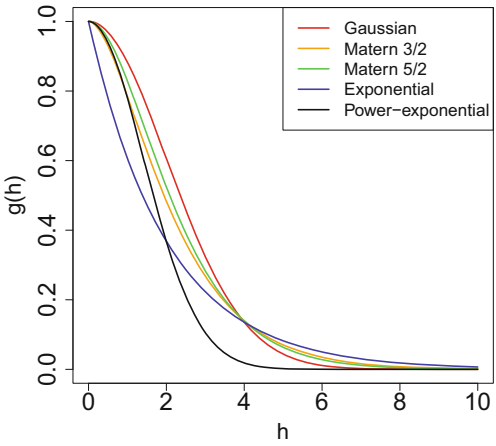


Fig. 4.7 Comparison of different kriging covariance kernels ($\theta = 2$ in all cases).

twice differentiable. Note that when $\nu = 1/2$, this kernel is the exponential kernel, which is simply continuous, and when ν is large, the Matérn kernel is similar to the Gaussian kernel.

The choice of the covariance kernel can be important, as this affects how points influence each other, and this is especially when the trend is known or is assumed to be constant (Roustant et al. 2012a). The selection of the covariance kernel is often based on the differentiability of the response surface, and in most cases this must be assumed. Additionally, the amount of smoothing in stochastic kriging is directly related to the covariance kernel, and hence the parameter(s) of the chosen covariance kernel. Consequently, the predicted response at any point becomes more dependent on the shape of the covariance kernel as that point gets closer to any experimental design point. However, the importance of the choice of the covariance kernel is also related to the end use of the metamodel. If the most accurate metamodel is desired that is to be used to replace a computationally demanding simulation, choosing an appropriate kernel is important. If the model is to be used to generally understand a process, and not for predictive purposes, the choice of a kernel is not as important.

4.2.5 Implementation

When kriging is used in practice, one must estimate the parameters β, Σ_M, and θ. For stochastic applications, τ^2 and Σ_ϵ must also be estimated and it is assumed that the simulation errors $\epsilon_j(\mathbf{x}^{(i)})$ for $j = 1, 2, \ldots, n_i$ (where n_i is the number of replications at design point $\mathbf{x}^{(i)}$) are i.i.d. normal random variables (Chen and Kim 2014).

Whereas many conventional design of experiments require specific experimental designs (i.e., selection of design points), kriging does not have this requirement. Space filling designs are typically used in computer experiments when using kriging, with a popular choice being Latin hypercube sampling (LHS) (Chen and Kim 2014),

and alternatives being quasi-random sequences such as Halton (1960), Hammersley (1960), Sobol' (1967), Faure (1982), and Niederreiter (1992) sequences. The selection of a particular design, however, is difficult and no single design has been found to generally work well, though some studies have provided suggestions based on different experiment characteristics (Sacks et al. 1989; Chen et al. 2006).

Convergent subregions from sequential CART are only regarded as regions that are likely to have convergent reinforcement learning runs, but convergence is not guaranteed at all, and it is possible that design points within the convergent bounds may not converge. Consequently, we require that at least three of the replications at any design point converge for it to be included in the kriging metamodel. This was done because preliminary evaluation showed that including design points that did not converge or that had a low convergence rate resulted in a kriging model that better approximated convergence rather than the response surface of the model. The estimation of the kriging model parameters (trend and covariance kernel parameters) relies on a hybrid algorithm that uses both gradients and a genetic algorithm (the `genoud` solver from the R package `DiceKriging`, which was used for all kriging modeling (Roustant et al. 2012a)).

All independent variables were scaled over $[0, 1]$ prior to creating kriging models. All kriging models used a linear trend and a Matérn $\frac{5}{2}$ covariance kernel. For a kriging model that approximates the response surface of a function with d variables, the model will have $1 + 2 * d$ terms: one for the intercept term, d trend terms, and d covariance kernel parameters. All covariance kernel parameters were allowed to range over $[0, 2]$.

4.2.6 Analysis of Kriging Metamodels

We create kriging metamodels for the mountain car problem and the truck backer-upper problem, and we use the same procedures to analyze each model, which are detailed in this section. These methods include assessing the fit of the metamodel, computing parameter sensitivity indices and using FANOVA (Functional ANOVA) graphs, and plotting response surface projections.

For each of the convergent parameter subregions, we present a number of summary statistics based on the responses of the kriging model experimental design, as well as of the kriging model itself. From the responses of the experimental designs, we present the number and proportion of convergent runs, and the minimum, median, and maximum response values. From the kriging metamodels, we compute the root-mean-square error (RMSE) and the mean absolute error (MAE), as well as R^2 and adjusted–R^2 (R^2_{adj}) of the predicted versus the observed responses, which are computed as:

$$R^2 = 1 - \frac{\sum_i \left(y_i - \hat{y}_i\right)^2}{\sum_i \left(y_i - \bar{y}\right)^2}$$

$$R^2_{adj} = 1 - (1 - R^2)\frac{n-1}{n-d-1}$$

where y_i is the ith response value, $\hat{y}_i$ is the model-predicted response, and $\bar{y}$ is the mean of the response values. The adjusted–R^2 adjusts the value of R^2 by penalizing the addition of variables to the model; in this computation, n is the number of data points in the model, and d is the number of regressors. Additionally, similar metrics are computed from a leave-one-out (LOO) cross validation, which also include RMSE, MAE, and Q^2, which is a LOO-computed version of the R^2. LOO cross validation consisted of removing design points one at a time without re-estimating the kriging model parameters. The computation of Q^2 is similar to that of R^2, except that Q is used to indicate this metric was computed from the LOO analysis, and that in this computation, the predicted values $\hat{y}_i$ are computed from a model that does not contain observation i. Due to the local dependence on each data point of the kriging model, the removal of a single data point can severely change the model, and this is especially so at the location of said data point. It is therefore not surprising if the values of Q^2 is much lower than the values of R^2, and that the RMSE and MAE values are greater for LOO. We use the R^2 and related measures because they assess the linearity of the predictions relative to the true responses while also considering the prediction errors; the commonly used alternative measure of Pearson's correlation coefficient merely assesses the linearity of predicted points relative to the true response values.

We use a series of global sensitivity indices (Sobol' 2001; Saltelli et al. 2004) to assess the importance of each variable. Global sensitivity analysis decomposes the variance of the response (i.e., mean number of time steps to reach the goal) and apportions it to the input parameters. This type of sensitivity analysis differs from a conventional local sensitivity analysis that is gradient-based. In global sensitivity analysis, parameters that have greater sensitivity indices have a greater effect on the variation in the response. We provide a brief review of how Sobol' indices are computed, from which many sensitivity indices are based on similar computations, although we direct the reader to the literature, namely Sobol' (2001) and Saltelli et al. (2004), for a more comprehensive review.

Suppose that $f(\cdot)$ is a square-integrable function that is defined over the hypercube $[0, 1]^d$, where d is the dimensionality of the function. The function can be represented as a sum of functions:

$$f(\mathbf{X}) = f_0 + \sum_{i=1}^{d} f_i(X_i) + \sum_{i<j}^{d} f_{ij}(X_i, X_j) + \ldots + f_{12\ldots d}(\mathbf{X})$$

This decomposition is called an ANOVA decomposition if:

$$\int_0^1 f_{i_1\ldots i_s}\left(x_{i_1}, \ldots, x_{i_s}\right) dx_{i_k} = 0,\ 1 \le k \le s,\ \{i_1, \ldots, i_s\} \subseteq \{1, \ldots, d\}$$

The variance of the output $Y = f(\mathbf{X})$, where $\mathbf{X} = (X_i, \ldots, X_d)$ with mutually independent variables, can be decomposed as a functional decomposition, also known

as a functional ANOVA:

$$\text{Var}\,(Y) = \sum_{i=1}^{d} D_i(Y) + \sum_{i<j}^{d} D_{ij}(Y) + \ldots + D_{12\ldots d}(Y)$$

where $D_i(Y) = \text{Var}[\mathbb{E}(Y|X_i)]$, $D_{ij}(Y) = \text{Var}[\mathbb{E}(Y|X_i, X_j)] - D_i(Y) - D_j(Y)$, and so on for higher order terms. Sobol' indices can then be computed as:

$$S_i = \frac{D_i(Y)}{\text{Var}(Y)},\ S_{ij} = \frac{D_{ij}(Y)}{\text{Var}(Y)},\ \ldots$$

which apportion the variance of the output Y to each input or combination of inputs.

Rather than relying on a single sensitivity index, we use three sensitivity indices that are computed using slightly different methods, and we then take a consensus of these indices. These sensitivity measures include Fourier amplitude sensitivity testing (FAST) (Saltelli et al. 1999), an asymptotically efficient approach to estimating Sobol' indices (Eff) (Monod et al. 2006), and Jansen's approach for estimating Sobol' indices (Jansen 1999). Each of these approaches estimates the first order and total order global sensitivity indices using slightly different methods. Only the first order sensitivity estimates are analyzed here because the total order indices are more variable, and we instead use FANOVA graphs to understand low order interactions between variables. All sensitivity indices were computed with the R package `sensitivity` (Pujol et al. 2012).

FANOVA graphs are a visual tool for easily understanding the main effects and interaction structure among variables of a function. We provide a brief overview of FANOVA graphs and interested readers are directed to Fruth et al. (2013) for a more comprehensive description. The variables are presented as a connected graph, where the size of the nodes represent the magnitude of their main effects, and the size of the edges between variables represent the magnitude of their pairwise interactions. The main effects or influence of each variable are estimated by first-order Sobol' indices, and the interaction among variables, also called the total interaction index (TII) (Fruth et al. 2013), is computed from the Sobol' decomposition as the sum of all Sobol' indices which contain both variables i and j:

$$\mathcal{D}_{ij} := \sum_{I \supseteq \{i,j\}} D_I$$

All computations for the FANOVA graphs use the R package `fanovaGraph` (Fruth et al. 2013).

The experiments in this work include more than two variables, and thus visualizing their response surfaces is not possible. In lieu of this, we use a method that is used for structural reliability modeling that projects the probability of excursion (i.e., exceeding some threshold value) onto pairs of variables. In these plots, we use the median response value as the threshold value. These figures are very useful for gaining insight to the shape of the response function and often are intuitively consistent with the sensitivity indices.

References

Ankenman, B., Nelson, B. L., & Staum, J. (2010). Stochastic kriging for simulation metamodeling. *Operations Research*, 58(2), 371–382.

Breiman, L. (2001). Random forestss. *Machine Learning*, 45(1), 5–32.

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and Regression Trees*. New York, NY: Chapman & Hall.

Chen, X. & Kim, K. (2014). Stochastic kriging with biased sample estimates. *ACM Transactions on Modeling and Computer Simulation*, 24(2). doi: 10.1145/2567893

Chen, V. C. P., Tsui, K.-L., Barton, R. R., & Mechesheimer, M. (2006). A review on design, modeling and applications of computer experiments. *IIE Transactions*, 38(4), 273–291.

Cho, K. & Dunn, S. M. (1994). Learning shape classes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(9), 882–888.

Cressie, N. A. C. (1993). *Statistics for Spatial Data (2nd edition)*. New York, NY: Wiley.

Faure, H. (1982). Discrepancy of sequences associated with a number system (in dimension s). *Acta Arithmetica*, 41(4), 337–351.

Fruth, J., Muehlenstaedt, T., & Roustant, O. (2013). *fanovaGraph: Building Kriging models from FANOVA graphs* (Manual for R package `fanovaGraph`, version 1.4.7). Retrieved from http://cran.r-project.org/web/packages/fanovaGraph/ fanovaGraph.pdf.

Gatti, C. J. & Embrechts, M. J. (2012). Reinforcement learning with neural networks: Tricks of the trade. In Georgieva, P., Mihayolva, L., & Jain, L. (Eds.), *Advances in Intelligent Signal Processing and Data Mining* (pp. 275–310). New York, NY: Springer-Verlag.

Gatti, C. J., Embrechts, M. J., & Linton, J. D. (2011a). Parameter settings of reinforcement learning for the game of Chung Toi. In *Proceedings of the 2011 IEEE International Conference on Systems, Man, and Cybernetics (SMC 2011), Anchorage, AK, 9–12 October* (pp. 3530–3535). doi: 10.1109/ICSMC.2011.6084216

Halton, J. (1960). On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. *Numerische Mathematik*, 2(1), 84–90.

Hammersley, J. M. (1960). Monte carlo methods for solving multivariate problems. *Annals of the New York Academy of Sciences*, 86, 844–874.

Hornberger, G. M. & Spear, R. C. (1981). An approach to the preliminary analysis of environmental systems. *Journal of Environmental Management*, 12, 7–18.

Jansen, M. J. W. (1999). Analysis of variance designs for model output. *Computational Physics Communications*, 117(1), 35–43.

Krige, D. G. (1951). A statistical approach to some basic mine valuation problems on the witwatersrand. *Journal of the Chemical, Metallurgical and Mining Society of South Africa*, 52(6), 119–139.

Matheron, G. (1963). Principles of geostatistics. *Economic Geology*, 58(8), 1246–1266.

Monod, H., Naud, C., & Makowski, D. (2006). Uncertainty and sensitivity analysis for crop models. In *Working with Dynamic Crop Models: Evaluation, Analysis, Parameterization, and Applications*. Amsterdam, Netherlands: Elsevier.

Niederreiter, H. (1992). *Random Number Generation and Quasi-Monte Carlo Methods*. Philadelphia: SIAM.

Pujol, G., Iooss, B., & Janon, A. (2012). *sensitivity: Sensitivity Analysis* (Manual for R package `sensitivity`, version 1.8-2). Retrieved from http://cran.r-project.org/web/packages/sensitivity/sensitivity.pdf.

Qu, H. & Fu, M. C. (2013). Gradient extrapolated stochastic kriging. *ACM Transactions on Modeling and Computer Simulation*, 9(4). doi: 10.1145/0000000. 0000000

Rasmussen, C. & Williams, C. (2006). *Gaussian Processes for Machine Learning*. Cambridge, MA: MIT Press.

Ratto, M., Pagano, A., & Young, P. (2007). *Factor mapping and metamodeling* (Technical Report EUR 21878 EN - 2007, European Commission, Joint Research Centre). Retrieved from http://publications.jrc.ec.europa.eu/repository/bitstream/111111111/13310/1/ reqno_jrc37692_ eur 21878 - factor mapping and metamodelling[2].pdf

Robertson, B. L., Price, C. J., & Reale, M. (2013). CARTopt: A random search method for nonsmooth unconstrained optimization. *Computational Optimization and Applications*, 56(2), 291–315.

Roustant, O., Ginsbourger, D., & Deville, Y. (2012a). DiceKriging, DiceOptim: Two R packages for the analysis of computer experiments by kriging-based metamodeling and optimization. *Journal of Statistical Software*, 51(1), 1–55.

Sacks, J., Welch, W. J., Mitchell, T. J., & Wynn, H. P. (1989). Design and analysis of computer experiments. *Statistical Science*, 4(4), 409–423.

Saltelli, A., Tarantola, S., & Chan, K. P.-S. (1999). A quantitative model-independent method for global sensitivity analysis of model output. *Technometrics*, 41(1), 39–56.

Saltelli, A., Tarantola, S., Campolongo, F., & Ratto, M. (2004). *Sensitivity Analysis in Practice*. Hoboken, NJ: Wiley.

Sobol', I. M. (1967). On the distribution of points in a cube and the approximate evaluation of integrals. *U.S.S.R. Computational Mathematics and Mathematical Physics*, 7(4), 86–112.

Sobol', I. M. (2001). Global sensitivity indices for nonlinear mathematical models and their Monte Carlo estimates. *Mathematics and Computers in Simulation*, 55(1–3), 271–280.

Therneau, T., Atkinson, B., & Ripley, B. (2012). *rpart: Recursive Partitioning and Regression Trees* (Manual for R package `rpart`, version 4.1-8). Retrieved from http://cran.r-project.org/web/packages/rpart/rpart.pdf.

van Beers, W. & Kleijnen, J. P. C. (2003). Kriging for interpolation in random simulations. *Journal of the Operational Research Society*, 54(3), 2233–2241.

Xie, W., Nelson, B., & Staum, J. (2010). The influence of correlation function on stochastic kriging metamodels. In *Proceedings of the 2010 Winter Simulation Conference (WSC), Baltimore, MD, 5–8 December* (pp. 1067–1078). doi: 10.1109/WSC.2010.5679083

Chapter 5
The Mountain Car Problem

The mountain car problem (Moore 1990) is commonly used as a benchmark reinforcement learning problem to evaluate learning algorithms. The problem places a car in a valley, where the goal is to get the car to drive out of the valley (Fig. 5.1). The car's engine is not powerful enough for it to drive out of the valley, and the car must instead build up momentum by successively driving up opposing sides of the valley. The state ($\mathbf{x} = [x, \dot{x}]$) of the car is defined by its position $x \in [-1.2, 0.5]$ and its velocity $\dot{x} \in [-1.5, 1.5]$, and the goal is located at $x = 0.5$. At the beginning of each episode, the x is uniformly randomly sampled from $[-1.2, 0.5]$ and $\dot{x} = 0$.

We define the current position and velocity of the car by x and $\dot{x}$, respectively, and the position and velocity of the car at the next time step by x' and $\dot{x}'$, respectively. The car's dynamics follow:

$$x' = x + \Delta t \dot{x}$$
$$\dot{x}' = \dot{x} + \Delta t \left(-9.8\, m\, \cos(3x) + \tfrac{f}{m}\, a - \mu\, \dot{x}\right)$$

where $\Delta t = 0.01$ is the time step, $m = 0.02$ is the car's mass, $f = 0.2$ is the engine force, and $\mu = 0.5$ is a friction coefficient. The variable a represents the action taken by the agent, where $a = -1$ for driving backwards, $a = 0$ for neutral, and $a = 1$ for driving forwards. In other words, at any discrete time step, the driver gets to choose from these three actions.

5.1 Reinforcement Learning Implementation

A three-layered neural network was used to learn the mountain car problem using the temporal difference algorithm TD(λ) (Sect. 2.2.3). The input to the network was the state of the car defined by its position and velocity, and the output of the network attempted to approximate the value function $V(s, a)$, or the utility of taking action a when in state s at time t. Consequently, the network had two input nodes and three output nodes. The number of hidden nodes was varied during experimentation and will be discussed later. The hidden layer of the network used a hyperbolic tangent

C. Gatti, *Design of Experiments for Reinforcement Learning*,
Springer Theses, DOI 10.1007/978-3-319-12197-0_5

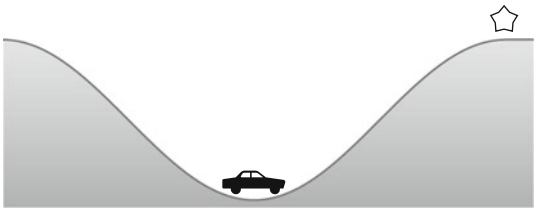


Fig. 5.1 Conceptual representation of the mountain car problem. The state of the car is defined by its position and velocity, and the goal of the problem is to select actions that build momentum so that the car can drive out of the right side of the valley.

Table 5.1 Domain characteristics for the mountain car problem.

Initial conditions	$x = U[-1.2, 0.5]$
	$\dot{x} = 0$
Actions	3: $[0.0, \pm 1.0]$
Reward	Achieve goal: $+1$
Reward function per time step	$r = x$
Number of episodes	2000
Number of time steps per episode (T_{max})	500
Performance time window	300
$conv_{val}$	0.5
$conv_{rng}$	0.005
$conv_m$	1×10^{-4}

transfer function and the output layer used a linear transfer function. The input and hidden layers both had bias nodes with constant values of + 1. Network weights were initialized by uniformly sampling from $[-0.1, 0.1]$.

Additional characteristics of the problem are provided in Table 5.1. The initial position of the car at the beginning of each episode was uniformly sampled over $[-1.2, 0.5]$ and the velocity of the car was zero. When the car reached the goal ($x = 0.5$), a reward of + 1 was provided; at all other time steps, feedback (i.e., a penalty) was provided that was equal to the position of the car x. This reward function provides positive reinforcement when the car is closer to the goal and negative reinforcement when the car is further from the goal. Episodes could terminate in two ways: the car could either reach the goal or the maximum number of time steps could be reached. The empirical convergence criteria described in Sect. 4.1.4 was used to automatically determine if learning had converged in each design run and the thresholds used for the convergence assessment are also shown in Table 5.1.

Table 5.2 Variables and their associated ranges used in sequential CART for the mountain car problem.

Variable	Description	Range	RL component
α_{mag}	Base (input–hidden layer) learning rate	[0.0005, 0.01]	Neural network
α_{ratio}	Learning rate ratio	[1.0, 8.0]	Neural network
λ	Temporal discount factor	[0.0, 1.0]	TD(λ) algorithm
γ	Next-state discount factor	[0.9, 1.0]	TD(λ) algorithm
ϵ	P(action exploitation)	[0.6, 1.0]	TD(λ) algorithm

5.2 Sequential CART

Sequential CART modeling was used to identify convergent parameter subregions. Five parameters were studied in this procedure, with two related to the neural network and three related to the TD(λ) algorithm (Table 5.2). The ranges of these parameters generally include what is recommended for reinforcement learning or neural networks, though these ranges were expanded in order to find potentially useful parameter subregions that are outside of what is typically recommended.

Both the magnitude of the learning rates and the ratio of the learning rates between the input–hidden and hidden–output layers are studied here. Generally, only the magnitude of the learning rates is given consideration and a single learning rate is used for the entire network. However, the weight update equations for the TD(λ) algorithm (and the back-propagation algorithm) naturally result in the gradient of the input–hidden layer having a smaller magnitude than that of the hidden–output layer, and there is evidence supporting the use of learning rates that are scaled between layers (LeCun et al. 1998; Embrechts et al. 2010). We are therefore interested to see if the ratio of the learning rates has an effect on reinforcement learning and include this parameter in this study. The learning rates of the network were set based on the variables α_{mag}, a base learning rate between the input and hidden layer of the network, and α_{ratio}, the ratio of the learning rates for each subsequent layer of the network with respect to the input–hidden layer. For example, in a three–layer neural network, the learning rates of the input–hidden layer (α_{hi}) and the hidden–output layer (α_{oh}), respectively, would be set as $\alpha_{hi} = \alpha_{mag}$ and $\alpha_{oh} = \frac{\alpha_{mag}}{\alpha_{ratio}}$. The parameters λ, γ, and ϵ are all related to the TD(λ) algorithm (Sect. 2.2.3), where λ is the temporal discount factor, γ is a next-state discount factor, and ϵ is an action selection exploration-exploitation trade-off parameter.

The parameters and settings used in the sequential CART algorithm are listed in Table 5.3. These settings were chosen based on preliminary experimentation and use with the sequential CART process. The initial, or seed, experimental run consisted of 60 design points with 3 replication each, and each subsequent sub-experiment in the sequential CART procedure used 20 new design points, also with 3 replications each. All designs were generated using Latin hypercube sampling. The proportion of points within each design labeled as *low* was 0.80, and the required convergence rate for any *low* leaf node was 90 %.

Table 5.3 Parameter settings for sequential CART modeling and CART models.

Description	Value
Number of variables	5
Initial number of design points	60
Replications per design point	3
Number of sequential CART iterations	4
Number of additional design points per *low*-leaf	20
Sample design method	Latin hypercube
Minimum design points per split[a]	$\lfloor 0.1\|\|y\|\| \rfloor$
Minimum design points per leaf[a]	$\lfloor 0.05\|\|y\|\| \rfloor$
Minimum complexity change per split	0.01
Maximum number of surrogates	5
Proportion of low points per leaf (ϕ)	0.8
Convergence proportion threshold (θ_ϕ)	0.9
Minimum design points per convergent domain	8

[a] $\|\|y\|\|$ is the length of the response vector for the current iteration

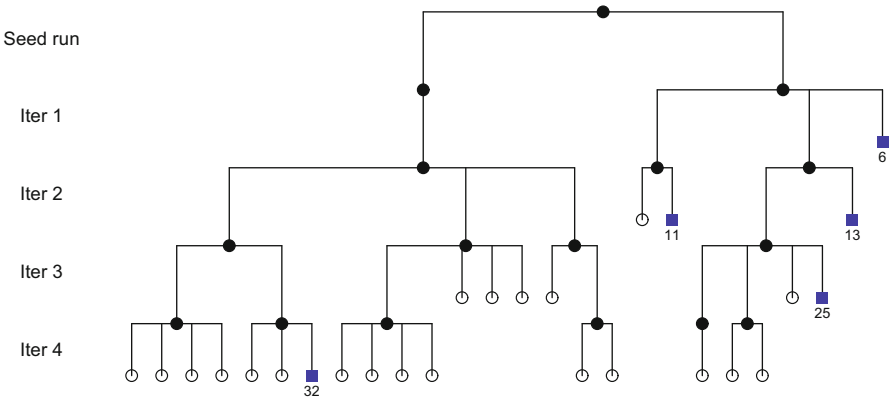


Fig. 5.2 Sequential CART process for the mountain car problem. Parameter subregions that are candidates for further experimentation are shown as *black circles*, subregions that are pruned from further experimentation are shown as *open circles*, and convergent subregions are shown as *blue squares*.

5.2.1 Convergent Subregions

The progression of the sequential CART procedure is shown in Fig. 5.2. Note that this figure is not a single CART tree from any sub-experiment in the sequential CART process. This figure provides a global overview of the entire sequential CART procedure for the four iterations of the sequential experiment. Each node in the tree corresponds to one parameter subregion that is explored using its own experimental

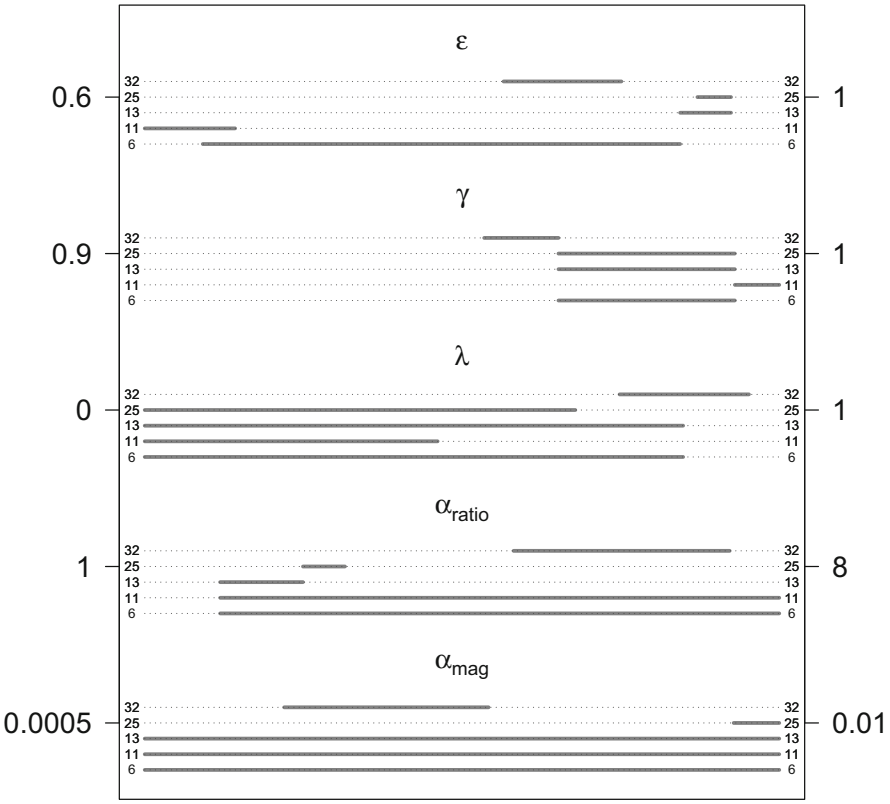


Fig. 5.3 Parameter ranges for the convergent subregions for the mountain car problem. Each parameter explore is shown as a group of *horizontal bars*. Each convergent subregion is specified by a small number within these blocks of horizontal bars. The bounds of the initial parameter ranges are show on the *left* and *right* sides of the figure. The horizontal bars for each parameter and subregion indicate the size and location of the respective parameter.

design based on the respective parameter bounds. More specifically, each node in the tree corresponds to a *low* region of points from the sequential CART algorithm; *high* regions are not further explored and are discarded in this algorithm, and these nodes are therefore not shown. Nodes that are lower in the tree in this figure represent smaller parameter subregions than nodes higher in the tree. When parameter subregions (i.e., nodes) have a convergence proportion greater than $\theta_\phi = 90\,\%$, the node is considered to have converged and further experimentation of that subregion is terminated. All nodes in the experimental tree are assigned a number based on when they were explored in the large experimental scheme, such that lower numbered nodes were explored earlier and larger numbered nodes are explored later in the experiment.

We further screened the convergent domains to have them contain at least 20 design points and have a dimensionality ratio of at most 20, and this resulted in five convergent subregions, shown as blue squares in Fig. 5.2. Figure 5.3 shows the

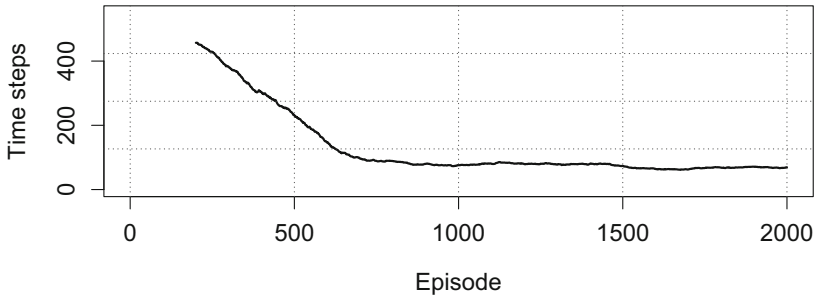


Fig. 5.4 Example performance from a sample in subregion 11, with parameters: $\alpha_{mag} = 1.28 \times 10^{-3}$, $\alpha_{ratio} = 5.212$, $\lambda = 0.091$, $\gamma = 0.999$, $\epsilon = 0.629$. This plot shows the 300-time step moving average of the number of time steps to the goal.

Table 5.4 Summary statistics for the convergent subregions of the mountain car problem.

Subregion #	Iteration	p_{conv}	No. points	Dim. ratio	Ave. radius	Sum radii
6	1	0.967	60	3.597	0.376	3.760
11	2	0.917	36	14.286	0.256	2.555
13	2	0.917	24	12.461	0.234	2.337
25	3	0.967	30	12.804	0.115	1.148
32	4	0.930	57	2.911	0.117	1.169

location and extent of each of the parameters for each of the convergent subregions. For example, convergent subregion 6 has ϵ in a large part of the middle of the parameter range explore, γ should be on the higher end of the interval [0.9, 1], λ and α_{ratio} can range almost over their entire ranges, and α_{mag} can be anywhere in the parameter range explored. An example of a convergent reinforcement learning run from subregion 11 is shown in Fig. 5.4, which shows the moving average of the number of time steps to reach the goal over the 2000 learning episodes.

Summary statistics on each of the convergent subregions are provided in Table 5.4, and numerical values of the subregion boundaries are provided in Table C.1 in the Appendix. The summary statistics include at what iteration the convergent subregion was found, the proportion and the number of convergent points within that subregion, the dimensionality ratio, the average radius of the subregion, and the sum of the radii of the subregion. The dimensionality ratio provides an indication of the proportionality of the size of the subregion bounds, and the measures based on the radii of the subregions provide an indication of the size of the subregions. We see that subregions 6 and 32 are more proportional than the other subregions, and that the size of the subregions decreases as they are found in later experimental iterations.

Figure 5.5 shows plots resulting from regional sensitivity analysis (RSA) based on the convergence indicator from all samples from the sequential CART process. These plots provide an indication of which parameters have an effect on convergence. For each parameter, the cumulative distributions of convergent and non-convergent

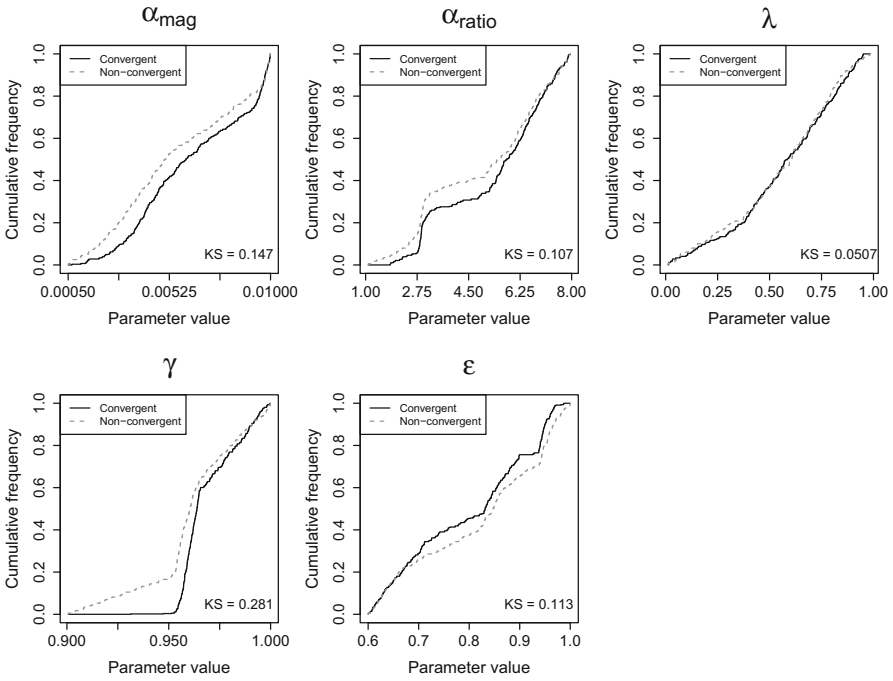


Fig. 5.5 Regional sensitivity analysis based on convergence of all experimental runs for the mountain car problem.

design points are plotted against the parameter values. Parameters that have small effects on convergence have cumulative distributions that are close together, and parameters that have large effects on convergence have distributions that are far apart and have different shapes. The difference in the distributions is quantified by a Kolmogorov-Smirnov (KS) statistics, which measures the maximum vertical distance between two distributions. All parameters except for γ seem to have a relatively small effect on convergence, and this is supported by the wide ranges of the parameter bounds for these parameters in Fig. 5.3. However, γ clearly has different convergent and non-convergent distributions, although the ranges of this parameter in Fig. 5.3 are completely on the higher end of the parameter space.

5.3 Response Surface Metamodeling

For each of the five convergent parameter subregions found using sequential CART, we model the response surface of the number of time steps for the car to get out of the valley. More specifically, we use a moving average of the number of time steps to the goal because the initial state of the car in the domain is randomized for

Table 5.5 Parameter settings for the experimental design and kriging metamodeling for the mountain car problem.

Description	Value
Number of variables (d)	5
Number of design points	300 ($60d$)
Replications per design point	5
Minimum number of convergent replications	3
Kriging model form	Linear (universal kriging)
Covariance kernel	Matérn 5/2
Covariance parameter range	[0, 2]

each learning attempt. Parameters and settings for the kriging experimental design and model are provided in Table 5.5. For each of the convergent subregions, we use an experimental design consisting of 300 design points, with 5 replications at each design point, created using Latin Hypercube sampling (LHS) from the subregion bounds corresponding to those listed in Table 5.4.

Table 5.6 shows summary statistics and kriging model metrics for each of the experiments for the convergent subregions, and model parameters for each of the kriging models are provided in the Appendix in Table C.2. In general, the ranges of the responses of each of the subregions are similar, with the exception of subregion 11, which seems to generally have larger performance values. Based on the R^2 and Q^2, the metamodels for subregions 6 and 11 seem to fit the response surface well, and these metamodels are also created from the largest number of points. The other metamodels for subregions 13, 25, and 32 may need more convergent design points to improve the model fit. As the sequential CART procedure is a random algorithm because of the experimental designs (which affects the resulting CART models), the subregions defined as convergent may be not be as convergent as thought, especially with a small number of design points, and this is shown by the low convergence proportions. This could be due to experimental design sampling such that, for any iteration of the sequential CART algorithm, the number of design points that fall within a convergent subregion is relatively small compared to the total number used at that iteration.

The sensitivity analysis (Fig. 5.6) shows that there is no consistency for the parameter sensitivities across the five convergent subregions (see Sect. 4.2.6 for a description of the sensitivity indices). We also see that some subregions have a single and very dominant parameter (subregions 6 and 11), whereas other subregions have multiple parameters that have a great effect on the response (subregions 13, 25, and 32). These results suggest that the shape of the response surface in different regions of the parameter space is quite different, and what might be an important parameter in one subregion may not be important in another. We also see that the three sensitivity indices that we use are generally consistent, especially when parameters are ranked based on their sensitivities.

Table 5.6 Kriging metamodel statistics for the mountain car problem.

Subregion #	N	p_{conv}	y_{min}	y_{med}	y_{max}	RMSE	MAE	RMSE_{loo}	MAE_{loo}	R^2	R^2_{adj}	Q^2
6	123	0.821	36.8	48.1	78.8	3.15	2.46	4.03	3.13	0.894	0.889	0.826
11	132	0.847	50.6	59.9	81.7	3.39	2.63	3.78	2.92	0.762	0.753	0.705
13	89	0.470	34.2	40.8	49.0	2.66	2.14	3.06	2.50	0.382	0.345	0.183
25	106	0.703	32.9	40	46.8	2.72	2.17	2.95	2.38	0.294	0.259	0.166
32	120	0.777	36	43.7	56.1	2.63	2.02	3.22	2.53	0.576	0.557	0.364

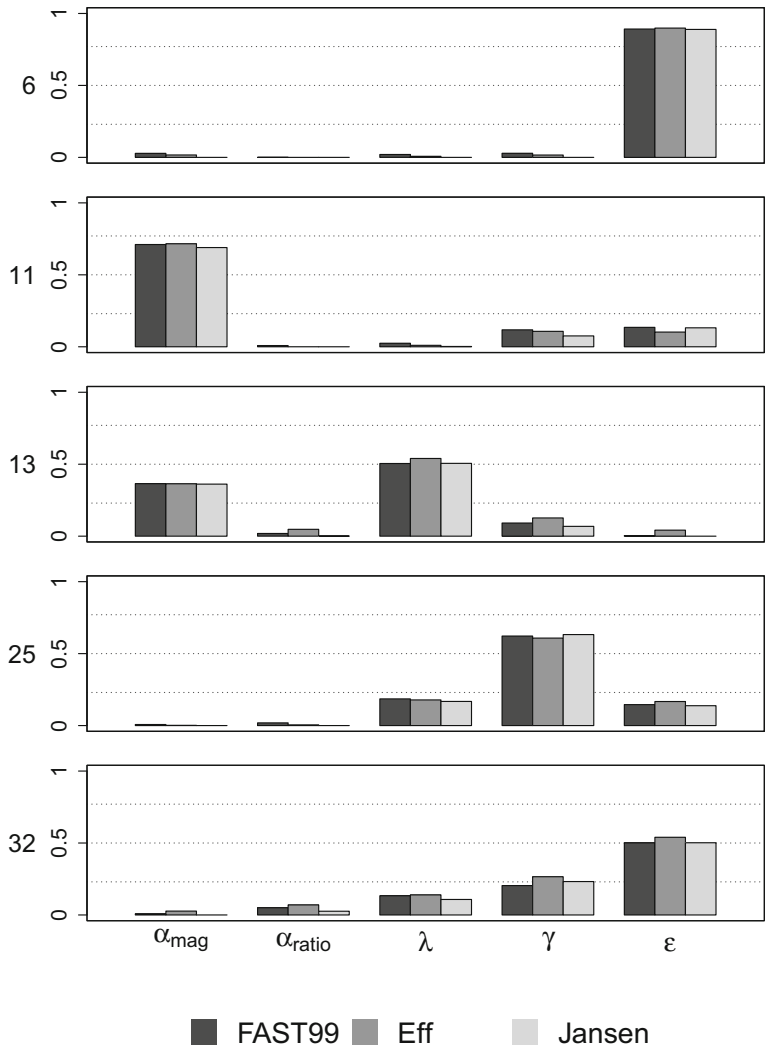


Fig. 5.6 First-order sensitivity indices for the mountain car domain. The subregion number is shown on the *left* side of the plots.

In addition to the first-order sensitivity indices, we use functional ANOVA (FANOVA) graphs to assess two-way interaction effects between the parameters. Figure 5.7 shows the FANOVA decomposition, where each node represents one parameter, the size of each node is scaled to the first-order sensitivity index, and the width of the line between nodes is scaled to the size of the interaction between pairs of parameters. It should be noted that the size of the nodes and the width of the interaction lines should not be compared across subregions, as these are scaled to the

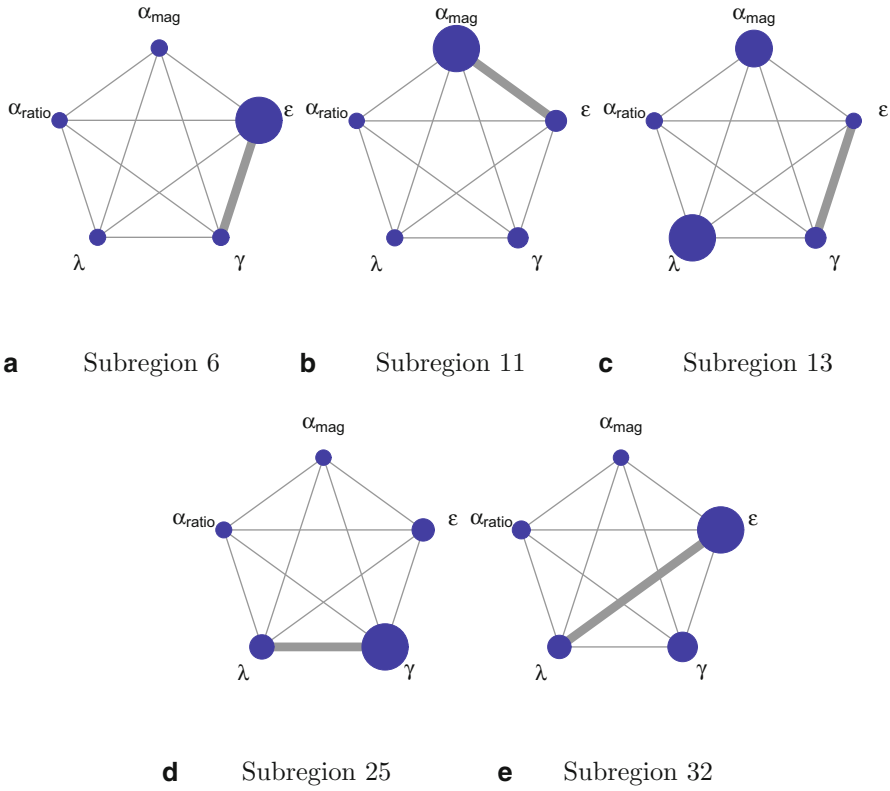
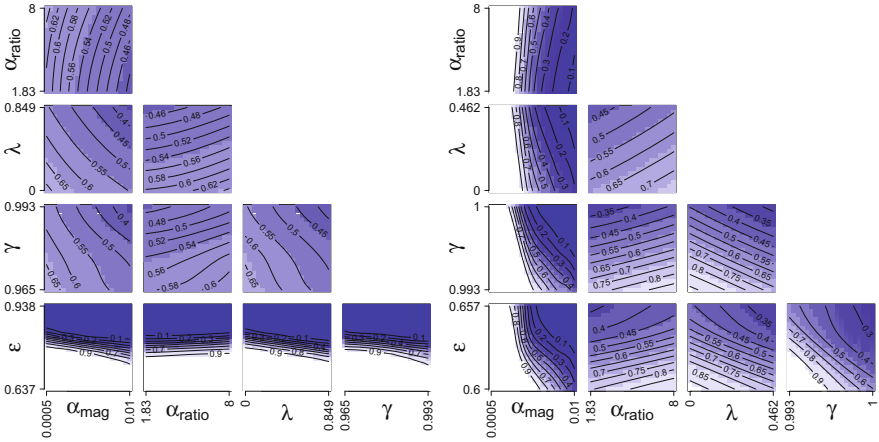


Fig. 5.7 FANOVA graphs showing first-order and two-way interaction effects for the mountain car domain.

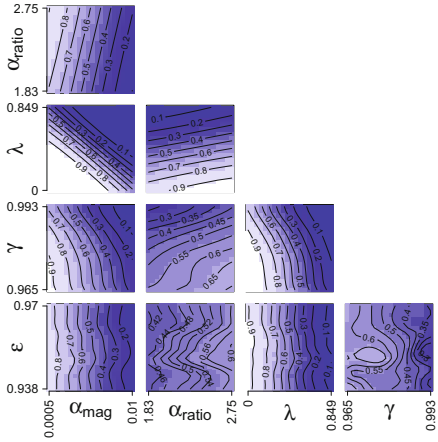
first-order indices and interactions within each subregion. Similarly to Fig. 5.6, we see that each of these graphs look very different, such that the first-order sensitivity indices are not consistent across subregions, nor are the interaction effects between variables.

Figures 5.8 and 5.9 show plots of the response surface projected onto pairs of parameters. These plots show the probability over [0, 1] that the surface is above a set threshold value where the threshold value used was the median value of the responses, also shown in Table 5.6. Again, the surface projections for each of the subregions are quite different, as with the sensitivity indices and with the FANOVA graphs. The results of the sensitivity analysis pair well with these surface plots as well, where the sensitivity of the parameters are reflected in the magnitude of change across their respective dimensions in the surface plots. Additionally, the interactions among parameters are also evident in these plots as well.



a Subregion 6: Threshold value: 48.1

b Subregion 11: Threshold value: 59.9



c Subregion 13: Threshold value: 40.8

Fig. 5.8 Response surface projections for convergent subregions 6, 11, and 13. The contour lines correspond to the probability that the surface lies above the threshold value of the subregion.

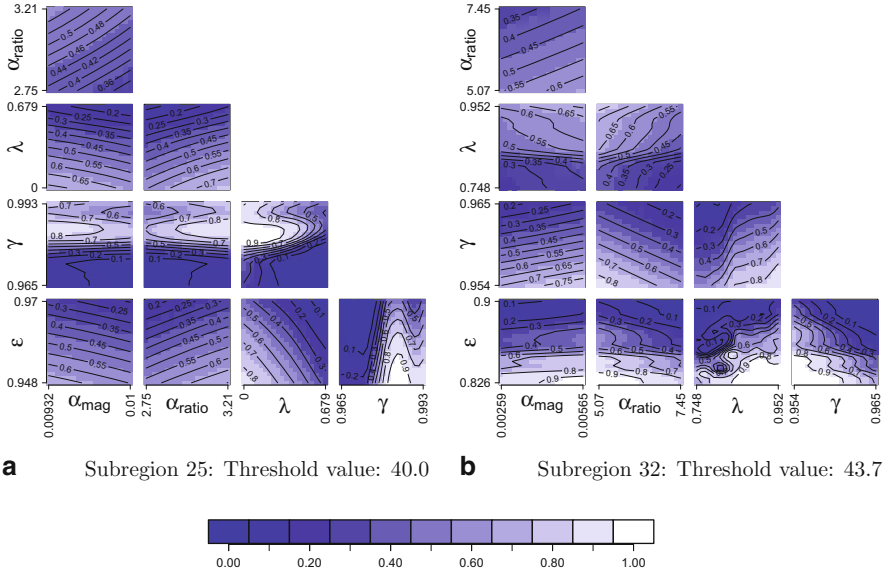


Fig. 5.9 Response surface projections for convergent subregions 25 and 32. The contour lines correspond to the probability that the surface lies above the threshold value of the subregion.

5.4 Discussion

Although the mountain car problem is a simple problem from a reinforcement learning perspective, we find that its parameter space is quite complex. We find that parameters influence the performance of learning this domain in very different ways for the different convergent subregions of the parameter space. Some subregions have a single parameter that has a significant effect on the learning performance, whereas others have multiple parameters that all have non-negligible effects on learning performance. Additionally, the interaction structure among variables is also quite different across the parameter subregions. Furthermore, we find that the shape of the response surfaces are quite different across subregions as well.

In terms of convergence of the TD(λ) algorithm, the ranges of the parameters of the convergent subregions are both consistent and inconsistent with what is generally recommended for reinforcement learning. The three parameters for the TD(λ) algorithm, λ, γ, and ϵ are both consistent and inconsistent with what is recommended. The temporal discount factor λ is generally recommended to be in the middle to upper range of the parameter space (approximately 0.5–0.8) (Tesauro 1992; Patist and Wiering 2004; Wiering et al. 2007; Wiering 2010). We find that even low values of λ allow for convergence. When $\lambda \rightarrow 0$, the weight updates from the TD(λ) algorithm become based only on the difference between the current and subsequent time steps, and information at other time steps has no or little role in the weight updates. This is regarded as a slow method of learning, though it seems to work fine in this particular problem.

The next-state discount factor γ should be in the upper range of [0.9, 1.0], and although this parameter isn't given much consideration in the literature, it is consistent with what is suggested. The action exploration-exploitation trade-off parameter ϵ does not have a consistent range over its parameter space that allows for convergence. However, in three subregions (13, 25, and 32), this parameter should be towards the upper end of [0.6, 1.0], and this is consistent with most applications and intuition that, in general, knowledge should be exploited. Although a high value of ϵ may choose many erroneous actions early on in the learning process, the correct actions will eventually surface and can then be exploited thereafter to maintain high performance. It is interesting that there are subregions that have a wide range or that have low values for ϵ (subregions 6 and 11, respectively). With subregion 6, ϵ is the most influential parameter with respect to performance and we find that larger values do have better performance based on the surface projections. Subregion 11 has α_{mag} as its most influential parameter and performance seems to improve with larger values of this parameter.

The magnitude of the learning rate of the neural network α_{mag} can be anywhere in the range $[5 \times 10^{-4}, 0.01]$ for some subregions, but it must be more constrained in other subregions. The ratio of the learning rates α_{ratio} can also take on a large and varying range; however, we do find that this ratio must just not be close to one, which supports Embrechts et al. (2010), and this makes sense based on the magnitudes of the error gradients between layers. Additionally, when considering α_{mag} and α_{ratio} together, especially for subregions 25 and 32 which have small ranges for each parameter, we see that if α_{mag} is larger, then α_{ratio} must be smaller (subregion 25), and vice versa (for subregion 32). In terms of performance within the convergent subregions, α_{mag} has a great influence in two of the subregions, although α_{ratio} has little influence in any of the subregions.

The mountain car problem was explored in previous work in which we used a more classical experimental design that was defined over a parameter space that was known to converge relatively well based on prior experience (Gatti et al. 2013) (Appendix B). A direct comparison of the results from the sequential CART approach to this previous work is not possible, though some things can be said. If all convergent parameter subregions are considered together, the neural network learning rate parameters used in the previous work are very similar to the convergent parameter space found using sequential CART. In the previous work, we allowed λ to range over [0.1, 0.9], γ to range over [0.95, 0.99], and ϵ to range over [0.7, 0.9], and these ranges are all generally consistent with what was found using sequential CART (Fig. 5.3). In the previous work, as $\gamma \rightarrow 0.95$, the probability of convergence became small, and all convergent subregions for γ were found to lie just above 0.95. Excluding variations that are possibly due to the sampling resolution, the previous work showed rather consistent convergence across values of λ, and the convergent subregions in the current work show that λ can be set over a rather large range of [0.000, 0.952]. Because of the differences in the methods used to assess the performance (i.e., number of time steps to the goal) of reinforcement learning, where the previous work used a linear model and the current work uses kriging metamodeling and global sensitivity analysis, we refrain from making any comparisons.

In general, we find that the ranges and locations of parameters that allow for convergence are consistent with intuition and are explainable. However, once individual subregions are explored on a more detailed level, we find that some subregions can be quite simple with a single parameter that dominates performance, whereas others can be quite complex with multiple parameters that have an effect on performance. Additionally, in these complex subregions, the reasons why each of these parameters have the effects that are seen is not intuitively obvious and will likely require further experimentation.

References

Embrechts, M. J., Hargis, B. J., & Linton, J. D. (2010). An augmented efficient backpropagation training strategy for deep autoassociative neural networks. In *Proceedings of the 2010 International Joint Conference on Neural Networks (IJCNN), Barcelona, Spain, 18–23 July* (pp. 1–6). doi: 10.1109/IJCNN.2010. 5596828

Gatti, C. J., Embrechts, M. J., & Linton, J. D. (2013). An empirical analysis of reinforcement learning using design of experiments. In *Proceedings of the* 21st *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN), Bruges, Belgium, 24–26 April* (pp. 221–226). Bruges, Belgium: ESANN.

LeCun, Y., Bottou, L., Orr, G., , & Müller, K. (1998). Efficient backprop. In Orr, G. & Müller, K. (Eds.), *Neural Networks: Tricks of the Trade*, volume 1524 (pp. 5–50). Berlin: Springer.

Moore, A. W. (1990). *Efficient memory-based learning for robot control*. Unpublished PhD dissertation, University of Cambridge, Cambridge, United Kingdom.

Patist, J. P. & Wiering, M. (2004). Learning to play draughts using temporal difference learning with neural networks and databases. In *Proceedings of the 13th Belgian-Dutch Conference on Machine Learning, Brussels, Belgium, 8–9 January* (pp. 87–94). doi: 10.1007/978-3-540-88190-2_13

Tesauro, G. (1992). Practical issues in temporal difference learning. *Machine Learning*, 8(3–4), 257–277.

Wiering, M. A. (2010). Self-play and using an expert to learn to play backgammon with temporal difference learning. *Journal of Intelligent Learning Systems & Applications*, 2(2), 57–68.

Wiering, M. A., Patist, J. P., & Mannen, H. (2007). *Learning to play board games using temporal difference methods* (Technical Report UU–CS–2005–048, Institute of Information and Computing Sciences, Utrecht University). Retrieved from http://www.ai.rug.nl/~mwiering/group/articles/learning_games_TR.pdf.

Chapter 6
The Truck Backer-upper Problem

In this chapter, we focus on analyzing the truck backer-upper problem (TBU), a real world-like control problem. In this problem a tractor trailer truck must be backed into a specific location with a specific orientation by controlling the orientation of the wheels of the truck cab. This problem has been considered in other works, albeit using slightly different or more complex approaches. Nguyen and Widrow (1990a, b) used a neural network-based self-learning control system approach to back up a single trailer truck, though this was not based on a reinforcement learning approach. Vollbrecht (2003) used a complex hierarchical reinforcement learning approach based state space partitioning using a kd-tree (a k-dimensional partitioning data structure) and a tabular Q-function to learn how to back up a single trailer truck. As far as it is known, there has been no work by others that uses a simple reinforcement learning approach to learning the truck backer-upper problem. However, it should be acknowledged that we have used a similar implementation for learning the TBU problem (Gatti and Embrechts 2014), although this previous work did not use a large design of experiments approach as in the current work. Parts of the description of the problem in this chapter are similar to that presented in (Gatti and Embrechts 2014).

The goal of the truck backer-upper problem is to learn a control system that is capable of backing up a tractor trailer truck from an arbitrary location and orientation to a loading dock (Fig. 6.1). The dynamics of the trailer truck were based on those from Schoenauer and Ronald (1994). It is implicitly assumed in this problem that the truck can only back up, and that it cannot drive forward and attempt to back up again. The position of the rear of the trailer was defined by its horizontal and vertical coordinates, x and y (in meters), respectively. The orientation of the trailer with respect to a horizontal axis was defined by θ_T, and the orientation of the cab with respect to the trailer was defined by θ_C (radians). The state of the trailer $\mathbf{x}$ at any time step was characterized by these four state variables: $\mathbf{x} = (x, y, \theta_T, \theta_C)$. The state update equations are as follows:

$$x' = x - B \cdot cos(\theta_T)$$
$$y' = y - B \cdot sin(\theta_T)$$

C. Gatti, *Design of Experiments for Reinforcement Learning*,
Springer Theses, DOI 10.1007/978-3-319-12197-0_6

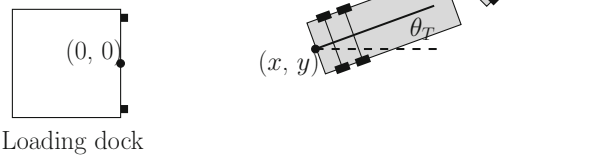


Fig. 6.1 The state of the truck is defined by the rear trailer position (x,y), the trailer angle θ_T, and the cab angle θ_C. The goal of the problem is to back the truck into the loading dock at $(x, y) = (0, 0)$ where $\theta_T = 0$.

$$\theta'_T = \theta_T - arcsin\left(\frac{A \cdot sin(\theta_C)}{L_T}\right)$$

$$\theta'_C = \theta_C + arcsin\left(\frac{v \cdot sin(u)}{L_C + L_T}\right)$$

where $A = v \cdot cos(u)$, $B = A \cdot cos(\theta_C)$, $v = 3$, $L_T = 14$ (tailer length), and $L_C = 6$ (cab length). The wheel angle relative to the cab angle is specified by u (radians), and three discrete actions were allowed: $u = \{-1, 0, 1\}$. The truck velocity was not taken into account as backing the trailer is assumed to be a slow process. The truck was restricted to the domain boundaries $x = [0, 200]$ and $y = [-100, 100]$. The goal of this problem was to have the trailer positioned at the loading dock with a specific orientation within a fixed number of time steps. This goal criteria can be represented as: $x = 0$, $y = 0$, and $\theta_T = 0$.

6.1 Reinforcement Learning Implementation

The truck backer-upper problem can be viewed as a reinforcement learning problem where each learning run attempts to back up the truck from some initial state to a goal state, and where many learning runs are used to learn how to control the truck at different locations and orientations throughout the domain. More specifically, the truck begins at a random location and orientation, and the wheels of the truck are controlled to back up the truck to a specific location and orientation. When the truck reaches the goal, positive feedback is provided, indicating that the control strategy in that learning run was good, whereas if the truck does not reach the goal for some reason, negative feedback is provided.

The temporal difference algorithm TD(λ) (Sect. 2.2.3) was used to train a three-layer neural network to learn the value function $V(\mathbf{x}, a)$ that approximates the value of being in state $\mathbf{x}$ and taking action a (at time the current time step). The neural network had four inputs, corresponding to the four state variables. This is a control-type problem that naturally lends itself to using a neural network with an output node for each of the possible actions, and thus the neural network had three output nodes. The number of nodes in the hidden layer was a variable in the experimental design. The x and y components of the state vector were scaled over $[-3, 3]$ based on the

Table 6.1 Domain characteristics for the truck backer-upper domain.

Initial conditions	$x = U[100, 150]$
	$y = U[-20, 20]$
	$\theta_T = U[-1.0, 1.0]$ (radians)
	$\theta_C = U[-0.5, 0.5]$ (radians)
Actions	3: $[0.0, \pm 1.0]$ (radians)
Rewards/penalties	Achieve goal: +5
	Exit domain boundaries: −0.1
	Trailer-cab jack-knife: −0.1
	Large trailer angle: −0.1
	T_{max} exceeded: −0.1
Reward function per time step[a]	$r = 0.2\left(-0.15x^{0.6} - 0.01\lvert y\rvert^{1.2} - 0.5\delta\theta_T + 2\right)$
Goal tolerance[a]	$d \leq 5$
	$\delta\theta_T \leq 0.5$
Number of episodes	10,000
Number of time steps per episode T_{max}	300
Performance time window (p_{win})	300
$conv_{val}$	0.5
$conv_{rng}$	0.005
$conv_m$	1×10^{-4}

[a] $\delta\theta_T = \|\theta_T^* - \theta_t\| \in [-\pi, \pi]$; $d = \sqrt{x^2 + y^2}$

boundaries of the domain in order to put these state values on approximately the same scale as θ_T and θ_C. The hidden layer used a hyperbolic tangent transfer function and the output layer used a linear transfer function. The input and hidden layers both had bias nodes with constant values of +1. Network weights were initialized by sampling from $U[-0.1, 0.1]$.

Additional characteristics and parameter settings of the domain are shown in Table 6.1. Each episode began with the initial position and the orientation of the truck sampled from relatively wide uniform distributions. The goal of this problem was to position the truck at the loading dock (positioned at $(0, 0)$) in the correct orientation such that its distance to the loading dock was less than 5, and the difference in its orientation with respect a neutral orientation (i.e., $\theta_T = 0$) was less than 0.5. These bounds are somewhat loose though we could consider this initial learning procedure to be a seed to subsequent training, and thus we are interested in learning general knowledge about controlling the truck in this initial stage. When the truck reached the goal within these tolerances, a reward of r $= +5$ was provided. When the truck was outside of this region, a reward was provided based on a function of the trailer position and orientation as specified in Table 6.1. This reward function was conceived based on its shape over the state variable space such that the reward is greater (i.e., positive

Table 6.2 Variables and their associated ranges used in sequential CART for the TBU problem.

Variable	Description	Range	RL component
n_{hnodes}	Number of hidden layer nodes	[11, 61][a]	Neural network
α_{mag}	Base (input–hidden layer) learning rate	[0.003, 0.01]	Neural network
α_{ratio}	Learning rate ratio	[1.0, 8.0]	Neural network
λ	Temporal discount factor	[0.0, 1.0]	TD(λ) algorithm
γ	Next-state discount factor	[0.9, 1.0]	TD(λ) algorithm
ϵ	P(action exploitation)	[0.6, 1.0]	TD(λ) algorithm

[a] Values take on discrete, integer values

or less negative) if the state of the truck is closer to its goal. Note that other reward functions would also likely work in this problem.

Ideally, episodes would terminate if the truck was successfully backed into the loading dock with the desired orientation. Alternatively, episodes could terminate if the cab jack-knifed ($\theta_C > \frac{\pi}{2}$), if the trailer angle became large ($\theta_T > 4\pi$), if the front of the cab or the rear of the trailer exited the domain boundaries, or if the maximum number of time steps allowed was reached. In any of these cases, a penalty of $r = -0.1$ was provided to the agent. The empirical convergence criteria described in Sect. 4.1.4 was used to automatically determine if learning had converged in each design run and the threshold values used for the convergence assessment are also shown in Table 6.1.

6.2 Sequential CART

Our initial exploration of the parameter space for the truck backer-upper problem consists of applying sequential CART to find convergent parameter subregions. Six variables of the neural network and the TD(λ) algorithm were studied for the TBU problem (Table 6.2). Similar to the mountain car problem, the ranges of these parameters were expanded somewhat relative to what is generally used with the TD(λ) algorithm and based on prior experience with these methods. Variables related to the neural network included the number of nodes in the hidden layer and the learning rates of the neural network, where the learning rates are specified by a magnitude α_{mag} and ratio α_{ratio}.

The learning rates of the network were set based on the variables α_{mag}, a base learning rate between the input and hidden layer of the network, and α_{ratio}, the ratio of the learning rates for each subsequent layer of the network with respect to the input–hidden layer; these actual learning rates were determined using the same procedure as presented for the mountain car problem in Sect. 5.2. Variables related to the learning algorithm TD(λ) algorithm include λ, γ, and ϵ.

Table 6.3 Parameter settings for sequential CART modeling and CART models.

Description	Value
Number of variables (d)	6
Initial number of design points	120 ($20d$)
Replications per design point	3
Number of sequential CART iterations	4
Number of additional design points per *low*-leaf	30
Sample design method	Latin hypercube
Minimum design points per split[a]	$\lfloor 0.03\|\|y\|\| \rfloor$
Minimum design points per leaf[a]	$\lfloor 0.01\|\|y\|\| \rfloor$
Minimum complexity change per split	0.001
Maximum number of surrogates	5
Proportion of low points per leaf	0.8
Convergence proportion threshold	0.9
Minimum design points per convergent domain	8

[a] $\|\|y\|\|$ is the length of the response vector for the current iteration

With the exception of the number of nodes in the hidden layer (n_{hnodes}), all of these variables take on continuous values. The number of hidden nodes was included in this study in order to determine if there was a specific size network that was required to learn this problem; a very small network may not be able to learn a complex value function, whereas a very large network may not be able to be trained sufficiently in the alloted learning time. Despite n_{hnodes} taking on discrete and integer values, it was treated as a continuous variable in both the sequential CART modeling and in the stochastic kriging model because it was assumed that there is some continuity of the response surface between nearby values. When generating design points using Latin Hypercube sampling, the number of hidden nodes was sampled continuously over the defined range, however, this value was rounded to the nearest integer value, and this integer was used in the reinforcement learning run and when creating the CART models and kriging metamodel.

The parameters used in the sequential CART modeling are shown in Table 6.3. The number of initial design points was set to $20d$ (where $d = 6$ is the number of design variables). Note that this number of design points is nearly twice that recommended for deterministic computer experiments (Loeppky et al. 2009), though there are significant differences between these problems. Each design point evaluated using sequential CART consisted of having the agent attempt to learn the TBU domain in 10,000 episodes, where an episode consists of one attempt at backing the truck into the loading dock. The initial experimental design therefore consisted of 120 unique design points, with 3 replicates each, totaling 360 initial runs, which was generated using Latin Hypercube sampling (LHS). Subsequent designs for each iteration of the

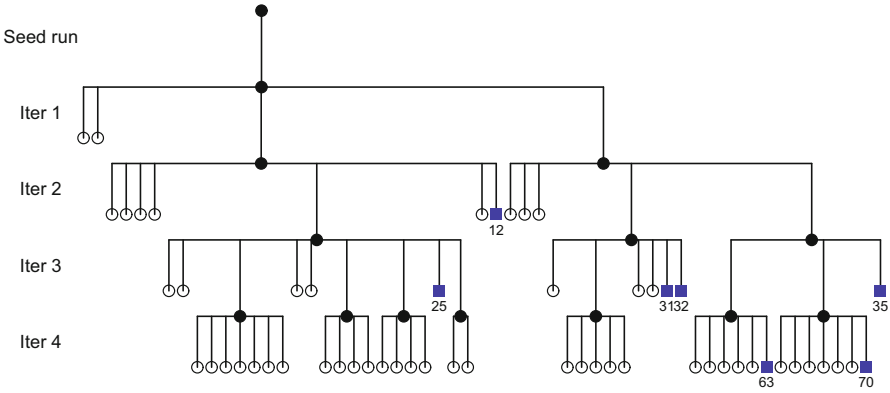


Fig. 6.2 Sequential CART process for the TBU problem. Parameter subregions that are candidates for further experimentation are shown as *black circles*, subregions that are pruned from further experimentation are shown as open circles, and convergent subregions are shown as *blue squares*.

sequential CART algorithm were also LHS designs, consisting of 30 design points, again with 3 replicates each, for a total of 90 runs.

6.2.1 Convergent Subregions

Figure 6.2 shows the sequential CART process for the TBU problem. Following the sequential CART procedure, additional screening of the leaf nodes required that there be at least 12 design points in the subregion and that the dimensionality ratio was less than or equal to 20. Figure 6.3 shows the ranges of the parameters for each of the convergent subregions. Table C.3 in the Appendix provides the numerical values of the convergent subregions.

The convergent domains have some interesting characteristics. For nearly all of the parameters, the ranges of each of the parameters are a small subset of the full parameter hypercube. In terms of the neural network, we can see that there is a minimum required size to the network of between 26 and 40 hidden nodes, suggesting that a smaller network is not likely to be able to learn this problem. The magnitude of the learning rates needs to be closer to 0.01, though there is not a consistent range of the ratio of the learning rates between the layers of the network across convergent subregions. In some subregions, this ratio does not matter at all or very little (e.g., subregions 12, 31, 32, 35, and 70), whereas this ratio can take on values over a small region in other subregions (e.g., subregions 25 and 63).

There seem to be very specific parameter subregions for the parameters of the TD(λ) learning algorithm. The action selection exploration/exploitation trade-off parameter ϵ needs to be set high to at least about 0.89, and these regions extend all the way to 1.00 in some subregions (e.g., 12, 35, and 63). This suggests that exploiting actions the vast majority of the time is most beneficial. That is, learning

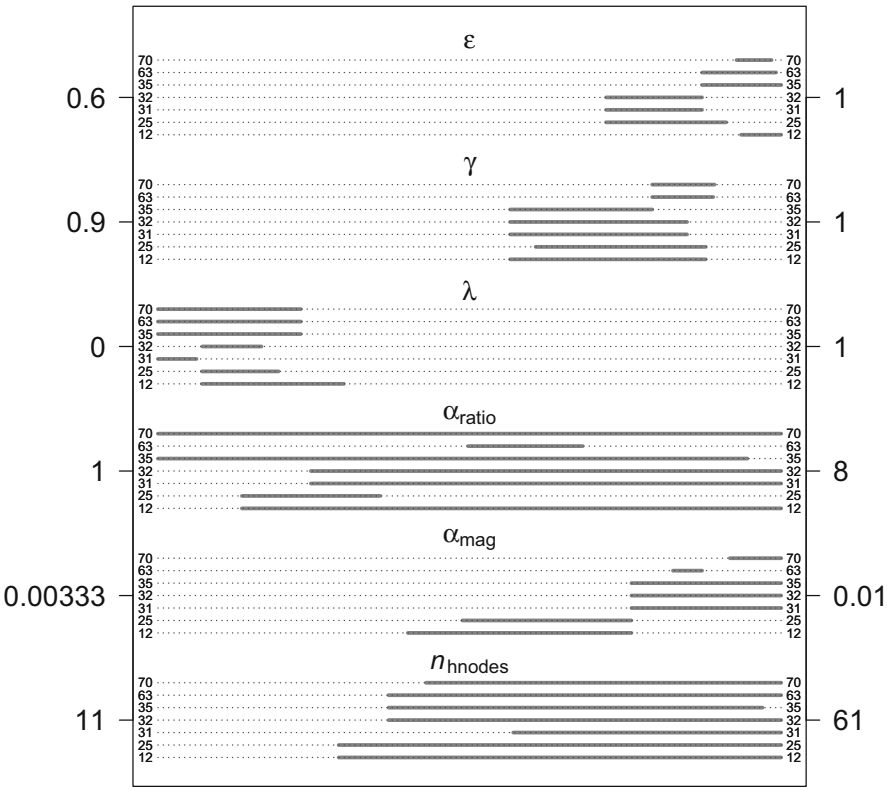
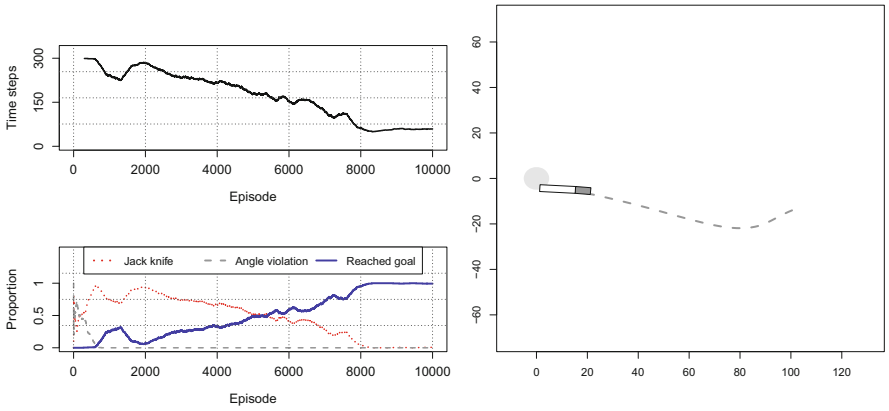


Fig. 6.3 Parameter ranges for the convergent subregions for the TBU problem.

through exploiting (possibly incorrect) knowledge is better than exploring actions, and this may be due to the fragility of the problem where small errors can result in the truck jack-knifing.

The next-state discount parameter γ generally ranges from 0.96 to 0.99, and this is quite consistent with what is suggested and used in other implementations in the literature (Thrun and Schwartz 1993; Thrun 1995; Gatti et al. 2013). However, the same cannot be said for the temporal discount factor λ. We find that this parameter generally has to be quite low (0.30 and below), which contrasts with the many implementations of TD(λ) that suggest that λ be set to between 0.6 and 0.8 (Tesauro 1992; Patist and Wiering 2004; Wiering et al. 2007; Wiering 2010). Lower values of λ pass back little information from time step to time step, and thus learning is generally perceived to be slow. An important part of the control strategy in this problem is simply selecting actions that avoid jack-knifing the truck. Much of what is learned may therefore simply be what actions to perform and what actions to avoid for any configuration of the truck regardless of the location of the truck. It is possible that learning this mapping between the truck configuration and selecting actions to avoid jack-knifing may be a rather *static* learning problem in that this mapping may



a Performance and episode termination proportions. **b** Example trajectory after learning.

Fig. 6.4 Example performance from a sample in subregion 12, with parameters: $n_{hnodes} = 28$, $\alpha_{mag} = 7.74 \times 10^{-3}$, $\alpha_{ratio} = 6.713$, $\lambda = 0.240$, $\gamma = 0.969$, $\epsilon = 0.990$. The *top plot* on the *left* shows the moving average of the number of time steps to the goal, and the *bottom plot* shows moving averages of the proportions of how episodes terminated. The figure on the *right* shows the trajectory of the truck backing up, when starting from a random position and orientation, after learning had converged. The gray shaded region in this figure indicates the acceptable error region that the truck must reach.

not extend for many time steps, and thus information does not need to be passed back far in time. It is also possible that the low values of λ in the convergent subregions are due to experimental sampling, as we do not perform an exhaustive experimentation, however, this is a deficiency of any experimental procedure.

In two of the subregions (35 and 70), the ratio of the learning rates of the network weights α_{ratio} extends from 1 to 8, however, the lower bound for this parameter in the other subregions is approximately 2. A large learning rate ratio may not be essential, but it does seem to be useful in many of the convergent subregions. The magnitude of the learning rate α_{mag} needs to be on the upper end of the interval $[0.00333, 0.01]$. If our supposition that the vast majority of what is learned is the avoidance of jack-knifing is true, (relatively) larger learning rates may aid in learning this knowledge quicker than smaller learning rates. An example of a convergent learning run from subregion 12 is shown in Fig. 6.4, which also shows an example trajectory of the truck backing up.

Summary statistics for the convergent subregions are shown in Table 6.4. Interestingly, when compared to the statistics for the convergent subregions for the mountain car problem, the average radius and the sum of the radii for the convergent subregions in the TBU problem do not necessarily get smaller for subregions found in later iterations.

Regional sensitivity analysis (Fig. 6.5) generally supports the convergent subregions found with sequential CART. For any single parameter, the larger the difference

Table 6.4 Summary statistics for the convergent subregions for the TBU problem.

Subregion #	Iteration	p_{conv}	No. points	Dim. ratio	Ave. radius	Sum radii
12	2	0.917	12	13.415	0.212	2.540
25	3	0.917	12	5.776	0.149	1.792
31	3	0.905	21	12.194	0.160	1.925
32	3	0.939	33	7.947	0.180	2.158
35	3	0.910	78	7.445	0.198	2.370
63	4	1.000	18	13.462	0.109	1.307
70	4	0.967	30	17.778	0.170	2.038

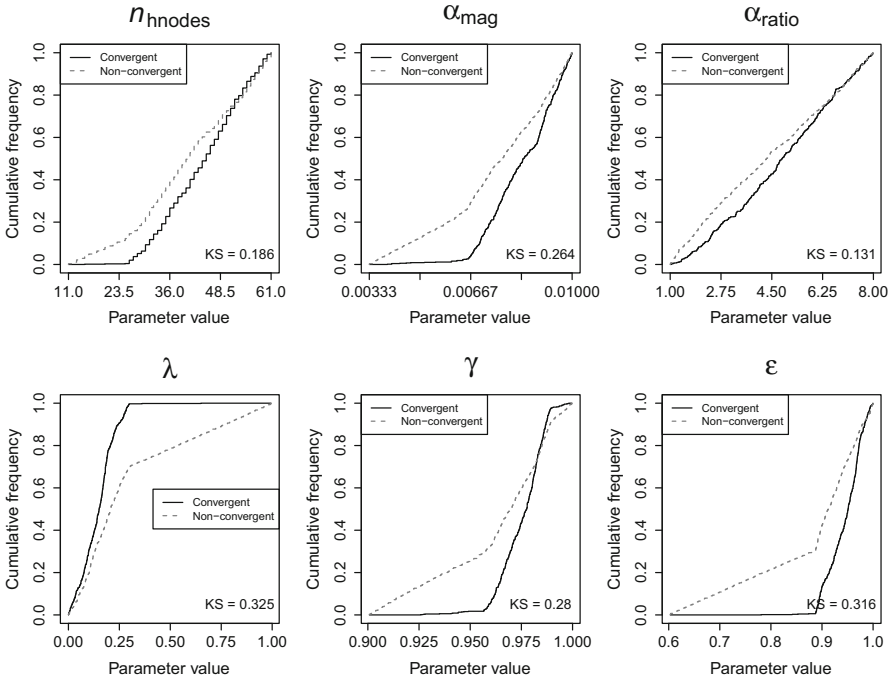


Fig. 6.5 Regional sensitivity analysis based on the convergence of all experimental runs for the TBU problem.

between the cumulative distributions, the more likely that the parameter has an effect on convergence. Furthermore, the relative shape of the distributions can indicate what region of the parameter space allows for convergence. For example, the RSA plot for λ shows that the convergent cumulative distribution rises quickly and reaches its maximum or total value at around $\lambda = 0.25$. This suggests that convergence is more likely to occur in the lower range of the parameter space. Comparing this qualitative

Table 6.5 Parameter settings for kriging metamodeling for the TBU domain.

Description	Value
Number of variables (d)	6
Number of design points	300 ($50d$)
Replications per design point	5
Minimum number of convergent replications	3
Kriging model form	Linear (universal kriging)
Covariance kernel	Matérn 5/2
Covariance parameter range	[0, 2]

assessment to the subregion boundaries in Fig. 6.3, we can see that all of the convergent subregions have low values for λ. RSA also shows what parameters have little effect on convergence. For example, there is little difference between the convergent and non-convergent cumulative distributions for the ratio of the learning rates, and this is supported by what is shown in Fig. 6.3 for this parameter such that there is not a consistent portion of the parameter space that enables convergence. The RSA plots for all of the other parameters also have good agreement with the convergent subregions found with sequential CART.

6.3 Response Surface Metamodeling

For each of the convergent parameter subregions found using sequential CART modeling, we using stochastic kriging to create a metamodel of the response surface. The experimental design for each of the convergent subregions consists of a Latin-hypercube design (LHS) of 300 design points ($50d$) where each design point is replicated five times. Design points were only included in the model that converged for at least three replications. The kriging model was constructed based on the mean and variance of each of the convergent replications of each design point. Additional details on the experimental design and kriging metamodeling are shown in Table 6.5.

Statistics on each of the metamodels for the convergent subregions are shown in Table 6.6; model parameters for each of the kriging models are provided in Table C.4 in the Appendix. We find that the performance in all of these subregions is very similar, and these performance values are closer than the subregions for the mountain car problem. In general, the metamodels fit relatively well with the exception of subregions 63 and 70 (based on R^2 and Q^2), although the RMSE and MAE is rather close to these metrics for the other metamodels. Still, it may be wise to interpret the results from the metamodels of subregions 63 and 70 with caution; furthermore, using the metamodels from these subregions as a replacement model should be done with caution as well.

Table 6.6 Kriging metamodel statistics for the TBU domain.

Subregion #	No. points	p_{conv}	y_{min}	y_{med}	y_{max}	RMSE	MAE	RMSE_{loo}	MAE_{loo}	R^2	R^2_{adj}	Q^2
12	145	0.595	50.7	59.0	65.0	1.090	0.829	1.490	1.180	0.906	0.902	0.826
25	156	0.641	54.0	58.6	62.2	0.693	0.504	0.800	0.605	0.784	0.759	0.712
31	198	0.76	51.9	58.9	62.5	0.867	0.658	0.978	0.749	0.837	0.832	0.793
32	180	0.683	51.0	58.6	62.3	0.813	0.627	1.110	0.859	0.898	0.895	0.811
35	178	0.739	49.0	57.9	62.4	1.040	0.803	1.430	1.120	0.871	0.866	0.758
63	183	0.711	57.6	61.2	65.6	0.978	0.750	1.140	0.891	0.451	0.432	0.260
70	167	0.653	57.4	61.0	66.2	0.975	0.761	1.270	1.030	0.662	0.628	0.427

The sensitivity indices (Fig. 6.6) generally show consistent results across all three sensitivity methods for each convergent subregion and for each parameter (see Sect. 4.2.6 for a description of the sensitivity indices). The sensitivity indicies of subregions 12, 25, 31, 35, and 63 all indicate that γ is the most important parameter, and all other parameters have nearly negligible importance. Subregion 63 shows that, while γ is also the most important parameter, the number of hidden nodes, λ, and ϵ are also somewhat important. Subregion 70 shows that α_{ratio}, λ, γ, and ϵ all have modest importance.

From the FANOVA graphs (Fig. 6.7), we again see that γ overwhelmingly has the most dominant influence with respect to the performance in most of the subregions. The interaction structure between variables, however, is quite different across the subregions. In some subregions (12, 31, 32, and 35), γ also has a large two-way interaction, though the other variables that participate in this interaction are different. Additionally, subregions 25 and 70 don't include γ in the main two-way interaction at all.

Figures 6.8 and 6.9 show 2-dimensional projections of the response surface from the metamodel for each of the convergent subregions. The most distinguishing feature of most these surface projections are the stark qualitative differences for the parameter γ compared to all other parameters. The projections along the γ dimension have a severe and relatively quick transition from high values to low values, and this holds when γ is compared to any other parameter. All other parameters have relatively flat response surfaces, with some peaks and valleys, and that are all generally around the median response value. These observations hold for domains 12, 25, 31, and 32 (Fig. 6.8a–d) and domain 35 (Fig. 6.9a). The visual significance of γ in these plots for the shape of the response surface is also consistent with the findings of the sensitivity analysis (Fig. 6.6) and the FANOVA plots (Fig. 6.7a–e).

However, domains 63 and 70 (Fig. 6.9b–c) are qualitatively different than the other domains because the sharp transition of the response surface for γ is absent. These surfaces show oblique transitions between high and low valued regions across a number of parameters. Additionally, the surfaces along the α_{mag} dimension are generally more complex and less smooth than along other dimensions. The importance of other parameters in these subregions is also supported by their sensitivity indices (Fig. 6.6) and the FANOVA plots (Fig. 6.7f–g).

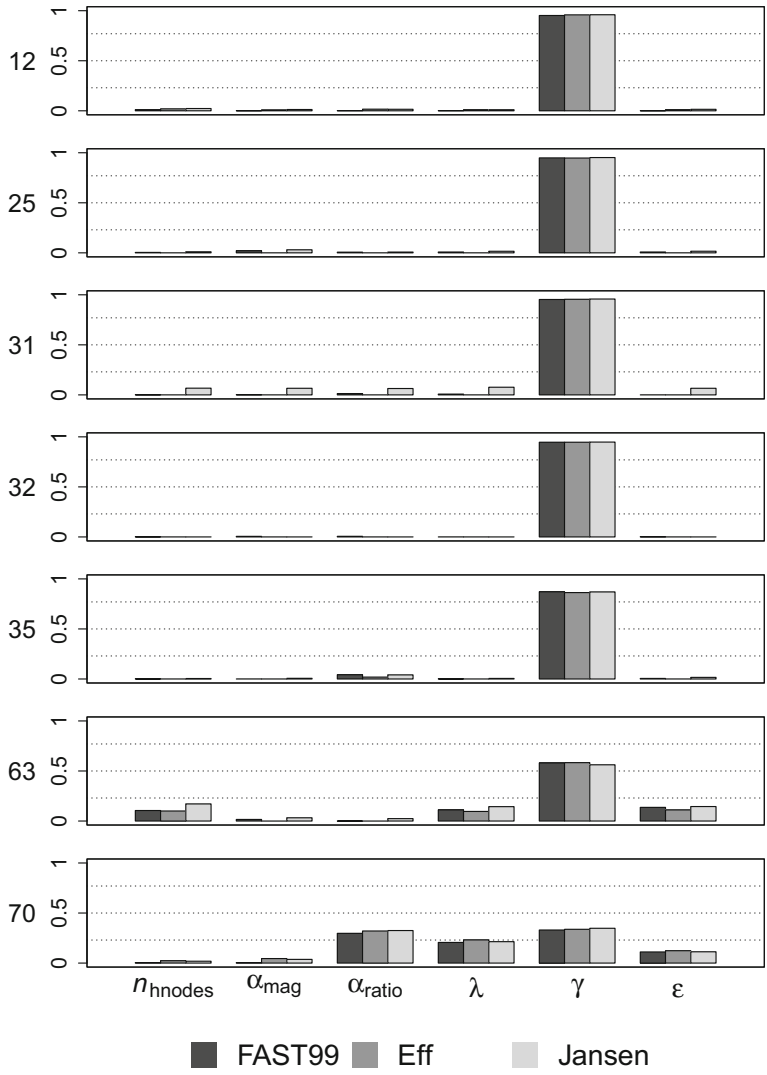
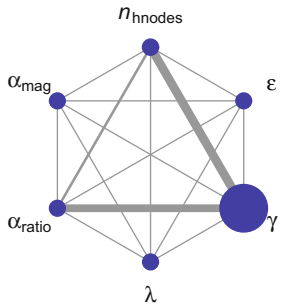


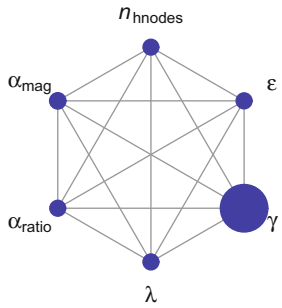
Fig. 6.6 First order sensitivity indices for the each parameter in the TBU problem. The subregion number is shown on the *left* side of the plots.

6.4 Discussion

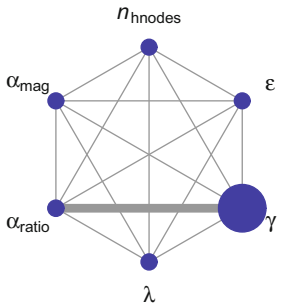
The truck backer-upper problem is considerably more difficult than the mountain car problem, and this is primarily because of the fragility and sensitivity of this problem. In other words, small and incorrect actions may easily result in the truck jack-knifing, which is an undesirable configuration. Yet, we are still able to find a



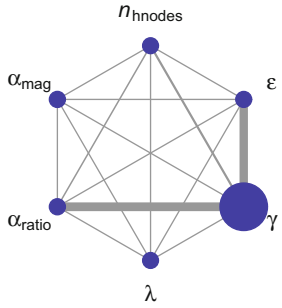
a Subregion 12



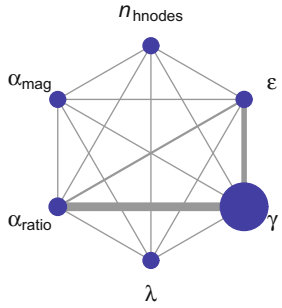
b Subregion 25



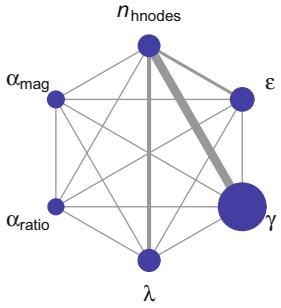
c Subregion 31



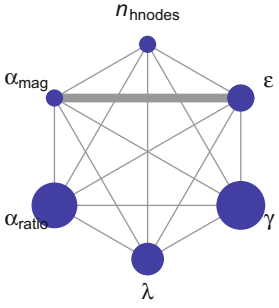
d Subregion 32



e Subregion 35



f Subregion 63



g Subregion 70

Fig. 6.7 FANOVA graphs showing first-order and two-way interaction effects for the TBU domain.

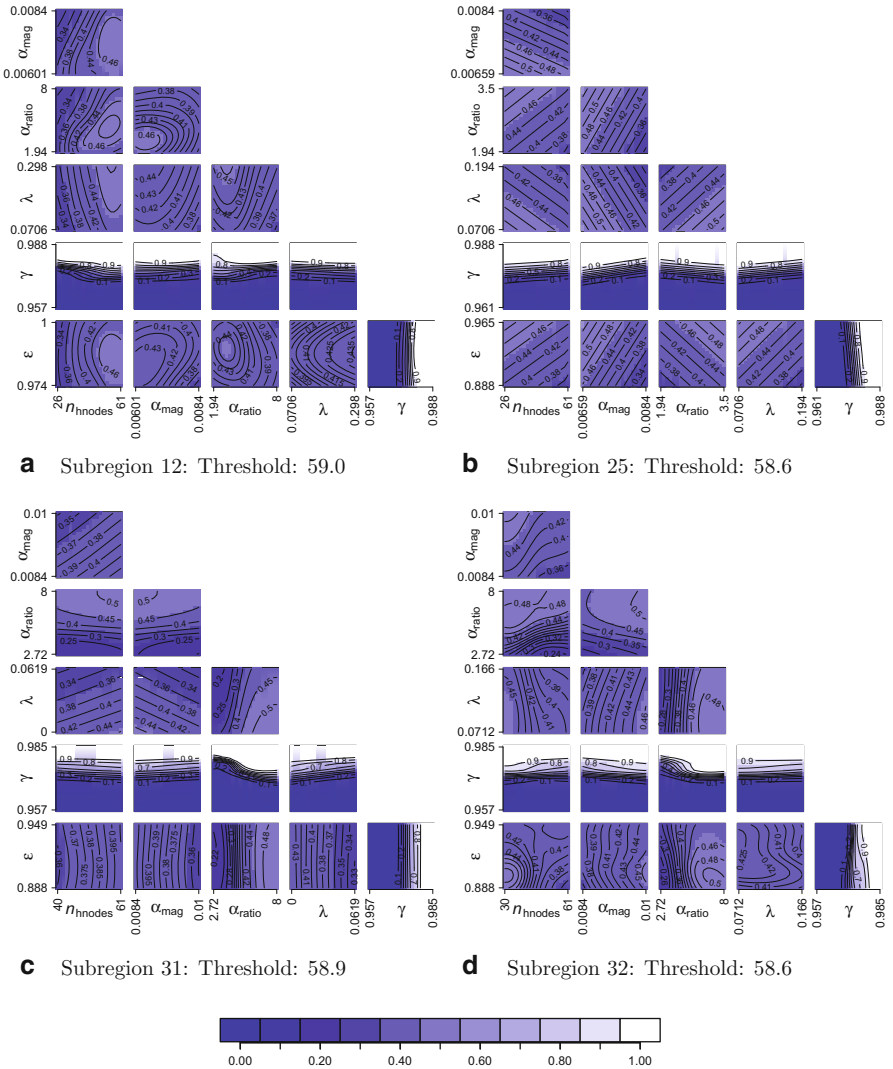


Fig. 6.8 Response surface projections for convergent subregions 12, 25, 31, and 32. The *contour lines* correspond to the probability that the surface lies above the threshold value of the subregion.

number of convergent parameter subregions that are able to learn how to control the truck and back it into a specific location with a specific orientation.

We found that most of the ranges of the parameters within these convergent subregions are consistent with those used in other applications of the TD(λ) learning algorithm, however, we found that λ must be set relatively low, which goes against what is generally recommended for the TD(λ) algorithm. The reason for this may be related to what knowledge is actually learned, i.e., the set of actions to avoid jack-knifing, although this was not specifically tested or known for sure.

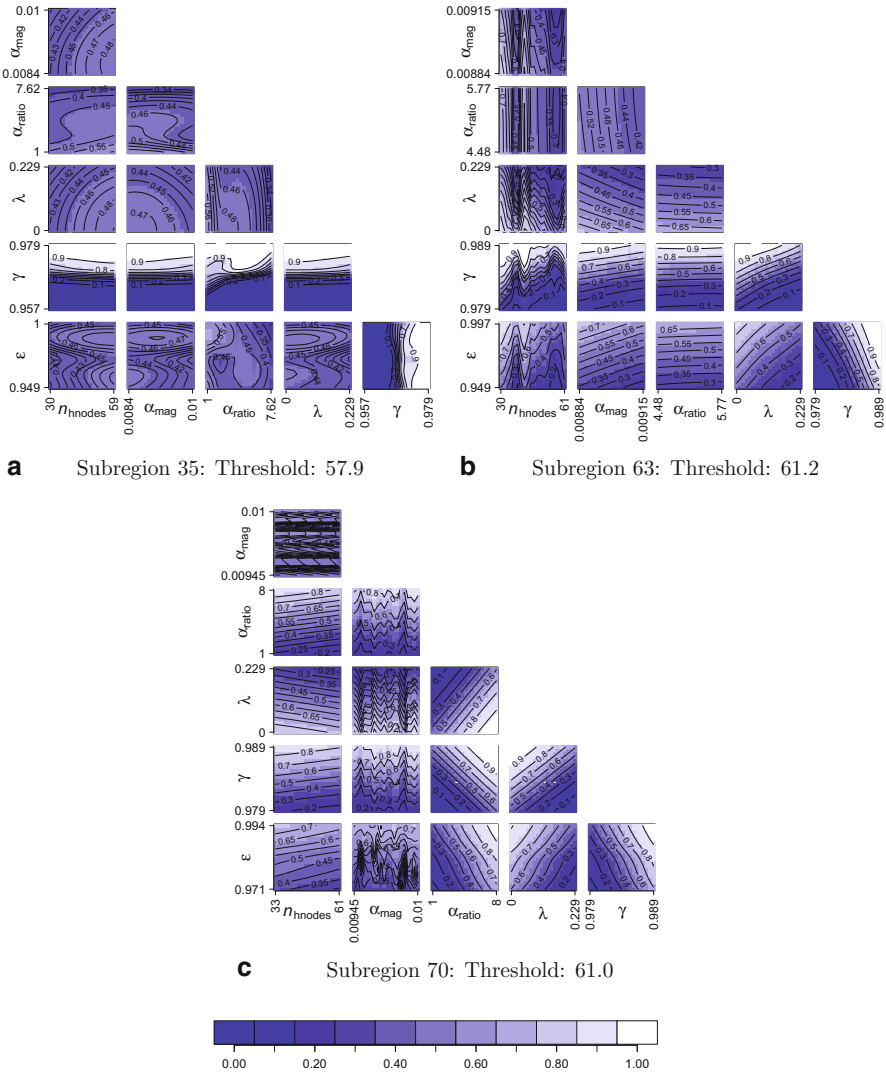


Fig. 6.9 Response surface projections for convergent subregions 35, 63, and 70. The *contour lines* correspond to the probability that the surface lies above the threshold value of the subregion.

The convergent subregions all had very similar values in terms of the number of time steps to the goal. Note that the initial position and orientation of the truck was randomized at the beginning of each learning run, and this will induce some variability in the number of steps to the goal. The consistency in the performance values may suggest that a very efficient control strategy is found whenever a learning run converges, and this was observed in the backing trajectory of a sample of convergent learning runs. That said, optimization of the control strategy may be a goal

in some circumstances, and it may be useful to explore the subregions that had the best performance (lowest number of time steps, i.e., subregions 12, and 35). Another consistency across the convergent subregions was the influence of γ, which was significantly greater than all other parameters in most cases. Beyond the first-order influence of γ, however, we found that the two-way interaction structure between the other parameters are quite different across the subregions, indicating that there are some differences in the shape or form of the response surfaces.

The TBU problem used in this work initialized the state of the truck to within a specific region and orientation, and it allowed for somewhat loose tolerances to achieve the goal location and orientation relative to a real-world implementation. However, we believe that the problem characteristics used herein is a good start to learning the TBU. The strategies learned from within any of the convergent subregions could then be used in a subsequent training scheme that is aimed at either generalizing the initial conditions of the truck, reducing the goal tolerances, or both. Such a sequential training scheme that goes beyond simply using only the TD(λ) algorithm is regarded as a heuristic. Although this is an interesting strategy for improving reinforcement learning, and will likely be explored in future work, it is beyond the scope of the current work.

References

Gatti, C. J. & Embrechts, M. J. (2014). An application of the temporal difference algorithm to the truck backer-upper problem. In *Proceedings of the* 22*nd European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*, Bruges, Belgium, 23–25 April. Bruges, Belgium: ESANN.

Gatti, C. J., Embrechts, M. J., & Linton, J. D. (2013). An empirical analysis of reinforcement learning using design of experiments. In *Proceedings of the* 21*st European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*, Bruges, Belgium, 24–26 April (pp. 221–226). Bruges, Belgium: ESANN.

Loeppky, J. L., Sacks, J., & Welch, W. J. (2009). Choosing the sample size of a computer experiment: A practical guide. *Technometrics*, 51(4), 366–376.

Nguyen, D. & Widrow, B. (1990a). Neural networks for self-learning control systems. *IEEE Control Systems Magazine*, 10(3), 18–23.

Nguyen, D. & Widrow, B. (1990b). The truck backer-upper: An example of self-learning in neural networks. In Miller, W. T., Sutton, R. S., & Werbos, P. J. (Eds.), *Neural Networks for Control*. Cambridge, MA: MIT Press.

Patist, J. P. & Wiering, M. (2004). Learning to play draughts using temporal difference learning with neural networks and databases. In *Proceedings of the 13th Belgian-Dutch Conference on Machine Learning, Brussels, Belgium, 8–9 January* (pp. 87–94). doi: 10.1007/978-3-540-88190-2_13

Schoenauer, M. & Ronald, E. (1994). Neuro-genetic truck backer-upper controller. In *Proceedings of the IEEE Conference on Computational Intelligence, Orlando, FL, 27 June–2 July* (Vol. 2, pp. 720–723). doi: 10.1109/ICEC.1994.349969

Tesauro, G. (1992). Practical issues in temporal difference learning. *Machine Learning*, 8(3–4), 257–277.

Thrun, S. (1995). Learning to play the game of Chess. In *Advances in Neural Information Processing Systems 7* (pp. 1069–1076). Cambridge, MA: MIT Press.

Thrun, S. & Schwartz, A. (1993). Issues in using function approximation for reinforcement learning. In Mozer, M., Smokensky, P., Touretzky, D., Elman, J., & Weigand, A. (Eds.), *Proceedings of the 4th Connectionist Models Summer School, Pittsburgh, PA, 2–5 August* (pp. 255–263). Hillsdale, NJ: Lawrence Erlbaum.

Vollbrecht, H. (2003). *Hierarchical reinforcement learning in continuous state spaces*. Unpublished PhD dissertation, University of Ulm, Ulm, Germany.

Wiering, M. A. (2010). Self-play and using an expert to learn to play backgammon with temporal difference learning. *Journal of Intelligent Learning Systems & Applications*, 2(2), 57–68.

Wiering, M. A., Patist, J. P., & Mannen, H. (2007). *Learning to play board games using temporal difference methods* (Technical Report UU–CS–2005–048, Institute of Information and Computing Sciences, Utrecht University). Retrieved from http://www.ai.rug.nl/~mwiering/group/articles/learning_games_TR.pdf.

Chapter 7
The Tandem Truck Backer-Upper Problem

The tandem truck backer-upper (TTBU) problem is an extension of the single trailer truck-backer upper problem where this problem uses a trailer truck with two trailers instead of one, which is considered to be nearly impossible for humans alone (Widrow and Lamego 2000a). Everything else about the problem, including the control of the truck and the goal of the problem, however, is very similar to that of the TBU. A pure reinforcement learning approach has not be used to solve this problem, however, an assistive control scheme for this problem has been developed by Widrow and Lamego (2000a, b). That work uses Neurointerfaces, a neural network-based control system that merely assists, but does not replace, the truck driver in backing up the tandem trailer. The work described herein is the first to use a simple reinforcement learning approach to begin to learn the tandem trailer-backer upper problem.

The state of the tandem trailer truck at any point in time is defined by five variables: $\mathbf{x} = [x, y, \theta_0, \theta_2, \theta_4]$ (Fig. 7.1). The location of the rear of the second trailer is defined by x and y. The angles θ_0, θ_2, and θ_4 define the orientation of the cab, the first trailer, and the second trailer, respectively, with respect to the horizontal axis. Angles θ_1 and θ_3 can be computed by $\theta_1 = \theta_0 - \theta_2$ and $\theta_3 = \theta_2 - \theta_4$, and these represent the angles between the cab and the first trailer, and the first trailer and the second trailer, respectively. Additionally, the variable u defines the angle of the front wheels with respect to the orientation of the cab. The kinematics of the tandem trailer are given by Riid et al. (2006) and Tanaka et al. (1997) as:

$$\theta_0' = \theta_0 + \frac{v\Delta t}{l} \tan u$$
$$\theta_1' = \theta_0 - \theta_2$$
$$\theta_2' = \theta_2 + \frac{v\Delta t}{L} \sin \theta_1$$
$$\theta_3' = \theta_2 - \theta_4$$
$$\theta_4' = \theta_4 + \frac{v\Delta t}{L} \sin \theta_3$$
$$y' = y + v\Delta t \cos \theta_3 \sin\left(\frac{\theta_4' + \theta_4}{2}\right)$$

C. Gatti, *Design of Experiments for Reinforcement Learning*,
Springer Theses, DOI 10.1007/978-3-319-12197-0_7

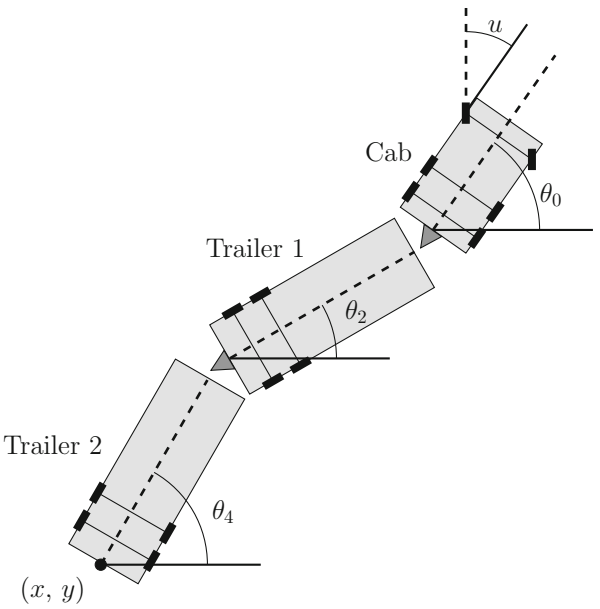


Fig. 7.1 The state of the tandem trailer truck is defined by the rear trailer position (x, y), the cab angle θ_0, the first trailer angle θ_2, and the second trailer angle θ_4. The direction of the cab wheels is defined by u.

$$x' = x + v\Delta t \cos\theta_3 \cos\left(\frac{\theta_4' + \theta_4}{2}\right)$$

where $v = -1$, $\Delta t = 0.1$, $L = 5$, and $l = 2$.

The general problem and goal for the tandem trailer problem is basically the same as that for the single trailer problem: to back up a trailer truck to a specific location and orientation by controlling the orientation of the front wheels. As with the single trailer problem, the truck can only back up, and forward actions are not allowed in this implementation. The increased difficulty of adding another trailer is quite significant however, and the tandem trailer problem is a very challenging control problem, especially considering that the methods we employ are straightforward and relatively simple with no heuristics. We therefore believe that a sequential training paradigm is most appropriate and reasonable for learning this problem. By a sequential training paradigm, we are referring to a procedure in which the problem is learned in stages, where in each sequential stage, the learned control scheme becomes more and more efficient and/or precise. An example of this could be where in each training stage, the tolerances to the goal in terms of the desired location and orientation of the truck, are reduced. Note that in this work, however, we focus on only the first phases of such a sequential training scheme, such that the truck is only required to back up to a specific location with a relatively loose tolerance. We consider a full sequential training procedure to be a heuristic, which is beyond the scope of this work, and we therefore reserve full development of a sequential scheme for future work.

7.1 Reinforcement Learning Implementation

The tandem trailer problem can be formulated as a reinforcement learning problem just as with the single trailer problem. The trailer truck must be backed up from a random initial position and orientation by controlling the orientation of the front wheels of the cab. When the truck reaches the goal (within some tolerance), a positive reward is provided indicating that actions in that episode were useful. Alternatively, when the truck does not reach the goal, negative feedback is provided indicating that actions in that episode were not beneficial. The truck may not reach the goal for a number of reasons, such as if the truck jack-knifes at either hinge, the maximum number of time steps is reached, or if state-space constraints are violated (i.e., the truck is driven far outside of an acceptable space).

Table 7.1 provides specific problem characteristics and settings for this problem. At the beginning of each episode, the position of the rear of the truck was set to $(x, y) = (50, -10)$. The cab angle θ_0 was randomly initialized over $U[-0.5, 0.5]$, and the other trailer angles θ_2 and θ_4 were also set to this angle. This initial state amounts to the truck being in a random orientation, but with the cab-trailer and trailer-trailer hinges straight. The front wheels of the cab could be oriented to nine different directions, which allows for much greater control compared to the TBU problem. However, the greater number of actions also means there are more choices, which increases the complexity of learning a correct control strategy.

The reward scheme for this problem is similar to that of the TBU problem. However, as we take the view that this problem could be solved in multiple learning phases, we are only initially interested in learning very general behavior. We therefore specify that the goal of this first learning phase only be to back the truck to a specific location. When the trailer truck reached the goal location at $(x, y) = (0, 0)$ (within a rather loose tolerance of $d = \sqrt{x^2 + y^2} \leq 10$), a reward of $+1$ was provide as positive feedback. While this is somewhat removed from requiring that the path of the truck be constrained by physical barriers or that the truck be in a specific orientation at the goal, the goal used herein requires that actions be learned that avoid the tandem trailer truck from jack-knifing at either hinge, which is arguably one of the most difficult parts of this task.

If, during an episode, the trailer truck jack-knifed, exited the domain boundaries, or the maximum number of time steps was exceeded, feedback of -0.3 was provided. The angles between the cab and the first trailer and the first and second trailers, θ_1 and θ_3, respectively, were computed at every time step, and if either of these angles were outside of $\left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$, the truck had jack-knifed and the episode was terminated. The domain boundaries were set to $x, y = [-100, 100]$, and if at any point the truck exited these boundaries, the episode was also terminated. The reward and penalty values were chosen somewhat based on trial and error during initial domain development, but also based on requiring that (naturally) the reward be positive and the penalties be negative, and that the magnitude of the reward (which occurs less frequently) be greater than the penalties.

Table 7.1 Domain characteristics for the tandem truck backer-upper domain.

Initial conditions	$x = 50$
	$y = -10$
	$\theta_0 = U[-0.5, 0.5]$ (radians)
	$\theta_2 = \theta_4 = \theta_0$
Actions	9: $[0.0, \pm 0.1, \pm 0.3, \pm 0.5, \pm 0.7]$ (radians)
Rewards/penalties	Achieve goal: $+1$
	Exit domain boundaries: -0.3
	Trailer-cab or trailer-trailer jack-knife: -0.3
	T_{max} exceeded: -0.3
Reward function per time step	0.05 if $\|(x', y')\| < \|(x, y)\|$
	-0.05 if $\|(x', y')\| \geq \|(x, y)\|$
Goal tolerance[a]	$d \leq 10$
Number of episodes	10,000
Number of time steps per episode (T_{max})	1000
Performance time window (p_{win})	100
$conv_{val}$	0.5
$conv_{rng}$	0.005
$conv_m$	1×10^{-4}

[a] $\delta\theta_T = \|\theta_T^* - \theta_t\| \in [-\pi, \pi]$; $d = \sqrt{x^2 + y^2}$

At each time step, a reward function based on two subsequent state values (i.e., a relative reward function) was used to provided frequent feedback, rather than a reward function that was based on the absolute state values, as was used in the mountain car and TBU problems. The reward function was based on the difference in the distance to the goal between one state (x, y) and its subsequent state (x', y'). Let the distance to the goal at one time step be $d = \|(x, y)\|$ and the distance to the goal at a subsequent time step be $d' = \|(x', y')\|$. When $d' < d$, positive feedback of 0.05 was provided; when $d' \geq d$ negative feedback of -0.05 was provided. This is a similar type of reward function that was used in the mountain car problem where if the truck gets closer to the goal, this is generally seen as a good thing, and if the truck gets further away from the goal, this is generally seen as a bad thing. We acknowledge that this is a simple reward function that doesn't take into account the orientation of the truck or the angles of the truck hinges, though exploring how the reward function affects learning is beyond the scope of this work, but the reward function could be explored using the same methods presented herein.

A three-layer neural network (one hidden layer) was used to learn the value function for this problem. In the single trailer truck backer-upper problem, the number of nodes in the hidden layer was a variable included in the experimental design. However, in this problem, we fixed the number of hidden nodes at 51; this number of hidden nodes was selected based on the performance in the single trailer problem

Table 7.2 Variables and their associated ranges used in sequential CART for the TTBU problem.

Variable	Description	Range	RL component
α_{mag}	Base (input-hidden layer) learning rate	[0.0001, 0.01]	Neural network
α_{ratio}	Learning rate ratio	[2.0, 5.0]	Neural network
λ	Temporal discount factor	[0.4, 0.7]	TD(λ) algorithm
γ	Next-state discount factor	[0.96, 0.99]	TD(λ) algorithm
ϵ	P(action exploitation)	[0.85, 0.97]	TD(λ) algorithm

and based on preliminary testing. We do not claim that 51 nodes is an ideal number of nodes in the hidden layer, and experimentation including this as a variable could be performed, but removing this variable reduces the complexity of the experimentation. The input layer had five nodes (for the five state variables), and the output layer had nine nodes (for the nine possible actions). Prior to passing the state into the neural network, the state variables x and y were scaled to $[-0.1, 0.1]$ (relative to the domain bounds of $x, y = [-100, 100]$) so that they were on a similar scale as the truck angles θ_0, θ_2, and θ_4.

7.2 Sequential CART

We investigated the convergence in the TTBU domain for five different parameters of the neural network and of the TD(λ) algorithm. The parameters and their initial ranges are shown in Table 7.2. These parameter ranges are slightly smaller than those used in the previous problems, and this was done to have a slightly more focused experiment due to the longer simulation times of the TTBU domain. The ranges of these parameters were chosen based on prior experience with these methods, and with little excess range on either end of the generally used parameters, which still results in a rather large parameter space.

The parameters used in the sequential CART modeling are shown in Table 7.3. Each design point evaluated using sequential CART consisted of having the agent attempt to learn the TTBU domain in 10,000 episodes, where an episode consists of one attempt at backing the tandem trailer truck to the goal location. The initial experimental design therefore consisted of 125 unique design points, with 3 replicates each, totaling 375 initial runs, which was generated using Latin Hypercube sampling (LHS). Subsequent designs for each iteration of the sequential CART algorithm were also LHS designs, consisting of 25 design points, again with 3 replicates each, for a total of 75 runs. Only three iterations of sequential CART was used because we allowed for fewer design points to fall into each leaf. When fewer design points are allowed to fall into each leaf, the CART model will often have more leaves, which results in more parameter subregions to explore in subsequent iterations, thus increasing the computation time.

Table 7.3 Parameter settings for sequential CART modeling and CART models.

Description	Value
Number of variables	5
Initial number of design points	125
Replications per design point	3
Number of sequential CART iterations	3
Number of additional design points per *low*-leaf	25
Sample design method	Latin hypercube
Minimum design points per split[a]	$\lfloor 0.02\|\|y\|\| \rfloor$
Minimum design points per leaf[a]	$\lfloor 0.01\|\|y\|\| \rfloor$
Minimum complexity change per split	0.001
Maximum number of surrogates	5
Proportion of low points per leaf	0.8
Convergence proportion threshold	0.9
Minimum convergence proportion per convergent domain	0.05
Minimum design points per convergent domain	5

[a] $\|\|y\|\|$ is the length of the response vector for the current iteration

7.2.1 Convergent Subregions

The tandem truck backer-upper problem is significantly more challenging than the other problems included in this work, and this is because it has a higher dimensional state space and control of the truck is rather fragile. These domain features make learning a consistently successful control strategy through reinforcement learning considerably more difficult. With regards to the current work, this likely translates into convergent parameter subregions that are quite small or that convergence for the same parameters is unstable and variable. This was observed during the experimental runs for the sequential CART modeling.

In the mountain car and single trailer truck backer-upper problems, the performance measure used in sequential CART modeling was based on the number of time steps required to reach the goal after the empirical convergence criteria had been achieved, and the convergence indicator was based on whether or not the empirical convergence criteria had been satisfied. In the tandem truck backer-upper problem, however, it is very infrequent that the empirical convergence criteria is satisfied using the same convergence parameters as in the other problems. In light of the difficulty of this problem, and considering that, in this initial learning phase, we are only interested in learning a very general control strategy, we relax the convergence criteria and change the performance metric used in sequential CART modeling.

Thus, rather than using the number of time steps to the goal at convergence, we define a performance metric $perf = 1 - \max\left(p_{goal}\right)$ where $\max\left(p_{goal}\right)$ is the maximum moving average of the proportion of times the goal is reached over all

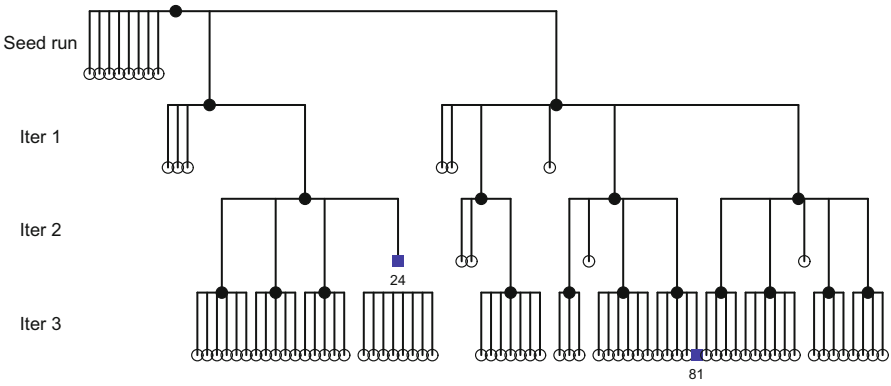


Fig. 7.2 Sequential CART process for the TTBU problem. Parameter subregions that are candidates for further experimentation are shown as *black* circles, subregions that are pruned from further experimentation are shown as *open* circles, and convergent subregions are shown as *blue* squares.

episodes. This measure is converted (from the $1-$) to a metric where a lower value is better because the sequential CART approach as defined herein is based on finding regions with lower performance values, although the sequential CART procedures could be modified. Additionally, due to the fragility of this problem, p_{goal} often does not remain high for an extended period of time in order to fulfill the empirical convergence criteria. The convergence indicator in this problem is therefore simply $conv = \left(p_{goal} > 0.9\right)$; in other words, if p_{goal} reached at least 90 % at any point during the learning run. Note that p_{goal} is still computed using a window of the learning runs and is not an instantaneous value.

Parameter subregions of the sequential CART process were defined as leaf nodes if the convergence proportion (i.e., the proportion of design points in the subregion with a true RL convergence indicator) of subregion was greater than 50 %. We reduced the threshold convergence proportion per leaf node relative to the other problems explored in this work because the TTBU domain is so challenging. Additional screening of the leaf nodes required that there be at least 20 points in the subregion. Figure 7.2 shows the sequential CART process for the TTBU problem. While many parameter subregions were explored, in three iterations of sequential CART, only two convergent subregions were identified. Figure 7.3 shows the parameter ranges for each of these convergent subregions. Table 7.4 shows summary statistics of these subregions, and Table C.5 in the Appendix provides the numerical values of these subregions.

Both of the convergent subregions did not partition on the parameter ϵ, which is interesting compared to the TBU problem that required higher values of ϵ. We find that λ can range from approximately 0.5–0.7 (for both subregions), although the allowable range of λ for the experimental design was [0.4, 0.7]. Regardless, the convergent range for this parameter is different than what was found for the TBU, which requires λ to be set below approximately 0.3. However, we should note that in the TTBU problem, the bounds of the experimental design for λ did not extend

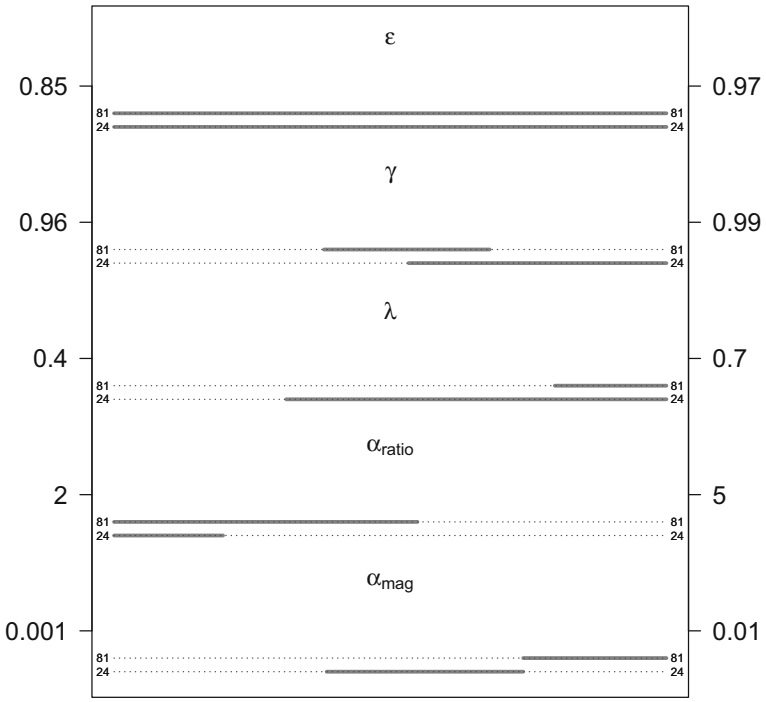


Fig. 7.3 Parameter ranges for the convergent subregions for the TTBU problem.

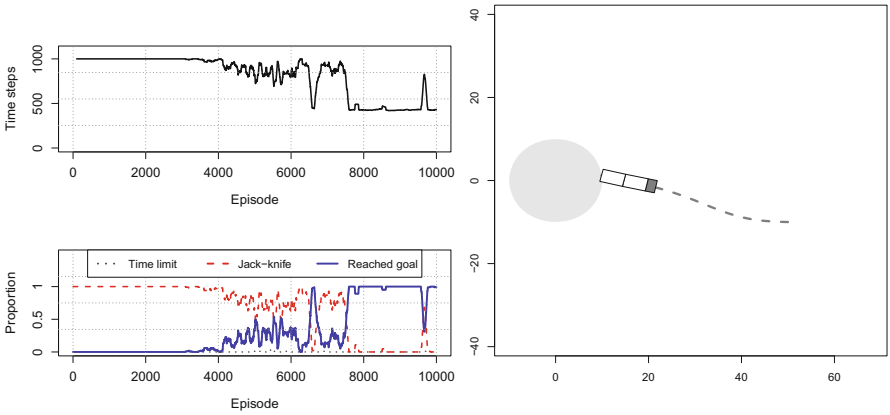
Table 7.4 Summary statistics for the convergent subregions for the TTBU problem.

Subregion #	Iteration	p_{conv}	No. points	Dim. ratio	Ave. radius	Sum radii
24	2	0.500	42	5.068	0.271	2.708
81	3	0.667	57	4.950	0.231	2.311

below 0.4, so it is possible that lower values of λ could work for this domain, though we cannot say for sure.

The magnitude of the base learning rate α should be set rather high, which is consistent with the TBU problem, however, the ratio of the learning rates can be a bit lower than for the TBU problem. A possible explanation for the differences in the convergent parameter ranges between the the TTBU problem and the TBU problem could be that there are rather loose reinforcement learning convergence bounds (both for the reinforcement learning problem, i.e., goal tolerances, and for what is considered to be a convergent reinforcement learning run) in the TTBU problem compared to the TBU problem, and thus a refined control strategy is not required. This may also explain the wide acceptable range of ϵ.

An example learning run from within subregion 81 is shown in Fig. 7.4. This run reaches a steady state performance at around episode 8000 at which point the moving proportion of reaching the goal stays near 1 (Fig. 7.4b). The trajectory of the truck



a Performance and episode termination proportions. **b** Example trajectory after learning.

Fig. 7.4 Example performance from a sample in subregion 81, with parameters: $\alpha_{mag} = 8.36 \times 10^{-3}$, $\alpha_{ratio} = 2.204$, $\lambda = 0.695$, $\gamma = 0.978$, $\epsilon = 0.851$. The *top* plot on the *left* shows the moving average of the number of time steps to the goal, and the *bottom* plot shows moving averages of the proportions of how episodes terminated. The figure on the *right* shows the trajectory of the trailer truck backing up after learning had converged. The *gray shaded* region in this figure indicates the acceptable error region that the truck must reach.

is from a test run after learning had converged and shows a direct and smooth path to the goal.

Regional sensitivity analysis (Fig. 7.5) suggests that the three reinforcement learning parameters, λ, γ, and ϵ, have strong effects on convergence, whereas the magnitude and ratio of the learning rates matter very little. The differences in the interpretation of the RSA plots and the convergent parameter ranges produced by CART modeling is likely due to the variability in the responses (convergent or non-convergent) for the learning runs, as well as the fact that CART modeling only chooses the best partition, based on the available data, into convergent and non-convergent runs. Still, the general shapes of the RSA plots are consistent with the convergent parameter ranges.

7.3 Discussion

This work is the first of its kind to use a pure reinforcement learning approach to learning the tandem truck backer-upper problem. While we do not solve this problem completely such that the trailer truck can be controlled to back up to a very specific position and orientation from any initial state, we are able to learn how to control the truck for a relaxed case of this problem. Even when learning to control the truck for this relaxed case, we find the knowledge that is learned includes how to avoid jack-knifing the truck, as is evident by the truck reaching the goal location,

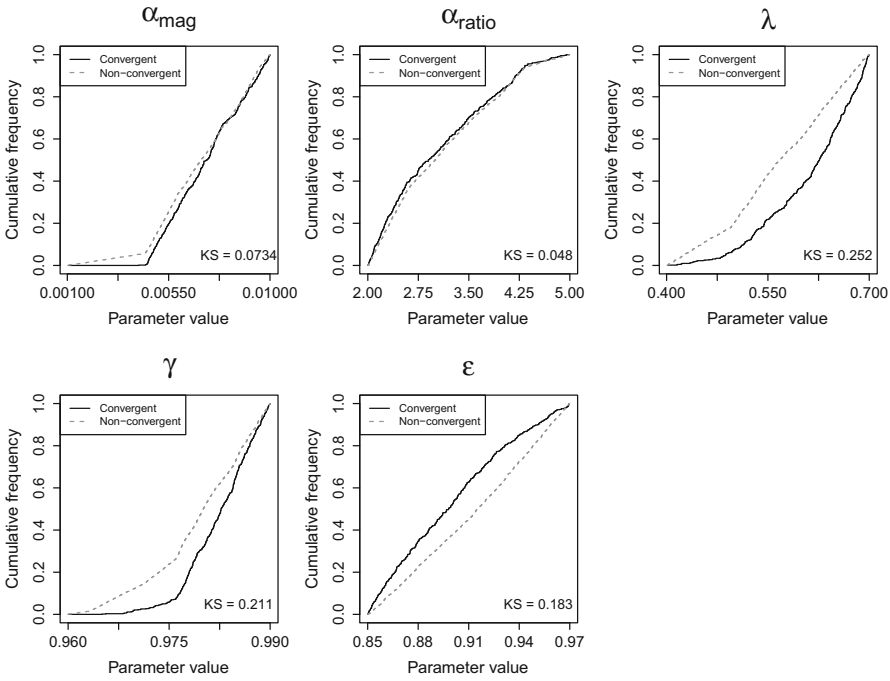


Fig. 7.5 Regional sensitivity analysis based on convergence of all experimental runs for the TTBU problem.

which is essential to solving this problem. Furthermore, we find learning algorithm parameter ranges that have good performance in being able to control the truck to the goal location, which is a notable achievement. While we'd like to have a robust controller that can be used more generally, technically, all that is needed for an implementation of this is one convergent learning run such as that shown in Fig. 7.4.

Due to the challenging nature of this problem, and partially due to the basic reinforcement learning strategy used, we believe that a sequential learning approach is essential to have a refined controller. Thus, the work we present consists of perhaps the first stage of this sequential learning process. Subsequent training could be used to improve the current controller in a number of ways. The tolerances on the goal location (and orientation) could be reduced in order to back up the truck to a more specific location. The ability of the controller could be improved so that it can generalize to successfully back up the truck from anywhere in the domain. These approaches could be used in stages or used in an adaptable learning procedure that refines the goal tolerances or increases the state space coverage based on the current learning performance. In any case, regardless of the sequential training approach, in each of these stages the network weights from the previous training stage would be used as the starting weights for learning in subsequent stages. We note that it is always possible that a new network with random weights could also be trained

to learn more challenging variants of this problem, though we believe that starting with weights from a more general problem would likely be a more efficient training scheme.

In the other domains investigated in this work, convergence was relatively frequent, especially compared to the TTBU domain. Hence, we could use a moving average (over a number of episodes) of the number of time steps to the goal as a performance metric. In the TTBU domain, however, convergence is very infrequent, and using a similar moving average metric would yield too few points to create a CART model. This motivated the use of an alternative performance metric. Creating CART models from the same moving average performance metric as in the other domains would often only result in a tree stump (root node) and not a tree. The alternative performance metric that we used was based on the maximum of the moving proportion of reaching the goal. Values closer to 0.0 indicated very good performance, and values closer to 1.0 indicated very poor performance where little or nothing was learned. By using this metric in the sequential CART procedure, we were able to find two parameter subregions that had good learning performance, though not perfect performance. The use of an alternative performance metric shows the flexibility of sequential CART such that it can be tailored to the characteristics of the responses of the problem at hand.

Kriging metamodels were created for the mountain car and TBU problems, however, we did not do this for the tandem truck backer-upper problem. The primary reason for this is because we do not consider the problem to be solved, and any metamodel would not be an accurate representation of the true performance. Additionally though, we found that the responses (i.e., number of time steps to the goal) for the same design points were extremely variable, and that a metamodel could not accurately model the response surface. This variability could be due to the fact that the problem was relaxed with loose goal tolerances, and it is possible that an accurate metamodel could be created after additional training is used to create a more accurate controller.

References

Riid, A., Leibak, A., & Rustern, E. (2006). Fuzzy backing control of truck and two trailers. In *Proceedings of the IEEE International Conference on Computational Cybernetics (ICCC), Budapest, Hungary, 20–22 August* (pp. 1–6). doi: 10.1109/ICCCYB.2006.305690

Tanaka, K., Taniguchi, T., & Wang, H. O. (1997). Model-based fuzzy control for two-trailers problem: Stability analysis and design via linear matrix inequalities. In *Proceedings of the 6th International Conference on Fuzzy Systems, Barcelona, Spain, 1–5 July* (pp. 343–348). doi: 10.1109/FUZZY.1997.616392

Widrow, B. & Lamego, M. M. (2000a). Neurointerfaces: Applications. In *IEEE Adaptive Systems for Signal Processing, Communications, and Control Symposium, Lake Louise, Alberta, Canada, 1–4 October* (pp. 441–444). doi: 10.1109/ ASSPCC.2000.882515

Widrow, B. & Lamego, M. M. (2000b). Neurointerfaces: Principles. In *IEEE Adaptive Systems for Signal Processing, Communications, and Control Symposium, Lake Louise, Alberta, Canada, 1–4 October* (pp. 315–320). doi: 10.1109/ASSPCC.2000.882492

Chapter 8
Discussion

This work should be viewed as basic science for reinforcement learning. That is, we consider a basic reinforcement learning algorithm and implementation and we attempt to better understand when learning is possible. We showed that there are specific and unique parameter subregions for which reinforcement learning frequently converges, in spite of the very challenging underlying problem of reinforcement learning with a neural network. This type of analysis has not been performed before and thus the results are novel. By exploring the parameter space of the learning algorithm and the neural network, we found that these convergent parameter subregions are different for different problems, and this is likely due to the underlying differences in the problem characteristics. Furthermore, we found differences in the convergent parameter ranges from what is typically recommended for parameter settings in the literature. However, additional experimentation would be required to better understand exactly why there are differences in the convergent parameter subregions between problems.

This work is the product of the combination of two fields: reinforcement learning and design of experiments. Furthermore, as was mentioned earlier, this work is as much about reinforcement learning as it is about design of experiments, and there are novelties and contributions in this work for each of these fields. Aside from our previous work combining these fields (Gatti et al. 2013), there is really no other work that uses the design of experiments paradigm to understand reinforcement learning, though there are some applications of design of experiments to the broader field of machine learning. There are numerous points to consider when viewing this work, from the perspective of both of these fields, as well as concerning our methodology applied, and this chapter discusses some of these points.

8.1 Reinforcement Learning

The TD(λ) learning algorithm is one of the fundamental learning algorithms used in reinforcement learning. Although it was popularized in the 1980s (Sutton and Barto 1998), it gained considerable popularity after its great success in the game

C. Gatti, *Design of Experiments for Reinforcement Learning,*
Springer Theses, DOI 10.1007/978-3-319-12197-0_8

of backgammon (Tesauro 1990, 1994), and it is still used by practitioners (Dann et al. 2014). Despite its use, questions remain concerning why learning occurs under some problems domains and learning is limited or does not occur in other domains. Furthermore, our understanding of this algorithm when paired with a neural network is even more limited. The inconsistencies in the success when using the TD(λ) algorithm with a neural network, as well as both successful and unsuccessful personal experiences with these methods, were the motivation for this work.

8.1.1 Parameter Effects

The temporal discount factor λ essentially scales the influence of weight updates from previous time steps, and it therefore attenuates how far information (in terms of weight changes) is propagated through time. In general, the literature suggests that setting this parameter between 0.6 and 0.8 works well in many applications (Tesauro 1992; Patist and Wiering 2004; Wiering et al. 2007; Wiering 2010), and there is little work that explores setting this to other values (Sutton and Barto 1998). In the mountain car problem, we found that λ could take on values over [0, 1] over all convergent subregions, though each individual subregion had a somewhat smaller (but still large) range of acceptable values. In the TBU problem, λ had to be quite consistently on the lower end of [0, 1] for all convergent subregions. In the TTBU problem, although experimentation evaluated a smaller range of λ, this parameter had to be between [0.5, 0.7] for the two convergent subregions. Our results are therefore both consistent and inconsistent with what is recommended in the literature.

The convergent parameter subregions for the action exploration-exploitation parameter ϵ were also inconsistent for the three domain problems. The mountain car domain had both small and large ranges for ϵ depending on the subregion, with some of the lower bounds of these subregions extending down to 0.6, which results in nearly as much action exploration as there is exploitation. In the TBU problem, the parameter ranges for ϵ were also consistent and similar across the convergent subregions (similar to the mountain car problem), and ϵ was required to range from approximately 0.9– 1.0 (across all subregions). Practically, this means that the majority of the time, knowledge about which actions to take was learned from trying actions that were perceived to be useful based on the current knowledge of the agent, rather than exploring other, non-knowledge based actions. In the TTBU problem, ϵ could range over the entire parameter space of [0.85, 0.97] that was evaluated. The differences between the specificity of ϵ in the TBU and TTBU problems could be due to the fact that the goal criteria in the TBU problem was stricter, and due to the larger state- and action-space of the TTBU problem that required more exploration of the actions.

The next-state discount factor γ attenuates the value of the next state value in the TD(λ) algorithm. Relative to λ and ϵ, γ is paid little attention in the literature, and some work even neglects including it in the description of the TD(λ) algorithm (Tesauro 1992). Some work suggests that no learning will occur if $\gamma = 1$

(Thrun 1995), although other work suggests that learning can be successful when $\gamma = 1$ (Ghory 2004). We find that, for the mountain car and the TBU problems, γ should range approximately over [0.96, 0.99], whereas for the TTBU problem, γ should range approximately over [0.97, 0.99]. In other words, slight next-state value discounting seems to be useful across all of these problems.

The ranges of the magnitude (α_{mag}) and ratio (α_{ratio}) of the learning rates of the neural network for the convergent subregions in the mountain car problem were found to span almost the entire original parameter space for these parameters. In the TBU and TTBU problems, the range of the magnitude of the learning rates should generally be set closer to 0.01. The ratio of the learning rates in the mountain car problem were found to range over approximately [2, 8] in the convergent subregions. In the TBU problem, the learning rates could range over [1, 8], and in the TTBU problem, this ratio could range over [2, 3.5] in the convergent subregions. Intuitively, a larger ratio of the learning rates between the layers (where layers closer to the input layer have larger learning rates) makes sense, as the error gradients at layers closer to the input layer are generally smaller, which is suggested to be useful in the non-reinforcement learning literature (Embrechts et al. 2010). The results of this work, considering all three problems together, suggest that a learning rate ratio of greater than 1 could aid in convergence, however, additional experimentation would be required to answer this for sure. We also evaluated whether the number of nodes in the hidden layer of the neural network had an effect on learning in the TBU problem. Interestingly, we found that a minimum of about 26 hidden nodes was required for the convergent subregions, suggesting that the neural network needs a minimum level of complexity to be able to learn this task.

Looking at the convergent parameter subregions for each problem at a higher level, we see that the mountain car problem had rather wide subregions. This suggests that this problem is relatively simple and that very specific parameter ranges are not required for learning to be successful. In the TBU problem, we found that there are very specific and consistent parameter subregions where learning can occur. Similarly, the TTBU problem was also found to require relatively smaller parameter subregions for learning to occur, though we note again that the original parameter space of the experiment was smaller than in the other problems as well.

If practitioners merely use parameter settings that are recommended in the literature, they may very well choose parameter combinations which do not fall within convergent parameter subregions. We note though, that we do not guarantee that the convergent parameter subregions found in this work are the only subregions that allow for successful reinforcement learning. The identification of these convergent subregions is dependent on the experimentation, and this will be discussed later. Additionally, small changes to the problem characteristics (i.e., action space, reward structure, problem dynamics, etc.) would likely change the convergent parameter subregions. We believe that there should be some consistency among parameter subregions for similar types of domain problems, however, additional work exploring the effects of altering domain characteristics would be required to support this claim.

In the mountain car and TBU problems, we explored the performance of reinforcement learning using kriging metamodels. The influence of the parameters in

the convergent subregions in the mountain car problem were found to vary considerably over the different subregions. In other words, there was no consistency in the influence of parameters across the subregions, and thus the behavior of the response surfaces are likely quite unique for the different subregions. Furthermore, the interactions among parameters in the FANOVA graphs are quite different among the subregions. In the TBU problem, however, the influence of parameters across the subregions was almost completely consistent, such that γ had an overwhelming influence on the variability of the performance of reinforcement learning, and all other parameters had very little influence. Based on the FANOVA graphs, there is also some consistency among parameter interactions as well. The convergent parameter subregions for the TBU problem have very consistent ranges across all subregions. Additional analysis on the ranges of the convergent subregions showed that none of these regions are immediately adjacent to each other. However, they may be close to each other, and this fact, along with the consistent parameter sensitivity profiles, suggests that these subregions come from a similar portion of the parameter space. The division of this portion of the parameter space into smaller subregions may either be due to the fact that there really are separate convergent subregions or due to the sampling of the experimental design, and this could be answered with a finer experimental design in this portion of the parameter space.

It is possible that there are convergent parameter subregions, or unique characteristics of the learning process, that are consistent for different classes of problems, but that may be different across problem classes. The intuition behind this is consistent with our belief that the success of reinforcement learning is dependent on the domain characteristics of the problem. To be able to know for sure if different problems classes share similarities, many more domains would have to be explored, which would result in a significant knowledge repository for the reinforcement learning community.

The problems explored in this work could all be considered to be vehicle control-type problems. With only exploring three different problems, one of which being a relaxed version of the true problem, it is difficult to give certain guidance on parameter settings, especially considering that finding consistencies across domains was not a primary goal of this work. However, based on the convergent subregions for the problems explore, γ should generally be high ($\sim$ 0.96–0.99) as should ϵ ($\sim$ 0.85–0.97). For λ, the only guidance we can give is not to simply follow what is typically used in the literature ($\sim$ 0.5–0.7). In terms of the neural network, we would suggest setting the ratio of the learning rates to at least 2, and using a base learning rate of close to 0.01. We should stress that these suggestions should only be used as starting points for the parameter settings, and that some exploration of these parameters around these ranges will likely be required for different problems.

The current work used only the bare bones TD(λ) learning algorithm with no additional heuristics. This was done primarily to determine what types of problems this learning algorithm could learn by itself. Heuristics are an effective way to improve the performance of a base learning algorithm by tailoring or customizing the learning process to the problem at hand. While the mountain car and the single trailer truck backer-upper problem were learned successfully using only the TD(λ) algorithm,

the tandem truck backer-upper problem could not be learned without some relaxations. We suggest that a particular heuristic, a sequential learning procedure, is likely needed to learn this problem better, such that the work we presented could serve as a base that could be refined with subsequent training. Additionally, a heuristic that prevents unlearning (i.e., effectively decreases in performance with additional training) would likely also be required, as learning is not guaranteed to increase monotonically with the methods used herein. One possible approach to prevent unlearning may be to save the network weights only when performance increases, and then to use these saved weights as a starting point when performance decreases. While these ideas for heuristics seem reasonable, only in an actual implementation would we be able to determine exactly how they should be implemented, and this could be the focus of future work.

8.1.2 Neural Network

This work used a neural network for the representation, which is not the first choice for a representation by many in the field of reinforcement learning, though its use in this manner is quite elegant. A neural network was chosen primarily because this was the type of representation that was used in very early work by myself. As mentioned in Sect. 2.2.2, there are numerous types of representations than can be used for reinforcement learning. Neural networks are attractive because they can approximate complicated functions, however, there is no guarantee on their convergence, and training can be challenging. This work has shown, however, that neural networks can successfully learn how to perform challenging tasks under the right conditions and parameter settings.

There are many applications of reinforcement learning that use other representations, though we cannot comment on their implementation from experience. An attractive alternative to a neural network is a function approximator that is powerful but that is not plagued with convergence issues, such as partial least squares (PLS). When a neural network is used with TD(λ), the weight update equations incorporate the TD(λ) learning algorithm. The primary limitation in applying PLS to reinforcement learning is that the PLS model must somehow be built based on the temporal difference learning paradigm, for which there is currently no extension. However, if we recall that the goal of the PLS model would be to approximate the value function (a mapping between states and their values), perhaps the underlying value function could be created without explicitly having to rely on the temporal difference learning paradigm.

In some subregions, we found that the reinforcement learning can be extremely variable and kriging metamodeling cannot approximate the response surface well based on the number of design points evaluated. Aside from the stochastic nature of reinforcement learning, the only other variable element in the reinforcement learning implementation is the random initial weight of the neural network. Network weights were initialized by sampling over a uniform distribution $U[-0.1, 0.1]$. A weight

initialization scheme that is based partially on the number of connections to a node has been found to be useful for supervised learning (LeCun et al. 1998). However, preliminary testing in these reinforcement learning domains suggested that uniform weight sampling performed just as well, and uniform weight sampling has been used in other successful applications reinforcement learning (Wiering 1995; Patist and Wiering 2004; Gatti et al. 2011a).

8.2 Experimentation

The methods used herein are a mix of contemporary experimental design, modeling, and analysis methods. These approaches were selected after careful investigation of any method that might prove to be useful in providing the specific information desired. The methods that were chosen generally grew out of the field of computer experimentation rather than traditional design of experiments. We used a battery of analysis techniques to understand the reinforcement learning problems, including regional sensitivity analysis, global sensitivity indices, FANOVA graphs, and contour surface projections. In most cases, we found that the conclusions from these different methods were relatively consistent, and this gives us confidence in our findings, especially in this challenging experimental problem. The procedures we used in this work are by no means the only methods that could obtain the information we desired, and other approaches could certainly be used.

Those who are familiar with the field of design of experiments know the utility and type of information that conventional methods can provide (e.g., main and interaction effects from ANOVA). The information resulting from the methods used in this work is not better nor worse, yet it is different, which is partially a result of the difficulty of the underlying problem. One of the major goals of this work was to find convergent parameter subregions, and subsequent analysis of those subregions could be done in a number of ways. Thus, more conventional experimentation using factorial designs and ANOVA could be used to provide more standard information on the parameters for the problems at hand. This could be done in future work to evaluate the consistency of the results across different modeling and analysis methods.

While the field of computer simulation and experimentation is not new, these types of problems may have unique qualities for which there is no concrete guidance. Take, for example, the number of design runs to perform. There are many cases in which computer simulations are very time-intensive (e.g., a single run may take 24 hours), and thus the computational budget is a constraint, and efficient experimentation is paramount. In our case, however, each reinforcement learning run is relatively cheap. A learning run for the mountain car problem takes less than 1 second, the TBU problem takes less than about 3 seconds, and the TTBU problem takes at most about 15 seconds. Thus we are not particularly constrained by a computational budget. While this is advantageous in that we can use many design runs, this also does not give us any default guidance on how many design points to run.

For some computer experiments there is some guidance on the number of design points to run, which has been suggested to be proportional to the number of dimensions of the problem as $10d$ where d is the dimensionality of the problem (Chapman et al. 1994; Jones et al. 1998; Loeppky et al. 2009). These studies used different methods than the current work however; for example, the study by Loeppky et al. (2009) is based on the use of a Gaussian process model to approximate the response surface of deterministic simulation model (an ordinary differential equation-based model of ligand activation of G-proteins). Based on preliminary experiments with the reinforcement learning problems explored in this work, we found that an initial design consisting of $10d$ is not sufficient for our application. The number of design points used in this particular application and when using sequential CART should definitely be explored in future work.

A popular area of contemporary design of experiments is concerned with efficient experimentation, especially in the case of experimental runs that may be expensive. For example, in computer experiments, each simulation may required hours or days, and in physical experiments, each experimental run may have a non-significant monetary cost. The goal of efficient experimentation is to gain as much knowledge (with respect to the goal or purpose of the experiment) about the underlying process using the fewest number of experimental runs. Naturally, there is a trade-off between the amount or quality of knowledge gained and the experimental cost, and the trade-off is domain specific.

The purpose of the work described herein was not efficient experimentation. This was, first and foremost, because very little is known about the problems of interest. Secondly, the only cost of each experimental run was computation time. Thirdly, the methods used to dissect the problems at hand are novel (e.g., sequential CART) or relatively new (stochastic kriging). We therefore took the view that gaining an understanding of how reinforcement learning behaves under different conditions and parameters is of greatest importance. Future work could certainly investigate improving the efficiency of these methods and how they respond to different experimental problems, not only including reinforcement learning.

The goal of finding a set of reinforcement learning parameters that result in successful learning runs could be viewed as an optimization problem. Taking this perspective, established optimization approaches, such as genetic algorithms for example, may be able to find these parameter combinations. Such an approach was not used in this work because our goals were larger than finding single instances of successful parameter combinations. Rather, in this work we are interested in *understanding* the parameter space in terms of the sizes and locations of the convergent parameter subregions, as well as how learning is affected by the parameters within these subregions.

8.2.1 Sequential CART

This work presented the sequential CART procedure, a novel method of finding parameter subregions for which the responses within these regions have specific qualities. As far as it is known, this work introduces and defines this approach. The idea was conceived based on the intersection of what was desired from the experimentation and analysis and from what information CART models can provide. This approach was then refined with some ideas from Robertson et al. (2013), who developed a similar approach for optimization.

Considering that sequential CART is a novel methodology, there is a lot of work to be done in order to fully understand how it behaves with different parameter settings and under different problems. The parameters of this algorithm used in the problems herein were selected based on preliminary testing and from suggestions from related work using CART models for optimization (Robertson et al. 2013). Furthermore, this algorithm is very flexible and extensible such that additional tuning features could be added. The methods employed in this work attempted to use a relatively straight-forward approach to using CART models for finding specific parameter subregions, but future work could also focus on extending this algorithm to improve its performance.

The sequential CART model developed and used in this work used the basic CART model formulation (Breiman et al. 1984), and although the implementation is rather flexible, we were still limited in its functionality. It may be possible to modify the CART model creation algorithm and tailor it specifically to sequential CART modeling, for example, by using both numerical performance values as well as convergence indicators. Another interesting use of the CART models, once a model has been developed, would be to estimate confidence intervals for the convergent subregion boundaries, which could be done by creating CART models based on a predefined structure but that are built on bootstrapped data. This approach is similar to that of random forests, with the exception that, in random forests, the structure (i.e., split variables and split points) of the CART models does not have to be consistent across all trees.

The purpose of the sequential CART procedure is to find convergent subregions. While this procedure exploits and centers on convergent subregions, it must also perform some exploration in order to be sure that convergent subregions are just that, and this exploitation/exploration trade-off is partially based on the proportion of points considered to be *low* points. Additionally, while each reinforcement learning run doesn't require significant amounts of time, they do require at least a few seconds, and exhaustive experimentation is not possible. Consequently, the proportions of convergent learning runs within each subregion are merely estimates based on a rather small set of samples, which may therefore be somewhat erroneous. To ensure convergent subregions are in fact convergent with greater certainty, it may be useful to require more design points to be in a convergent subregion or require a larger proportion of convergent runs. Both of these changes may also result in fewer, or no, convergent subregions found however.

Our problem could be considered similar to that of structural reliability, where the goal is to find the probability of failure of a system or structure which also attempts to find a boundary (which may be a contour line and not axis-aligned boundaries) between two regions. In such a problem, there is typically some measurable quantitative performance value that characterizes the quality of that under study; even if this value is poor or somewhat erroneous, there is always a value. In other words, the code is non-degenerate. However, our problem is different and unique in that in the cases when reinforcement learning does not converge, there is no measurable value of performance. Furthermore, the regions we seek are defined by convergence versus non-convergence, and it is likely that there is not a single value that separates these regions. Thus, methods that are often used for structural reliability cannot be directly applied, though similar concepts could be used to develop a procedure that could help identify convergent subregions.

Another potential use of the sequential CART procedure would be to use it as a method for screening variables, potentially reducing the number of variables to be explored in subsequent experimentation. In some of the problems considered here, some of the convergent parameter subregions extended over nearly the entire original parameter space for individual parameters and for most of the convergent subregions found (c.f., α_{mag} for the mountain car problem and ϵ for the TTBU problem). In these cases, a specific parameter range (as a subset of the range explored) may not be required for convergence. Setting these parameters at their average values would make any subsequent experimentation easier due to the smaller number of variables.

Ideally, the subregions labeled as convergent from the sequential CART procedure would have purely convergent runs. Though, due to the random sampling of the experimental design points and a computational budget, achieving pure subregions was not possible. Increasing the purity of these subregions would require more design points and more replicates for each design point. An obvious extension to the current sequential CART algorithm is to parallelize the experimentation within each iteration of the algorithm, and this would allow for running more design runs and improving the accuracy of our results.

8.2.2 Stochastic Kriging

Stochastic kriging is a rather recent extension of deterministic kriging. There are a number of studies that explore the effects of experimentation and modeling on either stochastic or deterministic kriging, including the experimental design (Chen and Lin 2013), the use of common random numbers in the experimental design (Chen et al. 2012), bootstrap model parameter estimation (Kliejnen 2013), and using gradient estimators to improve the metamodeling (Chen et al. 2013). However, these studies focus on low-dimensional (i.e., 1-D or 2-D) benchmark problems, and it is unknown how these methods extend to high dimensional problems.

The methods employed in the current work, with regards to the experimental design and stochastic kriging metamodeling, could be considered rather straightforward and textbook. This was done because of the lack of knowledge regarding the effects of the many study and model design decisions. Our confidence in the resulting methods and models is supported by analyzing the results using a variety of methods. In doing so, we find commonalities among the conclusions of each method which provides us with consensus conclusions for the questions of interest. Future work, however, should focus on factors related to the experimental design and stochastic kriging modeling, especially in higher-dimensional and challenging problems.

We used a single experimental design to create the metamodels for each of the convergent subregions, however sequential sampling and experimentation schemes have been developed for creating kriging models. However, these sequential and adaptive sampling schemes often have the goal of estimating a contour defining a boundary (i.e., level set) between acceptable and unacceptable regions (Bichon et al. 2008; Ranjan et al. 2008; Pichney et al. 2010; Bect et al. 2012) based on some threshold value, or of finding a minimum or maximum value of a function (Jones et al. 1998; Huang et al. 2006). All of these methods are based on a computing an expected improvement, with respect to some well-defined objective, for candidate design points, and the design point that maximizes the expected improvement is evaluated or simulated. Such methods were tested in the problems investigated in this work to improve the metamodel, and we found that sequential sampling schemes using the above mentioned methods generally resulted in new design points being grouped near the regions of the design space that naturally had a large response variance, and the entire design space would not be evenly evaluated. Our goal of creating the metamodel was to produce a model that was a *global* approximation rather than a model that fit one area of the design space really well. Alternative adaptive sampling schemes could be developed and tailored for this particular application, such as sequential sampling based on the convergence indicator, however, this is beyond the scope of this work and could be investigated in future work.

8.3 Innovations

This work has numerous innovations in the fields of both reinforcement learning and design of experiments. There are also methodological innovations that allow for studying a new type of problem class, that being potentially non-convergent simulations. The innovations from this work are listed below, and we indicate whether each innovation is related to reinforcement learning (RL) or the field of design of experiments (DoE), or if the innovation is methodological (M).

- **Application of design of experiments to reinforcement learning (RL, DoE):** Design of experiments techniques have been used to study machine learning methods, however, the application of these techniques to study reinforcement learning is novel. Our previous work set the groundwork for exploring the parameter

effects in reinforcement learning using simple one-factor-at-a-time experiments (Gatti et al. 2011a) and a full-factorial design (Gatti et al. 2013) in other problem domains. The work presented herein attempted to more comprehensively explore the effects of reinforcement learning parameters in additional domains to gain an understanding of the utility of this learning method.

- **Identification of a new class of scenarios in computer experiments (DoE):** As far as is known, the study of computer simulations that can potentially diverge has not been considered from a design of experiments perspective, and their study in this work is novel. While the problems we study are concerned only with reinforcement learning, there are other fields that use computer simulations that have the potential to not converge; for example, finite element modeling or computational fluid dynamics. The methods developed herein may be useful in those other fields as well.
- **New application area of computer experiments (DoE):** Stochastic computer experiments are relatively commonplace for methods such as finite element analysis. Although some learning algorithms are commonly acknowledged as stochastic algorithms, they are not studied using design of experiments approaches. This work considers learning algorithms to be another method that can be classified and studied as a stochastic computer experiment.
- **Novel domain applications (RL):** This work is the first to apply a pure reinforcement learning approach to learning the tandem truck backer-upper problem. Additionally, along with our brief introductory work that applied a pure reinforcement learning approach to learning the single truck backer-upper problem (Gatti and Embrechts 2014), we are the first to explore this problem as well. Although these problems are *in silico* problems, they are more challenging and closer to real world problems than the classic reinforcement learning benchmark problems, such as the mountain car problem. These domains could be used as new benchmark problems when developing and evaluating reinforcement learning methods.
- **Development of custom reinforcement learning code (M):** The code developed in this work allows for the inclusion of additional domains, learning algorithms, and representations. The addition of problem domains is relatively simple because of the structured and consistent format of the code and because of the extensive infrastructure of accessory functions. The addition of other learning algorithms and representation is also straightforward. Using the code is also simple and follows consistent procedures for each domain and representation, and thus this code could be made available to the reinforcement learning community.
- **Explicit definition and description of domain characteristics (RL):** As far as is known, the domain characteristics described in Sect. 2.2.1 have not been formally defined in the literature. We considered problem domains on a finer scale and explicitly define domain characteristics. The motivation for this is our belief that a successful reinforcement learning application lies at the intersection of specific combinations of domain characteristics and learning algorithm and representation parameter settings.

- **Empirical convergence assessment procedure (RL):** This work introduced a novel empirical convergence assessment procedure that can be used to automatically, with no human intervention or assistance required, determine if a reinforcement learning run has converged. Whereas most reinforcement learning implementations look at only a few runs, the work herein required that thousands of reinforcement learning runs be performed, and thus an automated procedure was essential.
- **Convergent subregion parameter range plots (DoE):** The sequential CART procedure produces a unique set of results, that being parameter bounds in multiple dimensions for potentially multiple convergent subregions. Understanding and comparing these parameter bounds required a novel visualization, which led to the development of the parameter range plots which allow for the visualization of high dimensional (> 2) parameter ranges.

8.4 Future Work

This work integrates two typically disparate fields, reinforcement learning (or more generally, machine learning) and design of experiments. We also introduce a novel methodological procedure, sequential CART for finding subregions within a parameter space that have specific characteristics or qualities. This work should be considered as a start to our overall goal of better understanding reinforcement learning so that it can be successfully applied to real world and challenging problems. As this work is comprised of so many different elements, there are many avenues to explore in future work.

- **Understanding why parameters have certain effects:** With the goal of obtaining a better understanding of the behavior of reinforcement learning, the next logical direction to pursue after having found convergent parameter subregions, as done in this work, would be to investigate exactly why parameters have the effects that they do. As we have seen, parameters can have vastly different effects in different regions of the parameter space for the same problem domain, and their influence on reinforcement learning convergence can also vary across problem domains. This strongly suggests that these differences are due to differences in the domain characteristics. Exploring these domain characteristics using controlled experiments could be done using either modified versions of specific domains of interest or using generalized domains that can be tailored to have specific domain characteristics (Kalyanakrishnan and Stone 2009, 2011).
- **Explore additional learning algorithms and representations:** From a reinforcement learning perspective, it would be interesting to apply the same methodology we used herein to other learning algorithms and other representations. As mentioned, the TD(λ) learning algorithm is the fundamental reinforcement learning algorithm, but there are other algorithms that have been found to be more efficient in some cases because they learn based on different and/or additional information.

Algorithms that would be of interest to evaluate include Q-learning (Watkins and Dayan 1992), Sarsa (and variations) (Sutton and Barto 1998), and policy search methods (Diesenroth et al. 2011). Additionally, evaluating different representations would also be of great interest as other representations do not have the same convergence issues as seen with neural networks, though other representations may also have their own deficiencies.

- **Assess learning quality consistency:** The goals of this work were merely to find convergent parameter subregions and to explore the effects of parameters within these subregions, and thus we took a pure quantitative or objective approach. Reinforcement learning runs could also be assessed from a qualitative perspective, however. Specifically, the shape and features of a performance curve from a reinforcement learning run could be examined. Such features could relate to the smoothness, stability, and the speed of learning. To generalize our findings from a large scale analysis, these features would have to be translated into quantitative metrics, though the analysis would still be aimed at understanding the quality of learning.
- **Explore additional domain problems:** The methodology developed in this work could be applied to any other reinforcement learning problem. The three domain problems used in this work give us an idea of what we could see in terms of consistency and variability among the effects of parameters. However, applying our methodology to other problems, those that are both closely related and not related, may allow us to form generalizations about what parameters work under what conditions. Additionally, in the current work, we only explored single agent domains, but dual agent domains (such as board games) are also very popular in the reinforcement learning community, and understanding the effects of parameters in these cases would also be of great interest.
- **Explore the efficacy of learning heuristics:** The experimental methodology developed and applied herein is not limited to exploring only learning algorithms and representations, but it can also be used to explore the effects of applying heuristics to a reinforcement learning algorithm. Examples of these heuristics include weight saving to eliminate unlearning, experience replay (Lin 1992), and changing or evolving the domain characteristics. Additionally, we believe that a sequential learning approach would be able to improve the performance in the TTBU problem. An initial learning run would be performed to learn general information about the TTBU domain, as in this work, such as learning how to avoid jack-knifing the trailers. Subsequent training runs would be run in order to refine the control of the truck.
- **Better understand sequential CART behavior:** There are also numerous future directions with respect to the experimental methodology that would be of interest. The sequential CART procedure is novel, and while its procedure is conceptually simple, there are numerous parameters that must be set that are either related to the algorithm itself or the CART models. This work used parameters and settings for this algorithm that seemed reasonable in preliminary testing and based on a related work (Robertson et al. 2013), though, admittedly, the dynamics of this

procedure can be better understood. Additionally, the performance of this procedure also depends on the underlying problem being investigated. Future work should therefore explore the effects of algorithm parameters and evaluate the performance of sequential CART on different functions with various characteristics. While this is the first use of the sequential CART procedure, we feel confident in its value for future applications based on its performance in the challenging problems presented in this work.

- **Experimental efficiency:** Experimental efficiency was not a high priority in this work, as we were interested in obtaining the most information about the reinforcement learning problems and we were not really limited by a computational budget. That said, we believe that the experimental procedure could be made more efficient, both for sequential CART and for kriging metamodeling. Improving the sequential CART efficiency could be done by more aggressively reducing the parameter space (i.e., selecting a smaller proportion of points to be considered ω_L), or using additional subregion pruning procedures; these avenues are also related to better understanding the dynamics of sequential CART. It is possible though, that in attempting to increase the experimental efficiency of these methods, the accuracy of the resulting convergent subregions and kriging models may be affected; however, this is not known for certain at this point. As mentioned, however, an obvious future direction is parallelizing this algorithm, which would likely be able to find convergent subregions with greater accuracy and in less elapsed time.
- **Sequential kriging experimentation:** The kriging metamodels were based on a single large experimental design, but sequential experimentation could reduce the total number of design points required. However, as mentioned, preliminary experiments using established resampling techniques resulted in poor candidate design points to be evaluated, and thus a new, possibly application-specific, resampling technique may need to be developed.

References

Bect, J., Ginsbourger, D., Li, L., Picheny, V., & Vazquez, E. (2012). Sequential design of computer experiments for the estimation of a probability of failure. *Statistics and Computing*, 22(3), 773–793.

Bichon, B. J., Eldred, M. S., Swiler, L. P., Mahadevan, S., & McFarland, J. M. (2008). Efficient global reliability analysis for nonlinear implicit performance analysis. *AIAA (American Institute of Aeronautics and Astronautics) Journal*, 46(10), 76–96.

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and Regression Trees*. New York, NY: Chapman & Hall.

Chapman, W. L., Welch, W. J., Bowman, K. P., Sacks, J., & Walsh, J. E. (1994). Arctic sea ice variability: Model sensitivities and a multidecadal simulation. *Journal of Geophysical Research*, 99(C1), 919–935.

Chen, E. J. & Lin, M. (2014). Design of experiments for interpolation-based metamodels. *Simulation Modelling Practice and Theory*, 44, 14–25.

Chen, X., Ankenman, B. E., & Nelson, B. L. (2012). The effects of Common Random Numbers on stochastic kriging metamodeling. *ACM Transactions on Modeling and Computer Simulation*, 22(2). doi: 10.1145/2133390.2133391

Chen, X., Ankenman, B. E., & Nelson, B. L. (2013). Enhancing stochastic kriging metamodels with gradient estimators. *Operations Research*, 61(2), 512–528.

Dann, C., Neumann, G., & Peters, J. (2014). Policy evaluation with temporal differences: A survey and comparison. *Journal of Machine Learning Research*, 15(1), 809–883.

Diesenroth, M. P., Neumann, G., & Peters, J. (2011). A survey on policy search for robotics. *Foundations and Trends in Robotics*, 2(1–2), 1–142.

Embrechts, M. J., Hargis, B. J., & Linton, J. D. (2010). An augmented efficient backpropagation training strategy for deep autoassociative neural networks. In *Proceedings of the 2010 International Joint Conference on Neural Networks (IJCNN), Barcelona, Spain, 18–23 July* (pp. 1–6). doi: 10.1109/IJCNN.2010. 5596828

Gatti, C. J. & Embrechts, M. J. (2014). An application of the temporal difference algorithm to the truck backer-upper problem. In *Proceedings of the* 22nd *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN), Bruges, Belgium, 23–25 April*. Bruges, Belgium: ESANN.

Gatti, C. J., Embrechts, M. J., & Linton, J. D. (2011a). Parameter settings of reinforcement learning for the game of Chung Toi. In *Proceedings of the 2011 IEEE International Conference on Systems, Man, and Cybernetics (SMC 2011), Anchorage, AK, 9–12 October* (pp. 3530–3535). doi: 10.1109/ICSMC.2011.6084216

Gatti, C. J., Embrechts, M. J., & Linton, J. D. (2013). An empirical analysis of reinforcement learning using design of experiments. In *Proceedings of the 21st European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN), Bruges, Belgium, 24–26 April* (pp. 221–226). Bruges, Belgium: ESANN.

Ghory, I. (2004). *Reinforcement learning in board games* (Technical Report CSTR-04-004, Department of Computer Science, University of Bristol). Retrieved from http://www.cs.bris.ac.uk/Publications/Papers/2000100.pdf.

Huang, D., Allen, T. T., Notz, W. I., & Zeng, N. (2006). Global optimization of stochastic black-box systems via sequential kriging meta-models. *Journal of Global Optimization*, 34(3), 441–466.

Jones, D. R., Schonlau, M., & Welch, W. J. (1998). Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13(4), 455–492.

Kalyanakrishnan, S. & Stone, P. (2009). An empirical analysis of value function-based and policy search reinforcement learning. In *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS '09), Budapest, Hungary, 10–15 May* (Vol. 2, pp. 749–756). Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Kalyanakrishnan, S. & Stone, P. (2011). Characterizing reinforcement learning methods through parameterized learning problems. *Machine Learning*, 84(1–2), 205–247.

Kliejnen, J. P. C. (2013). *Simulation-optimization via kriging and bootstrapping: A survey* (Technical Report 2013-064, Tilburg University: CentER). Retrieved from https://pure.uvt.nl/portal/files/1544115/2013-064.pdf.

LeCun, Y., Bottou, L., Orr, G., , & Müller, K. (1998). Efficient backprop. In Orr, G. & Müller, K. (Eds.), *Neural Networks: Tricks of the Trade*, volume 1524 (pp. 5–50). Berlin: Springer.

Lin, L.-J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8(3–4), 293–321.

Loeppky, J. L., Sacks, J., & Welch, W. J. (2009). Choosing the sample size of a computer experiment: A practical guide. *Technometrics*, 51(4), 366–376.

Patist, J. P. & Wiering, M. (2004). Learning to play draughts using temporal difference learning with neural networks and databases. In *Proceedings of the 13th Belgian-Dutch Conference on Machine Learning, Brussels, Belgium, 8–9 January* (pp. 87–94). doi: 10.1007/978-3-540-88190-2_13

Pichney, V., Ginsbourger, D., Roustant, O., Haftka, R. T., & Kim, N.-H. (2010). Adaptive design of experiments for accurate approximation of a target region. *Journal of Mechanical Engineering*, 132(7), 1–9.

Ranjan, P., Bingham, D., & Michailidis, G. (2008). Sequential experiment design for contour estimation from complex computer codes. *Technometrics*, 50(4), 527–541.

Robertson, B. L., Price, C. J., & Reale, M. (2013). CARTopt: A random search method for nonsmooth unconstrained optimization. *Computational Optimization and Applications*, 56(2), 291–315.

Sutton, R. S. & Barto, A. G. (1998). *Reinforcement Learning*. Cambridge, MA: MIT Press.

Tesauro, G. (1990). Neurogammon: A neural network backgammon program. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN), San Diego, CA, 17–21 June* (Vol. 3, pp. 33–39). doi: 10.1109/IJCNN.1990. 137821

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.

Thrun, S. (1995). Learning to play the game of Chess. In *Advances in Neural Information Processing Systems 7* (pp. 1069–1076). Cambridge, MA: MIT Press.

Watkins, C. J. C. H. & Dayan, P. (1992). Q-learning. *Machine Learning*, 8(3–4), 279–292.

Wiering, M. A. (1995). *TD learning of game evaluation functions with hierarchical neural architectures*. Unpublished masters thesis, Department of Computer Science, University of Amsterdam, Amsterdam, Netherlands.

Wiering, M. A. (2010). Self-play and using an expert to learn to play backgammon with temporal difference learning. *Journal of Intelligent Learning Systems & Applications*, 2(2), 57–68.

Wiering, M. A., Patist, J. P., & Mannen, H. (2007). *Learning to play board games using temporal difference methods* (Technical Report UU–CS–2005–048, Institute of Information and Computing Sciences, Utrecht University). Retrieved from http://www.ai.rug.nl/~mwiering/GROUP/ARTICLES/learning_games_TR.pdf.

Appendix A
Parameter Effects in the Game of Chung Toi

This chapter previously appeared as: Gatti et al. (2011). Parameter settings of reinforcement learning for the game of Chung Toi. In *IEEE International Conference on Systems, Man, and Cybernetics (SMC 2011), Anchorage, AK, 9–12 October* (pp. 3530–3535). doi: 10.1109/ICSMC.2011.6084216. This work used a one-factor-at-a-time (OFAT) study to assess the impact of changing individual parameters on the performance of reinforcement learning in a two player board game. This study was our first exploration into attempting to understand how parameters affect reinforcement learning, which led to a subsequent study using a classical experimental design (Appendix B), and the current work described in this dissertation.

A.1 Introduction

Reinforcement learning is a machine learning method in which an agent learns a behavior by repeatedly interacting with an environment with the goal of maximizing the total rewards received (Sutton and Barto 1998). This strategy is essentially a method of trial and error in which feedback is provided to the agent based on the utility of its actions. This feedback is used to improve the agent's knowledge of the environment so that, in subsequent interactions, the agent behaves in a more optimal manner. This paradigm pairs well with game playing in the sense that players aim to select actions which are likely to lead to the best outcome. Game playing is a unique scenario in that feedback is typically not provided during the game and following each action, rather it is provided only at the end of the game in terms of a win or loss. This sole piece of information must then be used to inform the agent of the utility of its actions.

Reinforcement learning has been applied to many board games including Tic-Tac-Toe (Wiering 1995; Ghory 2004), Othello (Binkley et al. 2007), and, most notably, backgammon (Tesauro 1992, 2002; Wiering et al. 2007). The current work adds to this body of literature by applying reinforcement learning to the game of Chung Toi (Gatti et al. 2011b), a challenging extension of Tic-Tac-Toe. This work extends our previous work which showed that a basic implementation of reinforcement learning,

C. Gatti, *Design of Experiments for Reinforcement Learning,*
Springer Theses, DOI 10.1007/978-3-319-12197-0

using a neural network, could learn a challenging environment (Gatti et al. 2011b). The reinforcement learning algorithm and the neural network have numerous parameters that are required to be set by the user. For neural networks in general, specific parameter settings and techniques can be employed that enable efficient training (LeCun et al. 1998; Embrechts et al. 2010). The utility of these techniques has not been explored in the context of the TD(λ) algorithm. The purpose of this work is to explore the effects of various algorithm settings, of the TD(λ) algorithm and of the neural network, in a basic implementation of reinforcement learning to the game of Chung Toi.

A.2 Methodology

A.2.1 Chung Toi

The game of Chung Toi is similar to that of Tic-Tac-Toe, as it is played on a 3×3 board and the goal of the game is to obtain 3 of ones' pieces in a row. Chung Toi is unique in that each player has only 3 of either white or red octagonal pieces. Additionally, the game proceeds in two phases: the first phase consists of each player taking turns and placing their pieces on the board, orienting each piece either cardinally or diagonally; the second phase consists of each player moving and/or rotating one of their pieces while attempting to align 3 of their pieces in a row. The pieces are labeled with arrows (Fig. A.1), which dictate the direction that each particular piece is allow to be moved. If the arrows are aligned cardinally with the board, the piece can be moved either horizontally or vertically to an open position, whereas if the arrows of a piece are aligned diagonally, the piece can move diagonally to an open position.

A.2.2 The Reinforcement Learning Method

Reinforcement learning enables an agent to learn how to behave in an environment by using iterative interactions with the environment to improve its ability to make decisions (Fig. A.2). With regards to the game of Chung Toi, the agent iteratively plays the game and improves its ability to place and/or move pieces around the board in order to win the game. At any instance in the game, the agent senses the state of the environment, which contains all information that is necessary to evaluate the value of the state. Such information can include the board configuration and which player is to play next. The agent selects actions by evaluating the value of all possible subsequent states, and selects the action which results in the greatest next-state value. Feedback is provided to the agent, in terms of rewards or penalties, which indicates the utility of the action, and this feedback is then used to improve the agent's estimate of the previous state value. Implementing such a paradigm therefore requires multiple entities, including models of the environment and the agent, as well as a method with which the agent can improve its knowledge about the environment.

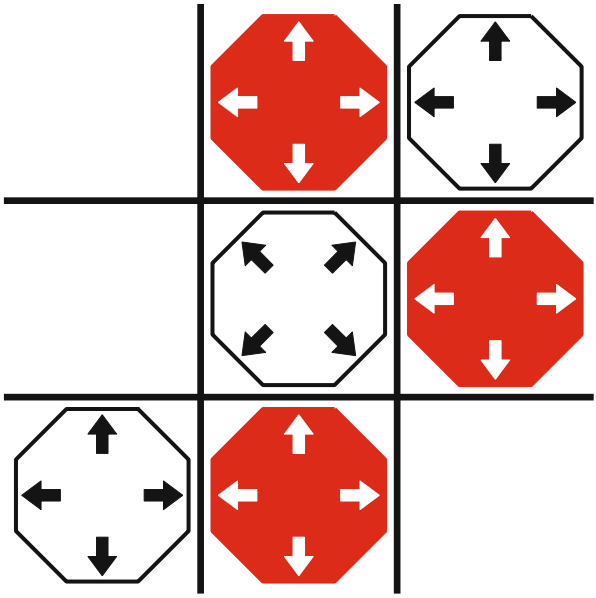


Fig. A.1 An example game board of Chung Toi from the neural network playing against a human opponent. Players first place 3 pieces on the board and then move and/or rotate pieces, trying to align 3 of ones' pieces in a row. This figure previously appeared in Gatti et al. (2011a).

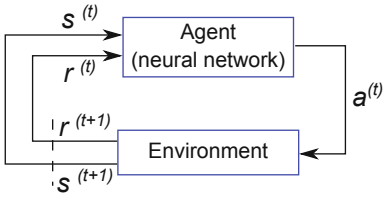


Fig. A.2 In reinforcement learning, an agent selects an action based on the current state, and rewards are provided based on the actions pursued, which are used to update state values. (Adapted from Sutton and Barto 1998. This figure previously appeared in Gatti et al. 2011a).

A.2.3 The Environment Model

The model of the environment consists of a board representation, the rules of the game, and an evaluation function. The board of Chung Toi consists of a 3×3 board, and the pieces on the board can be coded using two 9×1 vectors: one indicating the presence of a piece, and one indicating its orientation. The positions of the pieces were encoded with a 1 or -1, indicating a white piece or red piece, respectively, or a 0 indicating an open board location. The orientation of the pieces were encoded with a 1 or -1, indicating that a piece (in the corresponding position vector) was oriented cardinally or diagonally, respectively, or a 0 indicating an open board location.

Additionally, a 2×1 vector was used to indicate which player was to play next; $[1\ 0]^T$ indicated player 1 was to play next, and $[0\ 1]^T$ indicated player 2 was to play next. The state of the game was therefore fully represented by concatenating these vectors, resulting in a 20×1 state vector. Player 1 was assigned to the white pieces and always had the first turn during both training and evaluation games. The rules of the game were implemented simply as those stated above, which required determining all allowable moves for each player, as well as developing an evaluation function which could determine if a player had won the game.

A.2.4 The Agent Model

The agent interacts with the environment by selecting moves and learning from feedback resulting from winning or losing. The agent is represented as a multi-layer neural network, with inputs corresponding to the 20 components of the state vector, a single hidden layer, and a single output node. The network acts as a function approximator with the output considered to be the value of the state, with respect to player 1, that was input to the network; greater output values correspond to states from which player 1 is more likely to win the game. Moves are selected for player 1 by first identifying all possible moves, and then evaluating the value of each of these moves (i.e., evaluating all possible next states using the network). The agent follows an ϵ-greedy policy and selects the move with the greatest value with probability ϵ. This policy allows the agent to both exploit its expertise, and to explore moves that may be beneficial, but that aren't regarded as such by the current network.

The agent gains expertise by repeatedly playing games and using feedback to update the weights of the neural network and thus improve its state value estimations. In the game of Chung Toi, feedback is only provided at the end of the game, and this feedback is the only information from which the agent can learn each state value. This feedback is used with the temporal difference algorithm (TD(λ)) to adjust the weights in the neural network. This algorithm was formalized by Sutton and Barto (Sutton and Barto 1998) and can be thought of as an extension of the backpropagation algorithm (Rumelhart et al. 1986; Werbos 1974). The general form of the backpropagation algorithm for computing Δw_{jh}, the weight change for a node in layer h to a node in layer j, can be stated as:

$$\Delta w_{jh} = \alpha \times E \times \delta_j \times y_h \tag{A.1}$$

where α is the learning parameter, E is the prediction error, δ_j is the local gradient at the node in layer j, and y_h is the input to the node in layer j. The TD(λ) algorithm is based on discounting information from future states and modifies Eq. (A.1) by including a summation up to the current time step t and the temporal discount factor λ. The general form of TD(λ) for computing Δw_{jh} can be stated as:

$$\Delta w_{jh}^{(t)} = \alpha \times E \times \sum_{k=1}^{t} \left[\lambda^{t-k} \times \delta_j^{(k)} \times y_h^{(k)} \right] \tag{A.2}$$

Note that the superscripts in parentheses in Eq. (A.2) are not exponents and are instead used to indicate time steps within the game of the corresponding variables; however, the superscript on the temporal discount factor λ is an exponent. The effect of the discount parameter λ is as follows:. as $\lambda \rightarrow 0$, updates become based only on information from the subsequent state; as $\lambda \rightarrow 1$, updates amount to averaging information from all subsequent states. Equation (A.2) can be more explicitly written for computing $\Delta w_{jh}^{(t)}$, the change in the weight connecting output layer j and hidden layer h, as:

$$\Delta w_{jh}^{(t)} = \eta \Delta w_{jh}^{(t-1)} + \alpha \left(\gamma P^{(t+1)} - P^{(t)}\right) \sum_{k=1}^{t} \lambda^{t-k} f'\left(v_j^{(k)}\right) y_h^{(k)} \tag{A.3}$$

where γ is a next-state decay parameter, $P^{(t)}$ and $P^{(t+1)}$ are the state values (i.e., network output) at the current and subsequent time steps, respectively, λ is the temporal discount factor, $f'(v_j)$ is the transfer function derivative at node j evaluated at the induced local field $v_j = \sum_h w_{jh} y_h$, and $y_h = f(v_h)$ is the output of the hidden node h where $f(\cdot)$ is a transfer function. The parameter γ serves to discount the next-state value slightly, however, this parameter is sometimes not included or set to 1 as the importance of this parameter is uncertain. This equation includes a momentum term $\Delta w_{jh}^{(t-1)}$ which incorporates the weight update from the previous time step $t-1$ (scaled by the momentum learning rate η) into the weight update from the current time step t. Equation (A.3) can then be extended to compute $\Delta w_{hi}^{(t)}$, the change in the weight connecting hidden layer h and input layer i:

$$\Delta w_{hi}^{(t)} = \eta \Delta w_{hi}^{(t-1)} + \alpha \left(\gamma P^{(t+1)} - P^{(t)}\right) \sum_{k=1}^{t} \lambda^{t-k} f'\left(v_j^{(k)}\right) w_{jh} f'\left(v_h^{(k)}\right) x_i^{(k)} \tag{A.4}$$

The game and weight updates progress as follows: at time step t, a player selects the next move for time step $t+1$ and the board is updated with the new piece location/orientation; the board is evaluated for a win, loss, or no current winner; and the network weights are updated using state values P from the current (t) and next time steps ($t+1$). At the end of the game, there is no next state value $P^{(t+1)}$, and this is replaced with the reward for the current game. This method is an iterative weight updating scheme, in which weights are updated following each action.

A.2.5 Training and Performance Evaluation Methods

Training is performed by having the network play against itself: player 1 selects moves with the greatest state value (with probability ϵ), and player 2 selects moves with the smallest state value (with probability ϵ). Greedy wins and blocks were also allowed for both players during training, which allow either player to take moves that lead to a win (1st preference) or to blocking an opponent's win (2nd preference), regardless of the next-state value estimate. The performance of the network was

Table A.1 Parameter settings for the *base* experiment.

Parameter	Value
# hidden nodes	40
Weight init. method	Sampled from $\sim U[-0.2, 0.2]$
Hidden layer transfer function	Sigmoid: $f(x) = \frac{1}{1+e^{-x}}$
Output layer transfer function	Linear: $f(x) = x$
Learning rate	$\alpha = 0.001$ (constant across layers)
P(action exploitation)	$\epsilon = 0.75$
Next-state decay parameter	$\gamma = 1.0$
Temporal discount factor	$\lambda = 0.7$
Weight update method	Iterative, $\eta = 0.0$
# of training games	10,000
# of evaluation games	500

evaluated after every 1000 training games by playing 500 evaluation games. The performance was quantified by the proportions of the 500 evaluation games won. Games that had more than 100 moves were considered a draw, however in reality, there are no draws in Chung Toi. During evaluation games, player 1 always played the first move, always selected moves corresponding to the greatest state value ($\epsilon = 1$), and was not allowed to take greedy wins or blocks. The opponent selected moves at random, but was considered *'smart'* such that it was allowed to take greedy wins and blocks.

A.2.6 Experiments

The above-described implementation requires numerous parameters and settings, related to the TD(λ) algorithm, the neural network, or the game of Chung Toi, and these parameters and settings likely affect the ability of the network to learn. A base experiment (*base*) was run to serve as a reference point and used parameters that were set as recommended in other implementations of TD(λ) or for neural networks in general (Table A.1; Tesauro 1992; Wiering 1995).

Some parameters were set to remain constant for all experiments. Reward values consisted of 1 if player 1 wins, -1 if player 2 wins, and 0 for a draw. The neural network was a fully-connected 3-layer network with 20 input nodes, 40 hidden nodes (except in the experiments which varied the number of hidden nodes), and 1 output node; input and hidden layers also had a bias node with a constant input value of 1.

Individual experiments were then used to evaluate the effects of changing parameters and settings related to the TD(λ) algorithm, the neural network, and of the game of Chung Toi. Some of these settings have been found to have a significant impact on the training of neural networks (LeCun et al. 1998; Embrechts et al. 2010). For other

Table A.2 Experiment parameter settings.

Experiment	Parameter	Value
$nodes_{20}$	# hidden nodes	20
$nodes_{80}$	# hidden nodes	80
w_{LeCun}	Weight init. method	$\sim N(0, \sigma_w)$
tanh	Hidden transfer function	$f(x) = \tanh(x)$
$\tanh_{LeCun}$	Hidden transfer function	$f(x) = 1.7159 \tanh(\frac{2}{3}x)$
α_{anneal}	Learning rate α	linearly annealed
α_{Emb}	Learning rate α	init. as per (Embrechts et al. 2010)
$\alpha_{anneal,Emb}$	Learning rate α	annealed, init. as per (Embrechts et al. 2010)
$\epsilon_{0.90}$	P(action exploitation)	$\epsilon = 0.90$
ϵ_{anneal}	P(action exploitation)	Annealed from 0.75 to 1
$\gamma_{0.75}$	Next-state decay parameter	0.75
$\lambda_{0.9}$	Temporal discount factor	$\lambda = 0.9$
$\lambda_{0.4}$	Temporal discount factor	$\lambda = 0.4$
$batch_1/mom$	batch, w/momentum	epoch length = 1, $\eta = 0.5$
$batch_{10}/mom$	batch, w/momentum	epoch length = 10, $\eta = 0.5$
$games_{100k}$	# training games	100,000

parameters, such as ϵ or the number of hidden nodes, the parameter settings were selected in order to coarsely explore the network's learning ability for variations from the *base* scenario. For the experiments described below, a single parameter was changed and all others remained as those used in the *base* experiment described above. Table A.2 lists which parameters were changed for each experiment.

- **Hidden nodes**: $nodes_{20}$ and $nodes_{80}$ evaluated the effect of changing the number of nodes in the hidden layer.
- **Weight initialization**: The weight initialization method was changed in the w_{LeCun} experiment to the method described in (LeCun et al. 1998) which initializes weights by sampling from a distribution $\sim N(0, \sigma_w)$ where $\sigma_w = m^{-1/2}$ and m is the number of weights leading into node w.
- **Hidden transfer function**: The hidden transfer function was changed in two experiments to either the tanh function or a modified version of the tanh function presented in (LeCun et al. 1998), as shown in Table A.1.
- **Learning rate (α)**: The effects of the learning rate α was evaluated using three experiments: (1) linearly annealing α over the course of training (constant across layers), (2) setting the learning rates, by layer, similarly to that described in (Embrechts et al. 2010), and (3) combing the first two cases. More specifically, the learning rates in case 2 were set by: (1) setting all learning rates to 1, (2) scaling the learning rate of the input–hidden layer to $\sqrt{2}$, and (3) scaling all learning rates such that the largest was 0.001. The last step is a slight deviation from (Embrechts et al. 2010) where the largest learning rate was set such that it was equal to the learning rate used in the *base* experiment (0.001).

- **Exploitation versus exploration** (ϵ): The agent's tendency to exploit versus explore was evaluated in two experiments, one in which $\epsilon = 0.9$, and the other in which ϵ was linearly annealed over the course of training.
- **Temporal discount factor** (λ): The effect of the temporal discount factor λ was evaluated in two experiments by setting $\lambda = 0.9$ in one, and $\lambda = 0.4$ in the other.
- **Decay parameter** (γ): The decay parameter is often set to 1, however, there is some support for using a value smaller than 1 (Ghory 2004). This experiment used $\gamma = 0.75$.
- **Batch training**: The effect of using a batch training scheme was evaluated in two experiments. In these experiments, the network weights were updated after an epoch, rather than after every move. Two experiments were run, one with an epoch length of 1 ($batch_1/mom$; i.e., update weights after every game), and one with an epoch length of 10 ($batch_{10}/mom$; i.e., after every 10 games). These experiments also set the momentum parameter η to 0.5.
- **Number of training games**: The effect of extended training was evaluated using 100,000 training games; in this experiment, the performance of the network was evaluated every 10,000 games.

Following each of the experiments in which single parameters and/or settings were change, experiments were performed by changing multiple parameters. For this second round of experiments, the parameters that were changed and their values were selected based on individual experiments that resulted in good overall performance.

A.3 Results

A.3.1 Individual Experiments

The results of the experiment performed are shown in Table A.3, in decreasing order of the % of wins (from the unsmoothed performance curves). The game number at which the maximum % of wins was achieved is provided as well, however, this should not be taken as a definitive measure of the efficiency of learning. Figure A.3 provides a more complete picture of the evolution of the performance for each experiment where in each subfigure, different groups of experiments are compared to the *base* experiment. The performance curves were created by smoothing out the evaluation values by averaging each performance value with the two adjacent values (except for $games_{100k}$ which was evaluated every 10,000 games) (Wiering et al. 2007). In nearly all experiments, the % of games that ended as a draw (i.e., >100 moves) was generally between 0–2 % during the performance evaluation; only a few evaluations in the $\epsilon_{0.90}$ experiment did the % of draws exceed this level. A descriptive account of the performance of each group of experiments is provided below.

- **Base scenario**: The basic implementation (*base*) was found to perform remarkably well as its performance increased relatively quickly and maintained this level over the remainder of the training, achieving a maximum performance of 90.4 %.

Table A.3 Experiment results.

Experiment	Max % wins	Game # at max % wins
α_{Emb}	92.6	5000
$\epsilon_{0.90}$	91.8	10,000
$\gamma_{0.75}$	91.8	4000
$games_{100k}$	91.2	80,000
base	90.4	6000
$batch_1/mom$	90.2	10,000
tanh	90.2	10,000
α_{anneal}	89.0	2000
$\alpha_{anneal,Emb}$	88.8	3000
$batch_{10}/mom$	88.2	6000
$\tanh_{LeCun}$	86.4	1000
w_{LeCun}	84.2	8000
$nodes_{20}$	83.2	2000
ϵ_{anneal}	83.0	9000
$\lambda_{0.4}$	76.2	3000
$nodes_{80}$	74.0	4000
$\lambda_{0.9}$	71.4	1000

- **Hidden nodes**: The number of nodes in the hidden layer seemed to have a profound effect on the learning ability of the network (Fig. A.3a). In both the $nodes_{20}$ and $nodes_{80}$ experiments, the performance was poor overall. The small network with only 20 hidden nodes was likely unable to approximate the true state-value function, and the network with 80 hidden nodes may require additional training beyond the 10,000 games used in this comparison.
- **Weight initialization**: The experiment using the weight initialization (w_{LeCun}) method described in LeCun et al. (1998) had lower performance, compared to *base*, at every evaluation point during training, however, the performance was steadily increasing throughout (Fig. A.3b).
- **Hidden transfer function**: The experiments comparing the different hidden transfer functions performed worse throughout most of the training compared to the *base* experiment (Fig. A.3c), although performance did increase relatively quickly and was somewhat stable. Despite this, the tanh transfer function was able to achieve a maximal performance of 90.2 % at the end of training.

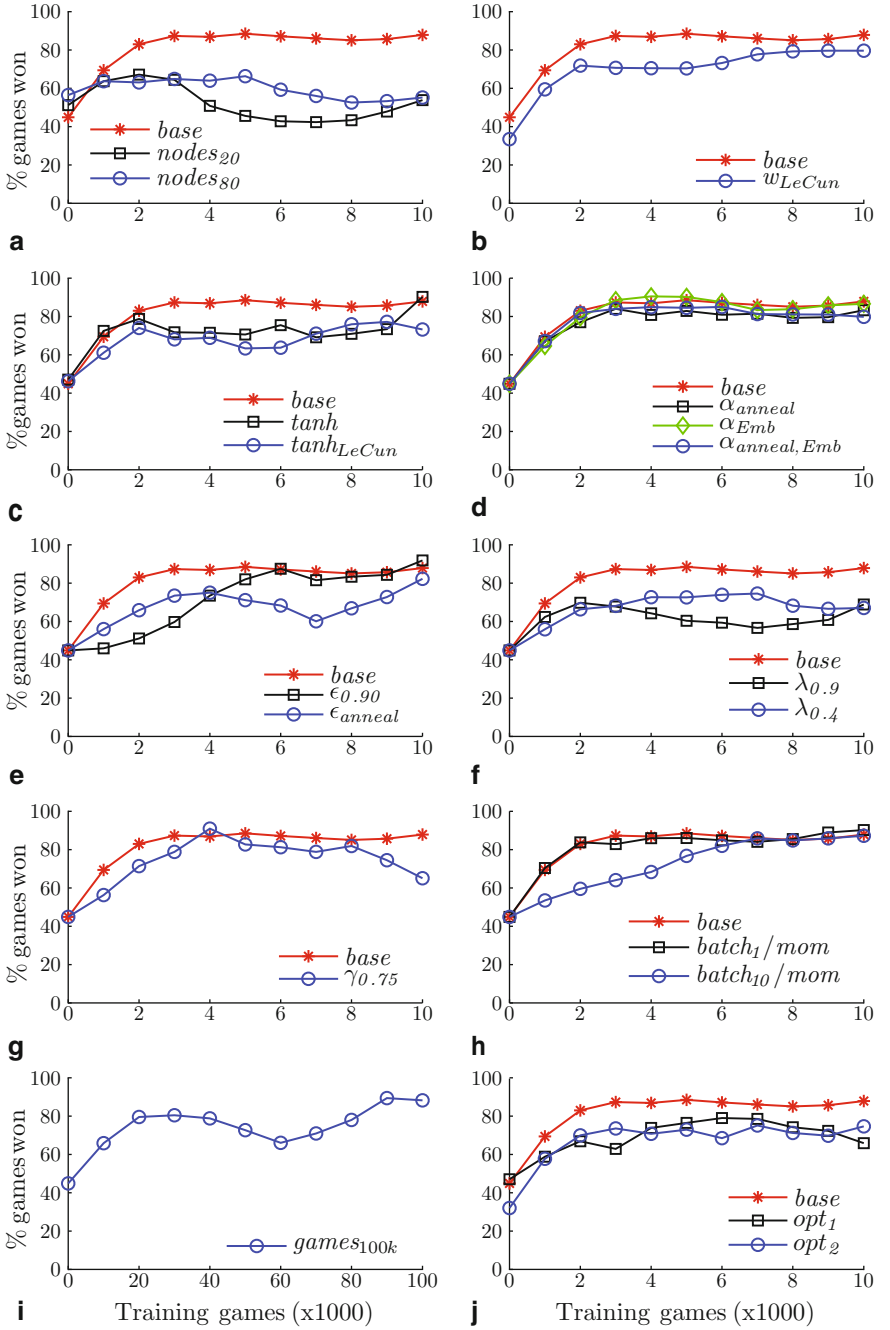


Fig. A.3 Playing performance during training for all experiments. These figures compare experiments to the *base* experiment. This figure previously appeared in Gatti et al. (2011a).

- **Learning rates** (α): The experiments evaluating the effect of using different learning rate strategies were all very similar, even to the *base* experiment (Fig. A.3d). In each experiment, the performance increased at about the same rate and remained high throughout training. When comparing maximal performance values, the weight initialization method described in Embrechts et al. (2010) (α_{Emb}) achieved the best performance overall at 92.6 %. This was about 3 % greater than the other learning rate experiments (α_{anneal} and $\alpha_{anneal,Emb}$), and about 2 % greater than the *base* experiment. It is interesting to note that, although $\alpha_{anneal,Emb}$ used the same learning rate initialization method as α_{Emb}, the performance was lower. This could be due to the fact that, in the early stages of training, the lower learning rates of $\alpha_{anneal,Emb}$ (due to annealing) were detrimental to learning.
- **Exploitation versus exploration** (ϵ): The experiments evaluating how much the agent exploits its expertise versus explores other options were seen to follow different performance curves (Fig. A.3e). The experiment with a relatively high value of ϵ ($\epsilon_{0.90}$) increased in performance slower than other experiments, but ultimately achieved a maximum performance of 91.8 %. Linearly annealing ϵ (ϵ_{anneal}) resulted in poor performance during training and in terms of maximum performance. These experiments resulted in many more draws than in other experiments during training, which is likely due the self-play training scheme. With high values of ϵ, it is more likely that each player selects actions that result in circular play.
- **Temporal discount factor** (λ): The experiments on the temporal discount factor λ show that the original setting of $\lambda = 0.7$ resulted in the best performance, which was substantially better than for values of 0.4 and 0.9, which both seemed to learn very little (Fig. A.3f).
- **Decay parameter** (γ): Changing γ to 0.75 resulted in very good maximal performance, however, the performance curve seemed to be more unstable compared to other experiments (Fig. A.3g).
- **Batch training**: Batch training with an epoch length of 1 resulted in very similar performance to that of the *base* experiment (Fig. A.3h). With an epoch length of 10, however, performance increased much slower but very steadily throughout training, achieving a maximal performance of 88.2 %.
- **Number of training games**: After training for 100,000 games, the agent achieve a maximum performance of 91.2 %, which ranked high among all experiments. However, this was neither the highest ranking performance nor did the extended training seem to add stability (Fig. A.3i).

A.3.2 Optimal Experiments

From the results presented in Table A.3, parameters were selected that resulted in the best-performing experiments, or that have been shown to perform well for neural networks in general, and were used in combination to try to improve performance further. These parameter settings are described below, where parameters that are not shown were set to the original settings as in Table A.1.

- **Optimal experiment 1** (opt_1): Hidden transfer function: $f(x) = \tanh(x)$; $\epsilon = 0.90$; learning rates = α_{Emb}; batch training ($\eta = 0.5$); $\gamma = 0.75$.
- **Optimal experiment 2** (opt_2): Weight initialization: w_{LeCun}; hidden transfer function: $f(x) = \tanh(x)$; $\epsilon = 0.95$; learning rates = α_{Emb}; batch training ($\eta = 0.5$).

The maximum performance achieved was 86.0 % and 92.0 % for the first and second experiments, respectively. These experiments had very unstable performance over training, resulting in a performance curve that was considerably worse that that of the *base* experiment (Fig. A.3j). These experiments show that, although individual parameter changes may improve learning, the effects of multiple parameter changes are not additive.

The top performing network, α_{Emb}, was then evaluated by playing five games against each of eight novice human opponents. As with the previous training and testing schemes, the network always had the first move. All actions selected by the network were based on exploiting its learned knowledge of the game (i.e., $\epsilon = 1.0$). This evaluation resulted in the network winning 12 of the 40 total games (30.0 %).

A.4 Discussion

This work explored the effects of parameter settings on the ability of a neural network to learn a challenging board game. In numerous experiments, performance was found to increase steadily throughout training. These experiments included the *base* experiment, w_{LeCun}, all α experiments, $\epsilon_{0.90}$, and both experiments with batch training and momentum. Excluding the *base* experiment, these well-performing experiments incorporate settings that seem to add stability to the training of the network. The weight initialization method by LeCun et al. (1998) is said to set initial network weights closer to their final values; this was observed after looking at the evolution of the weights over training (results not shown). The α experiments either reduced the learning rates over training or specified learning rates by layer, with a smaller learning rate on the last layer where the gradients at each node are generally largest. Lower learning rates in general seem to be advantageous, however, learning rates that are too low result in very slow training. The use of the batch weight updating method is often used to add stability by aggregating multiple weight updates.

It is interesting to note that other works that have implemented different board games have also used or recommend $\lambda \approx 0.7$, indicating that this value may result in the good overall performance across domains (Tesauro 1992; Wiering 1995; Wiering et al. 2007; Wiering 2010). Additionally, similar to the findings in (Wiering et al. 2007), playing more games did not necessarily improve training, as shown by the experiment with 100,000 games, which did not achieve the highest performance. Even during the other experiments performance was unstable and did not monotonically increase in some cases, which is likely due to the the fact that convergence is not guaranteed when using a neural network for reinforcement learning.

The evaluation of the network against human opponents was used to analyze the learned playing behavior. The network exhibited clear tactical knowledge, including the ability to take and block wins, as well as what seemed to be look-ahead plays. The human opponents would often exploit the network's knowledge gaps by repeatedly playing a winning strategy, and the network could not defend against this due to its deterministic style of play. The results of this evaluation indicate that while the basic TD(λ) algorithm was successful in training the network to play the game, the knowledge gained was below that of a novice player.

This work used only single runs of experiments for each parameter setting. TD(λ) includes a random component and thus performance may vary between runs with the same parameter settings. Running additional experiments may provide a more conclusive picture of the effects of each parameter. Other aspects of and extensions to reinforcement learning have the potential to affect the learning, but that were not explored, and some of these include other state encoding schemes, training opponents (Wiering 2010), and multi-step look-ahead (Tesauro 2002).

A.5 Conclusion

This work explored the effects of parameter settings in the TD(λ) algorithm for the game of Chung Toi. The experiments which resulted in stable and good maximal performance compared to the basic implementation, and which had only one parameter modified, included those with different learning rates α for each layer and that used a relatively high degree of action exploitation. Modifying multiple parameters resulted in worse performance that was unstable during training, indicating that the effects of parameter settings are not additive. This work adds to the body of literature concerning the application of neural network-based reinforcement learning to board games.

References

Binkley, K. J., Seehart, K., & Hagiwara, M. (2007). A study of artificial neural network architectures for Othello evaluation functions. *Information and Media Technologies*, 2(4), 1129–1139.

Embrechts, M. J., Hargis, B. J., & Linton, J. D. (2010). An augmented efficient backpropagation training strategy for deep autoassociative neural networks. In *Proceedings of the 2010 International Joint Conference on Neural Networks (IJCNN), Barcelona, Spain, 18–23 July* (pp. 1–6). doi: 10.1109/IJCNN.2010. 5596828

Gatti, C. J., Embrechts, M. J., & Linton, J. D. (2011a). Parameter settings of reinforcement learning for the game of Chung Toi. In *Proceedings of the 2011 IEEE International Conference on Systems, Man, and Cybernetics (SMC 2011), Anchorage, AK, 9–12 October* (pp. 3530–3535). doi: 10.1109/ICSMC.2011.6084216

Gatti, C. J., Linton, J. D., & Embrechts, M. J. (2011b). A brief tutorial on reinforcement learning: The game of Chung Toi. In *Proceedings of the 19th European Symposium on Articial Neural Networks, Computational Intelligence and Machine Learning (ESANN), Bruges, Belgium, 27–29 April* (pp. 129–134). Bruges, Belgium: ESANN.

Ghory, I. (2004). *Reinforcement learning in board games* (Technical Report CSTR-04-004, Department of Computer Science, University of Bristol). Retrieved from http://www.cs.bris.ac.uk/Publications/Papers/2000100.pdf.

LeCun, Y., Bottou, L., Orr, G., , & Müller, K. (1998). Efficient backprop. In Orr, G. & Müller, K. (Eds.), *Neural Networks: Tricks of the Trade*, volume 1524 (pp. 5–50). Berlin: Springer.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning internal representation by error propagation. In Rumelhart, D. E. & McClelland, J. L. (Eds.), *Parallel Distributed Processing: Exploration in the Microstructure of Cognition*. Cambridge, MA: MIT Press.

Sutton, R. S. & Barto, A. G. (1998). *Reinforcement Learning*. Cambridge, MA: MIT Press.

Tesauro, G. (1992). Practical issues in temporal difference learning. *Machine Learning*, 8(3–4), 257–277.

Tesauro, G. (2002). Programming backgammon using self-teaching neural nets. *Artificial Intelligence*, 134(1–2), 181–199.

Werbos, P. J. (1974). *Beyond regression: New tools for prediction and analysis in the behavioural sciences*. Unpublished PhD dissertation, Harvard University, Cambridge, MA.

Wiering, M. A. (1995). *TD learning of game evaluation functions with hierarchical neural architectures*. Unpublished masters thesis, Department of Computer Science, University of Amsterdam, Amsterdam, Netherlands.

Wiering, M. A. (2010). Self-play and using an expert to learn to play backgammon with temporal difference learning. *Journal of Intelligent Learning Systems & Applications*, 2(2), 57–68.

Wiering, M. A., Patist, J. P., & Mannen, H. (2007). *Learning to play board games using temporal difference methods* (Technical Report UU–CS–2005–048, Institute of Information and Computing Sciences, Utrecht University). Retrieved from http://www.ai.rug.nl/~mwiering/group/articles/learning_games_TR.pdf.

Appendix B
Design of Experiments for the Mountain Car Problem

This chapter previously appeared as: Gatti et al. (2013). An empirical analysis of reinforcement learning using design of experiments. In *Proceedings of the 21st European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN), Bruges, Belgium, 24–26 April* (pp. 221–226). Bruges, Belgium: ESANN. This work could be considered a proof-of-concept for applying a design of experiments approach to analyzing reinforcement learning by analyzing both convergence and performance, which has grown into the work described in this dissertation.

B.1 Introduction

Reinforcement learning has had a handful of successes in challenging domains, including backgammon (Tesauro 1995) and helicopter control (Ng et al. 2004). However, this learning method has not had nearly the success of other machine learning approaches, and this may be due to our limited understanding of the complex interactions between the learning algorithms, functional representations, and domain characteristics. Theoretical analysis of reinforcement learning is limited to rather simplistic scenarios (Tsitsiklis and Roy 1996), and this motivates the use of empirical methods to understand the behavior of reinforcement learning (Bhatnagar et al. 2009; Gatti et al. 2011a; Sutton and Barto 1998). However, empirical studies often use simple parameter studies and assess effects by comparative observations (Sutton and Barto 1998), which cannot easily reveal parameter interactions. A more efficient and statistically rigorous approach is to use formal design of experiments approaches (Montgomery 2008). The purpose of this study is to use a design of experiments approach to understand the effects of three parameters of the TD(λ) algorithm when using a neural network to learn the mountain car domain.

C. Gatti, *Design of Experiments for Reinforcement Learning,*
Springer Theses, DOI 10.1007/978-3-319-12197-0

B.2 Methodology

This study is based on coupling a reinforcement learning application, the mountain car domain, with an experimental design, and this section describes each component of this experimental system. We use this work as a proof-of-concept in applying design of experiments to understand the performance of reinforcement learning. Consequently, we restrict this work to a relatively simple domain and a fundamental model-free learning algorithm, however these methods can be used to understand other learning algorithms in additional domains.

B.2.1 Mountain Car Domain

The mountain car domain (Moore 1990) places a car in a valley, where the goal is to get the car to drive out of the valley. The car's engine is not powerful enough for it to drive out of the valley, and the car must instead build up momentum by successively driving up opposing sides of the valley. The state of the car is defined by its position $x \in [-1.2, 0.5]$ and its velocity $\dot{x} \in [-1.5, 1.5]$, and the goal is located at $x = 0.5$. At the beginning of each episode, the x is uniformly randomly sampled from $[-1.2, 0.5]$ and $\dot{x} = 0$. The dynamics of the car follow:

$$x_{t+1} = x_t + \Delta t \dot{x}_t$$
$$\dot{x}_{t+1} = \dot{x}_t + \Delta t \left(-9.8\, m\, \cos(3x_t) + \frac{f}{m}\, a - \mu\, \dot{x}_t \right)$$

where $\Delta t = 0.01$ is the time step, $m = 0.02$ is the car's mass, $f = 0.2$ is the engine force, and $\mu = 0.5$ is a friction coefficient. The variable a represents the action taken by the agent, where $a = -1$ for driving backwards, $a = 0$ for neutral, and $a = 1$ for driving forwards. At every time step that the car has not reached the goal, the agent receives a reward (i.e., penalty) r equal to x. When the car reaches the goal, the agent receives a reward of 1, and the episode ends.

B.2.2 Agent Representation

A three-layer neural network is used to represent the agent and to learn the value function $V(s_t, a_t)$, which represents the value of pursuing action a_t while in state s_t. The network has 2 inputs, corresponding to the state $s_t = [x_t, \dot{x}_t]^T$, 21 hidden nodes, and 3 output nodes that represent the values of the three actions. The hidden and output layers use tanh and linear transfer functions, respectively. Each time a network is created, new weights are initialized by uniform random sampling over $[-0.1, 0.1]$. The learning rates α are initialized by layer using a heuristic similar to that described in Gatti and Embrechts (2012). Input-hidden (α_{hi}) and hidden-output

(α_{oh}) learning rates are initially set to $1/\sqrt{n}$ where n is the number of nodes in the preceding layer, and α_{oh} is then divided by $\sqrt{3}$. All learning rates are divided by $\frac{1/(4 \cdot 500)}{min(\alpha)}$, which is based on the maximum number of time steps (500), resulting in $\alpha_{oh} = 0.0028$ and $\alpha_{hi} = 0.0005$.

For each experimental run, a network is trained for a maximum of 2000 episodes, where each episode consists of one attempt at trying to get the car to the goal. Network weights are updated at every time step using the temporal difference algorithm (TD(λ)) (Sutton and Barto 1998). The general form of the weight updates at time t follows $w_t = w_t + \alpha g_t$ where g_t is a λ-discounted update over all time steps up to and including t. The hidden-output layer (oh) and the input-hidden layer (hi) updates are computed, respectively, as:

$$(g_t)_{oh} = \lambda \, (g_{t-1})_{oh} + \delta_o y_h$$
$$(g_t)_{hi} = \lambda \, (g_{t-1})_{hi} + \delta_h z_i$$

where:

$$\delta_o = f'(v_o) \, e_o$$
$$\delta_h = f'(v_h) \sum_o e_o w_{oh}$$

The quantities $f'(v_o)$ and $f'(v_h)$ are the transfer function derivatives evaluated at the induced local fields $v_o = \sum_h w_{oh} y_h$ and $v_h = \sum_i w_{hi} z_i$, respectively, where $y_h = \tanh(v_h)$ and z_i is the value of input node i (all values are from time t). At the beginning of each episode, all values of g are set to zero.

The error e at time t is a 3-element vector for the 3 output nodes o:

$$e_o = \begin{cases} r_{t+1} + \gamma V(s_{t+1}, a_{t+1}) - V(s_t, a_t) & \text{if} \quad o = a_t \\ 0 & \text{if} \quad o \neq a_t \end{cases}$$

where γ is the next-state discount factor, r_{t+1} is the reward at time $t+1$, and a_t and a_{t+1} are the actions taken at times t and $t+1$.

For each experimental run, a newly initialized network is trained and tested in the mountain car domain. A network is considered to have converged if it satisfies both training and testing convergence criteria. Training convergence requires two conditions: (1) the range of the 200-episode moving average of the number of time steps for the car to reach the goal is less than 10 time steps, and (2) all episodes in the last 200 episodes require less than the maximum number of allowed time steps (500), otherwise a maximum of 2000 episodes is allowed. During training, the agent pursues an ϵ-greedy action selection procedure ($\epsilon = [0, 1]$) in which it selects the action with the greatest predicted state value $100\epsilon\%$ of the time, and it selects a random action $100(1 - \epsilon)\%$ of the time. Following training, agent performance is tested using 200 test episodes in which it uses a pure exploitative policy (i.e., $\epsilon = 1$). Testing convergence requires that the car reach the goal in all 200 test episodes, and performance was quantified by the average number of time steps required to reach the goal over the 200 test episodes.

B.2.3 Experimental Design and Analysis

The goal of this work is to *understand the effects* of λ, γ, and ϵ with respect to learning convergence and performance; this work is not aimed at optimizing (i.e., tuning) parameter settings. This study was based on a single experimental design with a two-stage analysis. The first analysis is aimed at assessing network convergence over a large parameter space. A full factorial experiment ($\mathcal{D}_1$) is run with the following continuous level settings for each parameter: λ over $[0.1, 0.9]$ incremented by 0.1, γ over $[0.95, 0.99]$ incremented by 0.01, and $\epsilon = \{0.7, 0.8, 0.9\}$. This experiment therefore consists of 135 factor-level combinations, with 10 replications at each factor-level combination. The outcome for this experiment is a binary variable indicating (empirical) convergence; recall that convergence requires that the network converge during both training and testing. A logistic regression (LR) model is then created to estimate the probability of convergence based on λ, γ, and ϵ in $\mathcal{D}_1$.

The second analysis aims to determine the effects of λ, γ, and ϵ on performance over a smaller parameter space in which the network frequently converges. The smaller parameter space $\mathcal{D}_2$ is a subset of and is extracted from $\mathcal{D}_1$ ($\mathcal{D}_2 \subset \mathcal{D}_1$), where $\mathcal{D}_2$ consists of the following level settings: $\lambda = \{0.6, 0.7, 0.8\}$, $\gamma = \{0.97, 0.98, 0.99\}$, and $\epsilon = \{0.7, 0.8, 0.9\}$. These factor levels were chosen after assessing network convergence over $\mathcal{D}_1$. This design is a 3×3 full factorial design, with 10 replications at each of the 27 factor-level combinations. Analysis of variance (ANOVA) with Type II sums of squares is used to determine if λ, γ, and ϵ (and their interactions) has significant effects on the convergence speed (i.e., episode at which training converged) and on the mean testing performance. Non-convergent runs are qualified as *undefined* responses, as opposed to *missing* data, and these runs are removed from the data for the analysis, resulting in unbalanced groups and the need for Type II sums of squares.

B.3 Results

Experimental design $\mathcal{D}_1$ resulted in 77.85 % (1051/1350) of the runs converging during training, and 48.59 % (656/1350) converging based on both training and testing convergence criteria. The proportion of times that unique factor-level combinations converged ranged from 0/10 to 10/10, confirming that some regions of the parameter space that are clearly better than others. Figure B.1 shows the empirical probabilities of convergence over $\mathcal{D}_1$. A LR model was created to estimate network convergence using linear, quadratic, and interaction terms (Table B.1), and nearly all terms have statistically significant coefficients. The LR model was used because it provides a compact functional form for predicting convergence in this application, though other function approximators, such as neural networks, could be used to model the convergence probability.

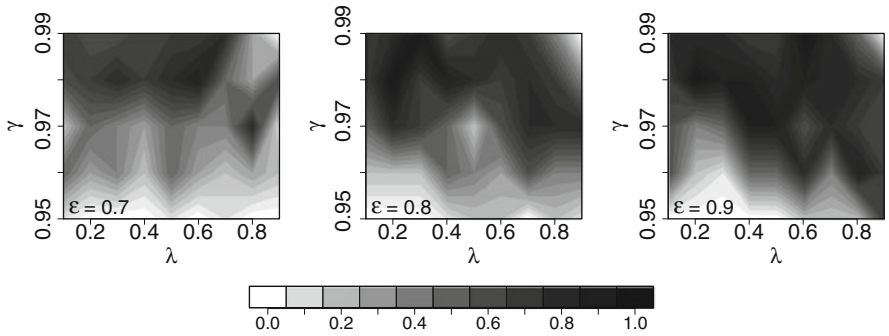


Fig. B.1 Empirical probability of convergence over $\mathcal{D}_1$ for values of ϵ. This figure previously appear in Gatti et al. (2013).

Table B.1 Summary of the logistic regression model.

Model term	Estimate	Standard error	Estimate p-value	Deviance reduction	Deviance p-value
Intercept	−3497.89	432.65	*<0.001*	–	
λ	−544.71	202.52	*0.007*	0.545	0.460
γ	7381.65	863.16	*<0.001*	228.685	*<0.001*
ϵ	−391.58	153.81	*0.011*	33.398	*<0.001*
λ^2	−4.26	1.19	*<0.001*	11.763	*<0.001*
γ^2	−3896.11	439.18	*<0.001*	79.475	*<0.001*
ϵ^2	−5.76	13.646	0.67	0.061	0.805
$\lambda : \gamma$	557.82	208.00	*0.007*	59.748	*<0.001*
$\lambda : \epsilon$	860.75	251.49	*<0.001*	5.655	*0.017*
$\gamma : \epsilon$	412.61	156.08	*0.008*	1.299	0.254
$\lambda : \gamma : \epsilon$	−876.96	258.31	*<0.001*	11.720	*<0.001*
Residual deviance	1438.1	($df = 1339$)			
Hosmer-Lemeshow GOF	$p = 0.318$				

Experimental design $\mathcal{D}_2$ resulted in 98.89 % (267/270) of the runs converging during training, and 67.78 % (183/270) converging based on both training and testing convergence criteria. Figure B.2 shows boxplots for the convergence episode and mean testing performance versus the levels of λ, γ, and ϵ. Tables B.2 and B.3 show the ANOVA models for the logarithm of the episode at convergence and the mean performance, respectively. Parameters λ and γ were found to have a significant interaction effect on convergence speed, and although the interpretation of their main effects is therefore not straightforward, it is likely they have significant main effects based on their low p-values; ϵ also has a significant main effect. For the mean performance after convergence, γ is the only parameter that had significant effects, and $\gamma = 0.99$ resulted in the best (smallest) mean performance (Fig. B.2).

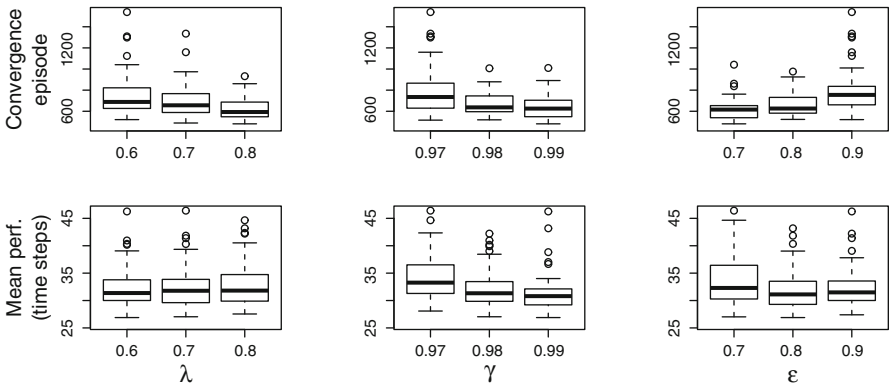


Fig. B.2 Boxplots of convergence episode and performance versus λ, γ, and ϵ. This figure previously appear in Gatti et al. (2013).

Table B.2 Analysis of variance for the logarithm of the episode at convergence; the colon between variables indicates a variable interaction (SS = sums of squares).

Model term	SS	F-ratio	p-value
λ	0.207	48.864	*<0.001*
γ	0.231	54.603	*<0.001*
ϵ	0.295	69.557	*<0.001*
$\lambda : \gamma$	0.037	8.812	*0.003*
$\lambda : \epsilon$	0.016	3.752	0.054
$\gamma : \epsilon$	0.005	1.164	0.282
Residuals	0.747		

Table B.3 Analysis of variance for the logarithm of the mean performance.

Model term	SS	F-ratio	p-value
λ	0.000	0.000	0.996
γ	0.043	20.275	*<0.001*
ϵ	0.005	2.368	0.126
Residuals	0.381		

B.4 Discussion

This study employs design of experiments and statistical analysis to aid in understanding the behavior of TD(λ) in a specific domain. Experimental design $\mathcal{D}_1$ shows that the network empirically converges to a stable solution in a fairly small region of the parameter space, however, convergence is not guaranteed. Experimental design $\mathcal{D}_2$ indicates that significant effects are of low order. While all parameters have an

effect on convergence speed, only γ has a significant effect on mean performance (though the effect is only ~ 4 time steps), and λ and ϵ do not affect the quality of the solution. In other words, while λ, γ, and ϵ and their interactions significantly affect convergence, they have little or no practical impact on mean performance. A natural extension of this work is to use additional design of experiments methods, such as response surface methodologies, to optimize parameter settings.

From a design of experiments perspective, this experiment has a unique characteristic such that some runs may not converge, and this scenario has received little or no attention in the literature. These unique outcomes motivated the use of the sequential analysis in order to separate convergence and parameters effects. Experimental design $\mathcal{D}_2$ focused on a small parameter space that converged very frequently, though it did not always converge. Furthermore, caution should be used when extrapolating the results to parameters outside of the ranges used in this study, as severe nonlinearities were observed at the parameter space edges (e.g., $\lambda = 0.01$ or 0.99, or $\gamma = 1$). This study investigated the effects of the primary variables of the TD(λ) algorithm, though the learning rate α likely also has an effect on learning, and this could be included in future work. Finally, the extensibility of the findings presented herein to different representations, learning algorithms, or domains is unknown, and more exhaustive studies are needed to form generalizable conclusions.

References

Bhatnagar, S., Sutton, R., Ghavamzadeh, M., & Lee, M. (2009). Natural actor critic algorithms. *Automatica*, 45(11), 2471–2482.

Gatti, C. J. & Embrechts, M. J. (2012). Reinforcement learning with neural networks: Tricks of the trade. In Georgieva, P., Mihayolva, L., & Jain, L. (Eds.), *Advances in Intelligent Signal Processing and Data Mining* (pp. 275–310). New York, NY: Springer-Verlag.

Gatti, C. J., Embrechts, M. J., & Linton, J. D. (2011a). Parameter settings of reinforcement learning for the game of Chung Toi. In *Proceedings of the 2011 IEEE International Conference on Systems, Man, and Cybernetics (SMC 2011), Anchorage, AK, 9–12 October* (pp. 3530–3535). doi: 10.1109/ICSMC.2011.6084216

Gatti, C. J., Embrechts, M. J., & Linton, J. D. (2013). An empirical analysis of reinforcement learning using design of experiments. In *Proceedings of the 21st European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN), Bruges, Belgium, 24–26 April* (pp. 221–226). Bruges, Belgium: ESANN.

Montgomery, D. C. (2008). *Design and Analysis of Experiments (7th edition)*. Hoboken, NJ: John Wiley & Sons, Inc.

Moore, A. W. (1990). *Efficient memory-based learning for robot control*. Unpublished PhD dissertation, University of Cambridge, Cambridge, United Kingdom.

Ng, A. Y., Coates, A., Diel, M., Ganapathi, V., Schulte, J., Tse, B., Berger, E. & Liang, E. (2004). Autonomous inverted helicopter flight via reinforcement learning. In *International Symposium on Experimental Robotics (ISER-2004), Singapore, 18–21 June* (pp. 363–372). Cambridge, MA: MIT Press.

Sutton, R. S. & Barto, A. G. (1998). *Reinforcement Learning*. Cambridge, MA: MIT Press.

Tesauro, G. (1995). Temporal difference learning and TD-Gammon. *Communications of the ACM*, 38(3), 58–68.

Tsitsiklis, J. N. & Roy, B. V. (1996). Feature-based methods for large scale dynamic programming. *Machine Learning*, 22(1–3), 59–94.

Appendix C
Supporting Tables

The tables in this appendix show supporting information for the convergent subregions and metamodels for the mountain car, truck backer-upper, and tandem truck backer-upper problems.

C. Gatti, *Design of Experiments for Reinforcement Learning,*
Springer Theses, DOI 10.1007/978-3-319-12197-0

Table C.1 Convergent subregion ranges for the mountain car problem.

Subregion #	α_{mag}	α_{ratio}	λ	γ	ϵ
6	$[5.0 \times 10^{-4}, 0.010)$	[1.832, 8.000)	[0.000, 0.849)	[0.965, 0.993)	[0.637, 0.938)
11	$[5.0 \times 10^{-4}, 0.010)$	[1.832, 8.000)	[0.000, 0.462)	[0.993, 1.000)	[0.600, 0.657)
13	$[5.0 \times 10^{-4}, 0.010)$	[1.832, 2.746)	[0.000, 0.849)	[0.965, 0.993)	[0.938, 0.970)
25	$[9.3 \times 10^{-3}, 0.010)$	[2.746, 3.209)	[0.000, 0.679)	[0.965, 0.993)	[0.948, 0.970)
32	$[2.6 \times 10^{-3}, 5.6 \times 10^{-3})$	[5.069, 7.453)	[0.748, 0.952)	[0.954, 0.965)	[0.826, 0.901)

Table C.2 Kriging model parameters for the mountain car problem.

		Trend parameters					Covariance parameters				
Subregion #	Intercept	α_{mag}	α_{mag}	λ	γ	ϵ	α_{mag}	α_{mag}	λ	γ	ϵ
6	75.523	−5.795	0.869	−4.829	−5.128	−31.952	1.196	2.000	2.000	0.574	0.182
11	83.321	−20.832	2.414	−4.018	−8.857	−9.190	0.153	2.000	2.000	2.000	0.554
13	50.208	−7.543	1.702	−9.021	−3.661	−0.274	2.000	0.522	0.534	0.147	0.091
25	39.790	−0.683	1.104	−3.417	5.044	−3.102	2.000	2.000	0.494	0.100	2.000
32	48.663	0.890	−2.288	3.495	−4.982	−7.135	2.000	2.000	0.092	2.000	0.058

Table C.3 Convergent subregion ranges for the TBU problem.

Subregion #	Hidden nodes	α_{mag}	α_{ratio}	λ	γ	ϵ
12	[26, 61)	$[6.007 \times 10^{-3}, 8.398 \times 10^{-3})$	[1.943, 8.000)	$[7.075 \times 10^{-2}, 0.298)$	[0.957, 0.988)	[0.974, 1.000)
25	[26, 61)	$[6.589 \times 10^{-3}, 8.398 \times 10^{-3})$	[1.943, 3.497)	$[7.075 \times 10^{-2}, 0.193)$	[0.961, 0.988)	[0.887, 0.965)
31	[40, 61)	$[8.398 \times 10^{-3}, 0.010)$	[2.718, 8.000)	$[0.000, 6.188 \times 10^{-2})$	[0.957, 0.985)	[0.888, 0.949)
32	[30, 61)	$[8.398 \times 10^{-3}, 0.010)$	[2.718, 8.000)	$[7.115 \times 10^{-2}, 0.166)$	[0.956, 0.985)	[0.888, 0.949)
35	[30, 59)	$[8.398 \times 10^{-3}, 0.010)$	[1.000, 7.619)	[0.000, 0.229)	[0.957, 0.979)	[0.949, 1.000)
63	[30, 61)	$[8.842 \times 10^{-3}, 9.154 \times 10^{-3})$	[4.484, 5.770)	[0.000, 0.229)	[0.979, 0.989)	[0.949, 0.997)
70	[33, 61)	$[9.451 \times 10^{-3}, 0.010)$	[1.000, 8.000)	[0.000, 0.229)	[0.979, 0.989)	[0.971, 0.994)

Table C.4 Kriging model parameters for the TBU problem.

		Trend parameters						Covariance parameters					
Subregion #	Intercept	Hidden nodes	α_{mag}	α_{mag}	λ	γ	ϵ	Hidden nodes	α_{mag}	α_{mag}	λ	γ	ϵ
12	50.958	0.932	−0.313	−0.289	0.486	11.832	−0.191	0.394	0.73	0.503	2.000	0.136	0.797
25	56.278	−0.31	−0.685	0.392	−0.431	4.611	0.435	0.000	2.000	2.000	2.000	0.098	0.403
31	54.508	0.176	−0.291	0.766	−0.626	6.891	−0.060	2.000	2.000	0.393	2.000	0.133	2.000
32	52.653	−0.117	0.671	0.653	0.054	8.255	−0.264	0.722	2.000	0.684	2.000	0.193	0.694
35	53.721	0.313	−0.033	−2.177	−0.197	8.716	−0.990	2.000	2.000	0.386	2.000	0.194	0.374
63	60.691	−0.885	−0.388	−0.048	−1.062	2.31	1.106	0.046	2.000	2.000	0.449	0.292	0.570
70	58.797	−0.293	0.347	2.334	−1.913	2.554	1.418	0.577	0.659	0.874	1.929	0.000	0.376

Table C.5 Convergent subregion ranges for the TTBU problem.

Subregion #	α_{mag}	α_{ratio}	λ	γ	ϵ
24	[0.004, 0.008)	[2.000, 2.592)	[0.493, 0.700)	[0.976, 0.990)	[0.850, 0.970)
81	[0.008, 0.010)	[2.000, 3.648)	[0.639, 0.700)	[0.971, 0.980)	[0.850, 0.970)

Christopher J. Gatti

50 2^{nd} Street, Troy, NY 12180 USA
Mobile: (734)-730-3190 *E-mail:* gatti.cj@gmail.edu

EDUCATION

Rensselaer Polytechnic Institute August 2010–August 2014
Troy, New York USA

Ph.D. Industrial and Systems Engineering

Dissertation title: *Design of Experiments for Reinforcement Learning*
Adviser: Professor Mark J. Embrechts

University of Michigan August 2000–May 2007
Ann Arbor, Michigan USA

M.S.E. Biomedical Engineering (May 2007)

B.S.E. Mechanical Engineering (May 2005)

EMPLOYMENT

Cirque du Soleil
Montreal, Quebec, Canada

Artistic acrobat July 2014–present

- Corteo: Tournik artist.

World Class Gymnastics Academy
Latham, NY, USA

Coach/instructor September 2011–July 2014

- Coached boys' and girls' Junior Olympic team gymnasts (levels 4–10).
- Instructed all types of classes, ranging from those for 3-year old children to those for adults.

Laboratory for Optimization and Computation in Orthopaedic Surgery
University of Michigan, Ann Arbor, MI, USA

Research Computer Specialist May 2007–August 2010

- Used computational modeling to analyze various orthopaedic and biomechanical problems.
- Developed and validated a finite element model of the glenoid labrum to determine the effects of humeral head migration.
- Used simulated annealing to determine optimal musculoskeletal model parameters to best match experimental measurements and reduce model development time from days to minutes.
- Developed discrete event simulation of surgery scheduling and procedures to determine the effects of changes in surgery durations.
- Used support vector machines and genetic algorithms to predict patients that will ultimately require shoulder surgery for rotator cuff repair.
- Estimated the probability of humeral head migration due to muscle force variability using Monte Carlo simulation.
- Predicted electromyographic activity of the shoulder muscles from external shoulder moments using an artificial neural network.

Great Lakes Cycling and Fitness
Ann Arbor, MI, USA

Bicycle Mechanics/Salesperson April 2008–August 2010

- Performed maintenance on customer bicycles.
  Assisted customers with bicycle serivce-related needs.

Orthopaedic Research Laboratories
University of Michigan, Ann Arbor, MI, USA

Graduate Student Research Assistant January 2006–April 2007

- Studied shoulder biomechanics using computational models and human subjects testing.

University of Michigan Athletic Department
University of Michigan, Ann Arbor, MI, USA

Gymnastics Camp Coach and Counselor Summers: 2001–2004

- Coached Junior Olympic gymnasts during summer camps.
- Supervised gymnasts in dormitories.

Graduate Research Appointments

Rensselaer Polytechnic Institute, Troy, NY, USA

Graph fragment search algorithm development May 2013–August 2014

- IARPA-sponsored Knowledge Discovery and Dissemination (KDD) program (Department of Defense)
- Developed graph fragment search algorithm for large (>1M nodes) attributed graphs
- Collaborators:
  - Dr. William Wallace (Rensselaer Polytechnic Institute)
  - Dr. Malik Magdon-Ismail (Rensselaer Polytechnic Institute)
  - Dr. Mark Goldberg (Rensselaer Polytechnic Institute)

Analysis of cortical brain layer development September 2012–April 2013

- Developed intuitive and interpretable method to understand gene activity measured by RNA-Seq in the development of the cortical layers of the brain based on a non-negative least squares model
- Collaborators:
  - Dr. Kristin Bennett (Rensselaer Polytechnic Institute)
  - Dr. Sally Temple (New York Neural Stem Cell Institute, NYNSCI)

Survival analysis of New York TB patients May 2012–August 2012

- Identified differences in trends among the survival of tuberculosis patients in seven different lineages
- Collaborators: Dr. Kristin Bennett (Rensselaer Polytechnic Institute)

Histology for Interface Stability over Time (HIST) January 2011–April 2012

- DARPA-sponsored multi-institution collaboration investigating biological stability of implantable neural electrodes
- Identified multiple phases of biological reactivity to neural implants using hierarchical clustering and biclustering
- Developed human knowledge-based approach for neural unit identification for >1TB of electrophysiology recordings
- Collaborators:

- Dr. Mark Embrechts (Rensselaer Polytechnic Institute)
- Dr. Kristin Bennett (Rensselaer Polytechnic Institute)
- Dr. Daryl Kipke (University of Michigan)
- Dr. Williaml Shain (University of Washington)
- Dr. Badri Roysam (University of Houston)

TEACHING EXPERIENCE

Rensselaer Polytechnic Institute, Troy, NY, USA

Teaching Assistant August 2010–May 2011

- ENGR 4760: Engineering Economics
  Spring 2011, Instructor: Professor Jack Reilly
- ISYE 4691: Human Performance Modeling and Support
  Fall 2010, Instructor: Professor David Mendonça

JOURNAL PUBLICATIONS

Linton J. D., Jiang Q., Gatti C. J., Chen X., and Embrechts M. J., 2013. All journals need to correct errors. *Nature*, 504: 33. (December 5, 2013)

Gatti C. J., Hallstrom B. R., and Hughes R. E., 2013. Surgeon variability in total knee arthroplasty component alignment: A Monte Carlo analysis. *Computer Methods in Biomechanics and Biomedical Engineering*, 17(15): 1738–1750.

Linton J. D., Jiang Q., Gatti C. J., and Embrechts M. J., 2013. Erratum to "Multi-attribute decision making for green electrical discharge machining" [Expert Syst. Appl. 38(7) (2011) 8370–8374]. *Expert Systems with Applications*, 40(4), 1407.

Linton J. D., Jiang Q., Gatti C. J., and Embrechts M. J., 2013. Erratum to "Taguchi-fuzzy multi output optimization (MOO) in high speed CNC turning of AISI P-20 tool steel" [Exp. Syst. Appl., 38(6) (2011) 6822–6828]. *Expert Systems with Applications*, 40(4), 1408–1409.

Linton J. D., Jiang Q., Gatti C. J., and Embrechts M. J., 2013. Erratum for "Anawa, E. M. and Olabi, A. G. Using Taguchi method to optimize welding pool of dissimilar laser-welded components" [Optics and Laser Technology 2008; 40(2): 379388]. *Optics & Laser Technology*, 48, 351–352.

Linton J. D., Jiang Q., Gatti C. J., and Embrechts M. J., 2013. Erratum to "Simultaneous optimization of flame spraying process parameters for high quality molybdenum coatings using taguchi methods" [Surf. Coat. Technol. 79(1–3) (1996) 276–288]. *Surface & Coatings Technology*, 214, 173–174.

Linton J. D., Jiang Q., Gatti C. J., and Embrechts M. J., 2013. Discussion of "Kapsiz, M., Durat, M., & Ficici, F. (2011). Friction and wear studies between cylinder liner and piston ring pair using Taguchi design method" Advances in Engineering Software, 42(8), 595-603." *Advances in Engineering Software*, 64, 71–73.

Gatti C. J., Maratt J. D., Palmer M. L., Hughes R. E., Carpenter J. E., 2010. Development and validation of a finite element model of the superior glenoid labrum. *Annals of Biomedical Engineering*, 38(12): 3766–3776.

Gatti C. J. and Hughes R. E., 2009. Optimization of muscle wrapping objects using simulated annealing. *Annals of Biomedical Engineering*, 37(7): 1342–1347.

Gatti C. J., Scibek J., Svintsitski O., Carpenter J. E., Hughes R. E., 2008. An integer programming model for optimizing shoulder rehabilitation. *Annals of Biomedical Engineering*, 36(7): 1242–1253.

Flieg N. G., Gatti C. J., Doro L. C., Langenderfer J. E., Carpenter J. E., Hughes R. E., 2008. A stochastic analysis of glenoid inclination angle and superior migration of the humeral head. *Clinical Biomechanics*, 23(5): 554–561.

Gatti C. J., Doro L. C., Langenderfer J. E., Mell A. G., Maratt J. D., Carpenter J. E., Hughes R. E., 2008. Evaluation of three methods for determining EMG-muscle force parameter estimates for the shoulder muscles. *Clinical Biomechanics*, 23(2): 166–174.

Gatti C. J., Dickerson C., Chadwick E. K., Mell A. G., Hughes R. E., 2007. Comparison of model-predicted and measured moment arms for the rotator cuff muscles. *Clinical Biomechanics*, 22(6): 639–644.

Book Chapters

Gatti C. J. and Embrechts M. J., 2012. Reinforcement Learning with Neural Networks: Tricks of the trade. In Georgieva P., Mihayolva L, and Jain L. (eds.), *Advances in Intelligent Signal Processing and Data Mining*, Springer-Verlag, Berlin-Hiedelberg, pp. 275–310.

Embrechts M. J., Gatti C. J., Linton J. D., and Roysam B., 2012. Hierarchical Clustering of Large Data Sets. In Georgieva P., Mihayolva L, and Jain L. (eds.), *Advances in Intelligent Signal Processing and Data Mining*, Springer-Verlag, Berlin-Hiedelberg, pp. 197–233.

Conference Publications

Gatti C. J. and Embrechts M. J., 2014. An application of the temporal difference algorithm to the truck backer-upper problem. *Proceedings of the 22nd European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, Bruges, Belgium, April 23–25.

Linton J., Gatti C., Jiang Q., and Embrechts M., 2012. Improving production and design processes through advances in experimental design analysis. *INFORMS Annual Meetings*, Phoenix, AZ, October 14–17.

Gatti C. J., Embrechts M. J., and Linton J. D., 2013. An empirical analysis of reinforcement learning using design of experiments. *Proceedings of the 21st European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, Bruges, Belgium, April 24–26.

Embrechts M. J., Gatti C. J., Linton J. D., Gruber T., and Sick B., 2012. Forecasting exchange rates with ensemble neural networks and ensemble K-PLS: A case study for the US Dollar per Indian Rupee. *Proceedings of the World Congress on Computational Intelligence: International Joint Conference on Neural Networks (IJCNN)*, Brisbane, Australia, June 10–15.

Embrechts M. J., Linton J. D., and Gatti C. J., 2012. Hybrid hierarchical clustering: Cluster assessment via cluster validation indices. *Proceedings of the 20th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, Bruges, Belgium, April 25–27.

Gatti C. J., Embrechts M. J., and Linton J. D., 2011. Parameter settings of reinforcement learning for the game of Chung Toi. *IEEE International Conference on Systems, Man, and Cybernetics*, pp. 3530–3535, Anchorage, Alaska, October 9–12.

Gatti C. J., Linton J. D., and Embrechts M. J., 2011. A brief tutorial on reinforcement learning: The game of Chung Toi. *Proceedings of the 19th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, Bruges, Belgium, April 27–29.

Runkle A. C., Gatti C. J., Thiele R. A. R., Palmer M. L., Hughes R. E., and Carpenter J. E., 2011. Superior glenoid labrum displacement with humeral head translation. *Transactions of the 57th Annual Meeting of the Orthopaedic Research Society*, Long Beach, CA, January 13–16.

Hughes R. E., Gatti C. J., Yacob A., Kijek T., Carpenter J .E., and Miller B. S., 2011. Predicting whether a rotator cuff tear patient will have surgery using supervised machine learning. *Transactions of the 57th Annual Meeting of the Orthopaedic Research Society*, Long Beach, CA, January 13–16.

Gatti C. J., Maratt J. D., Palmer M. L., Hughes R. E., and Carpenter J. E., 2010. Development and validation of a finite element model of the superior glenoid labrum. *American Society of Biomechanics Annual Meeting*, Providence, RI, August 18–21.

Gatti C. J. and Hughes R. E., 2009. Optimization of muscle wrapping objects using simulated annealing. *American Society of Biomechanics Annual Meeting*, Penn State University, State College, PA, August 26–29.

Gatti C. J., Scibek J., Svintsitski O., Carpenter J. E., and Hughes R. E., 2007. Integer programming models for optimizing shoulder rehabilitation. *American Society of Biomechanics Annual Meeting*, Stanford University, Stanford, CA, August 22–25.

Gatti C. J., Doro L. C., Langenderfer J. E., Mell A. G., Maratt J. D., Carpenter J. E., and Hughes R. E., 2007. Evaluation of three methods for determining EMG-muscle force parameter estimates for the shoulder muscles. *American Society of Biomechanics Annual Meeting*, Stanford University, Stanford, CA, August 22–25.

Dickerson C. R., Gatti C. J., Chadwick E. K. J., Mell A. G., and Hughes R. E., 2006. Comparison of mathematical model muscular moment arms with experimental data: A rotator cuff example. *International Shoulder Group Meeting*, Chicago, IL, October 9–10.

Submitted Journal Publications

Gatti C. J., Goldberg M., Magdon-Ismail M., Schwartz J., and Wallace W. Graph search beyond text. Submitted to: *ACM Transactions on Knowledge Discovery from Data*.

Working Publications

Brooks J. D., Gatti C. J., and Nurre S. G. Analysis of the impact and evolution of Operations Research literature using topic models.

Referee Service

- *American Journal of Sports Medicine* (2014–present)
- *IEEE Transactions on Neural Networks and Learning Systems* (2014–present)
- *Computer Methods in Biomechanics and Biomedical Engineering* (2013–present)
- *Neurocomputing* (2013–present)
- *Journal of Biomechanics* (2007–present)

Membership

- *IEEE Computational Intelligence Society* (2011–present)
- *IEEE Systems, Man, and Cybernetics Society* (2011–present)
- *IEEE* (graduate student member) (2010–present)
- *American Society of Biomechanics* (2007–2009)

Awards and Recognition

Rensselaer Polytechnic Institute

- Springer Thesis Prize (August 2014)
- Graduate Student Profile (Fall 2013)
- Founders Award of Excellence (2012)
- School of Engineering Student Profile (Spring 2012)
- Industrial and Systems Engineering Department Newsletter Feature (Summer 2011)

- Outstanding Teaching Assistant Award (Spring 2011)
- MATLAB promotional video: Robotic light painting (January 2013)

Service	Student volunteer for *IEEE International Conference on Systems, Man, and Cybernetics*, Anchorage, Alaska (2011)
	Elected captain of the University of Michigan Men's Gymnastics team for 3 years as a walk-on gymnast (2003–2005)
	Member of the University of Michigan Men's Gymnastics team (2001-2005)
	Attended 3 day leadership workshop for collegiate athletic team captains (2004)
	Served as the volunteer representative for the Men's Gymnastics team to the Student-Athlete Advisory Council and Michigan Peer Advisors Creating Trust organizations (2001-2004)
Technical Skills	Programming: R, MATLAB, C, MPI, Python
	Computer Applications: TEX (LATEX, BIBTEX), most common productivity packages (for Windows and OS X platforms)
	Finite Element Modeling: ABAQUS, HyperMesh, IA–FEMesh
	Musculoskeletal Modeling Software: AnyBody Modeling Software, Software for Interactive Musculoskeletal Modeling
	Passed the Fundamentals of Engineering (FE) Exam (2005).
Computational Modeling Experience	Neural networks, reinforcement learning, Monte Carlo simulation, simulated annealing, genetic algorithms, support vector machines, finite element analysis, topic modeling.
Statistics Experience	General statistics, regression, ANOVA models, linear mixed models, survival analysis, time series analysis, design of experiments, deterministic/stochastic kriging.

Glossary

Action selection policy The method by which the agent selects an action. This work uses an ϵ-greedy action selection policy.

Agent An entity that interacts with the environment or domain. The agent is comprised of a functional representation of its knowledge, a method of learning and updating its knowledge about the environment, and an action selection policy.

Benchmark problems Simple problems used by the reinforcement learning community to test, evaluate, and understand new algorithms and learning approaches. Some common benchmark problems include Gridworld, the mountain car problem, cart-pole balancing, and the pendulum swing-up problem.

CART (Classification And Regression Trees) A classification or regression model that results in a binary tree that partitions the input parameter space into non-overlapping hyperrectangular regions. See Breiman (2001).

Covariance kernel Kernel function which accounts for the spatial variance between points in a kriging model.

Design of experiments The formal process of designing an experimental protocol and analyzing the empirically collected data in order to discover valid and objective information about an underlying system. See Montgomery (2008).

Empirical convergence A region of the learning performance curve where learning stabilizes. This work uses a moving proportion of how often the goal is reached as a performance metric. Convergence is reached if the level, range, and slope of this curve passes threshold values.

C. Gatti, *Design of Experiments for Reinforcement Learning,*
Springer Theses, DOI 10.1007/978-3-319-12197-0

Environment/domain The system in which the agent interacts. The environment or domain is characterized by features and rules that are specific to the problem at hand.

Episode A single learning run in reinforcement learning. One attempt at learning a domain often consists of many episodes.

ϵ-greedy action selection policy An action selection policy in which the agent selects the action with the greatest next-state value $100\epsilon\%$ of the time, and selects a random action $100(1-\epsilon)\%$ of the time. This type of policy allows for the agent to both exploit its learned knowledge about the domain and to explore actions that may be potentially better than what it currently known.

FANOVA (functional analysis of variance) Based on global sensitivity analysis, this approach is useful to asses and visualize the main effects and interaction affects among variables. See Fruth et al. (2013).

Game domains Domain problems that have game-like qualities. These domains are most often thought of as, but are not limited to, those between two participants.

Generalize domains Domain problems that may be abstract and have little or no real world and physical representation.

Global sensitivity analysis A partitioning of the output variance to the input variables. This approach takes a *global* partitioning approach, which is different than the commonly used local sensitivity analysis that is based on local derivatives. See Sobol' (2001) and Saltelli et al. (2004).

Gridworld A benchmark 2-dimensional domain made up of a grid of squares in which the agent attempts to move from a starting location to a goal location. There is no set configuration of this domain, and additional features may be added to assess their impacts on learning performance.

Heuristic A trick that is used to assist a learning algorithm, possibly to assist with a deficiency in an algorithm.

Kriging An metamodeling (surrogate modeling) approach that interpolates between points based on the spatial variance between points. This approach can model both deterministic functions,

	in which case it is an exact interpolator (Matheron 1963; Cressie 1993), or stochastic functions (Ankenman et al. 2010), in which case each point has some intrinsic variance.
Learning algorithm	From a reinforcement learning perspective, an approach used by an agent to learn knowledge about a domain in order to improve its performance when interacting in the domain.
Markovian	The property/assumption that all information that is required to make a decision about which action to pursue is contained in the current state.
Reinforcement learning	A learning approach by which an agent interacts with an environment in a sequential decision making scenario. Over time, the agent learns which actions result in positive outcomes though trail and error, and thus learns how to behave in the environment to obtain a specific goal.
Representation	A functional form of the knowledge that is learned by the agent. This form must be able to both store and update information about the states of the environment and/or the allowable actions.
Reward	Feedback provided to the agent that indicates if an action or a learning run was successful. Although the term *reward* typically has a positive connotation, this quantity may be both positive or negative.
Temporal difference algorithm	A reinforcement learning algorithm that is based on the difference in the state values of two temporally adjacent states (or state-action pairs).
TBU	Truck backer-upper problem.
TTBU	Tandem truck backer-upper problem.
Value function	A function that specifies the value of being in a state in the environment; this may be extended to a function that specifies the value of being in a state in the environment while pursuing some specific action. Each environment has a true underlying value function that is unknown to the agent, and the agent attempts to learn this value function by only receiving feedback provided by the environment. This function is based on the Bellman equation (Eq. 2.3), which, in essence, states that the value of being in some state is dependent on the value of the subsequent states visited.