University of Alberta

Library Release Form

Name of Author: Lihong Li

Title of Thesis: Focus of Attention in Reinforcement Learning

Degree: Master of Science

Year this Degree Granted: 2004

Permission is hereby granted to the University of Alberta Library to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves all other publication and other rights in association with the copyright in the thesis, and except as herein before provided, neither the thesis nor any substantial portion thereof may be printed or otherwise reproduced in any material form whatever without the author's prior written permission.

Lihong Li
221 Athabasca Hall,
University of Alberta
Edmonton, Alberta
Canada, T6G 2E8

Date: ____________

"There is nothing more practical than a good theory."

James C. Maxwell

University of Alberta

Focus of Attention in Reinforcement Learning

by

Lihong Li

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Master of Science**.

Department of Computing Science

Edmonton, Alberta
Fall 2004

University of Alberta

Faculty of Graduate Studies and Research

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled **Focus of Attention in Reinforcement Learning** submitted by Lihong Li in partial fulfillment of the requirements for the degree of **Master of Science**.

Vadim Bulitko

Russell Greiner

Paul Messinger

Date:

To my parents.

Abstract

One key topic in reinforcement learning is function approximation which is critical for the success of reinforcement learning in domains with large state spaces. Unfortunately, function approximation can lead to several problems including the suboptimality of the produced policies and even divergence of learning. Thus, reinforcement learning with function approximation has remained an area of active research.

We demonstrate that in reinforcement learning, it is helpful for the agent to *focus* on more *important* states thereby producing better policies using less computing resources. The problem of *focused learning* is investigated formally, and two classes of reinforcement learning methods are considered: the classification-based approach and the value-function-based approach. For each of these two approaches, we will (i) define a formal metric of state importance, and (ii) utilize it in reinforcement learning with function approximation. The advantages of focusing attention on important states are supported both theoretically and empirically.

Acknowledgements

My deepest and foremost gratitude goes to my two supervisors, Vadim Bulitko and Russell Greiner, who have been serving the roles of mentor, teacher, encourager, collaborator, and friend. Without their insightful guidance, effective instruction, unwavering patience, generous donation of time, and everlasting encouragement, I could not have completed the thesis smoothly. I still remember the very late nights or early mornings when we were still working together to meet the paper deadlines. And I thank them for giving me the freedom to develop and pursue my own interests. Especially, I must acknowledge the dozens of things they taught me about writing in English — without their suggestions, maybe only myself could understand this thesis. I would also like to extend special thanks to Prof. Paul Messinger, from the Business School at the University of Alberta, for agreeing to be on my committee and spending time reading the thesis.

Undoubtedly, my research interests and viewpoints have been greatly affected by Rich Sutton. It is my great honor and pleasure to take his first reinforcement learning course given at the University of Alberta. I owe much to him for sharing many of his insights, for the very helpful discussions about artificial intelligence and reinforcement learning, and for the inspiration and encouragement. I wish to be on the same flight with him again so that I would have another free AI lecture.

I would like to mention the outstanding artificial intelligence/machine learning research group at the University of Alberta. The *Alberta Ingenuity Center for Machine Learning (AICML)* keeps growing rapidly and has been of great benefit to me. Dale Schuurmans is recognized to be an extremely nice professor always ready to talk, to discuss, and to help. I have enjoyed attending his lectures and regretted that he had only taught two graduate courses before I graduated. In addition, I would like to thank Michael Bowling for broadening my knowledge of game-theoretic learning as well as reinforcement learning, and Robert Holte for a discussion of my research when I was going to decide the topic.

I have enjoyed the graduate study at the department, partly due to the fellow students here. I can only name a few of them. I am lucky to work with the other members at the *Intelligent Reasoning Critiquing and Learning (IRCL)* research group, Greg Lee and Ilya Levner, who have helped and contributed to this thesis in a number of ways through our weekly meetings. Burc Derya Gunes helped with preparing some of the figures. The discussions of ML/RL with Yuxi Li and Tao Wang have been very helpful and encouraging.

In particular, I am grateful to my aunt's family. It is a great fortune to have their company during my stay in Edmonton. My aunt and uncle have helped in almost every aspect in life. My cousins, David and Dickson, are always ready to help me get used to the new life in Canada and make my life more colorful. I could not have focused entirely on my work without their unwavering dedication.

I will always wholeheartedly thank my parents and brother, who gave me everything. I am sorry for leaving home faraway for graduate study; I am sorry for talking less to them when busy with work. But they never complain and have been supporting me all the time with unwavering love and dedication. I could not have better parents and brother. Finally, I want to say a special thank you to Lilian, for sharing both happiness and unhappiness in the past two years. Her love and understanding has inspired me to work harder.

Funding for this project was provided in part by the University of Alberta, Natural Sciences and Engineering Council of Canada (NSERC), and the Alberta Ingenuity Center for Machine Learning (AICML).

Contents

1 Introduction **1**

1.1 Sequential Decision Making in Artificial Intelligence . . . 1

1.2 Reinforcement Learning: Learning in Sequential Decision Making . . . 2

1.3 Thesis Overview and Contributions . . . 3

1.4 Thesis Reading Guide . . . 5

2 Background and Motivation **6**

2.1 Sequential Decision Making and Markov Decision Processes . . . 6

2.2 Reinforcement Learning . . . 8

2.2.1 Notation . . . 9

2.2.2 Basic Algorithms . . . 11

2.2.3 Two Classes of General Reinforcement Learning Algorithms . . . 15

2.2.4 Two Types of Reinforcement Learning Problems . . . 17

2.3 Function Approximation for Reinforcement Learning . . . 20

2.3.1 Why Function Approximation? . . . 20

2.3.2 Reinforcement Learning with Function Approximation . . . 20

2.3.3 Classification-based Approximate Policy Iteration . . . 21

2.3.4 Policy Gradient and Actor-Critic Method . . . 23

2.4 Motivation . . . 25

2.4.1 Empirical Results in A Real-World System: MR ADORE . . . 25

2.4.2 A Discussion of Supervised Learning in Reinforcement Learning . . . 28

2.4.3 Research Objectives . . . 32

3 Related Work **33**

3.1 Two Problems with Function Approximation in Reinforcement Learning . . . 33

3.1.1 The Divergence Problem . . . 33

3.1.2 The Suboptimality Problem . . . 35

3.2 Limitations of Previous Work 36
3.3 Conclusions 38

4 Focus of Attention in Classification-based Reinforcement Learning 39

4.1 The Policy Switching Theorem 39
4.2 Focusing Attention: Batch Reinforcement Learning 40
4.3 Focusing Attention: Online Reinforcement Learning 43
4.4 Empirical Evaluation 46
4.4.1 Experimental Domain 47
4.4.2 Experiment I: Batch Reinforcement Learning 47
4.4.3 Experiment II: Online Reinforcement Learning 51
4.5 Proofs 56

5 An Extension to the Value-Function Methods 62

5.1 Focusing Attention: Batch Reinforcement Learning 62
5.1.1 Inconsistency between Policy Values and Value Function Approximation Accuracy 62
5.1.2 Focusing Attention on More Important States. 63
5.1.3 Penalty for Making Suboptimal Actions 65
5.2 Focusing Attention: Online Reinforcement Learning 65
5.2.1 A Small Example 68
5.2.2 Gradient of Policy Value in Changing Value Function Parameters . . . 70
5.2.3 A Combination of Policy-Gradient and Value-Function Approach . . . 71
5.2.4 An Instantiation of the PGVF Architecture 72
5.3 Empirical Evaluations 74
5.4 Further Discussion 76
5.4.1 PGVF vs. Policy Gradient 78
5.4.2 PGVF vs. VAPS 79

6 Conclusions and Future Work 80

6.1 Thesis Summary 80
6.2 Future Work 81

Bibliography 82

A Supervised Learning 89

A.1 Overview 89

A.2 PAC Learning 91
A.3 Ensemble Learning 92
A.4 Examples of Supervised Learning 93
A.4.1 Gradient Descent 93
A.4.2 Artificial Neural Networks 95

List of Figures

1.1 The agent-environment interface. 2

2.1 Policy iteration. 13

2.2 Value iteration. 14

2.3 The Monte Carlo algorithm for policy evaluation. 15

2.4 An implementation of the rollout technique. Simulate is a sub-routine that generates the next state and immediate reward using a generative model. . . 16

2.5 The SARSA algorithm. 17

2.6 The Q-learning algorithm. 18

2.7 CI-cRL: Cost-insensitive classification-based RL. Learn is a sub-routine that induces a classifier from the input training data. 22

2.8 Offline operation of MR ADORE for policy acquisition. 26

2.9 Training data used in MR ADORE. (a) An original photograph. (b) The corresponding desired labeling provided by an expert as a part of the training set. 27

2.10 Empirical results of SQUARELEV.R applied to MR ADORE. When the training and test errors decrease, the relative policy value increases. See Appendix A.1 for a formal definition of RMSE (root mean squared error). 29

2.11 Empirical results of SQUARELEV.R applied to MR ADORE. Although the training and test errors decrease, the relative policy value can decrease slightly. See Appendix A.1 for a formal definition of RMSE (root mean squared error). 30

2.12 The car-shopping problem. 31

3.1 Tsitsiklis and van Roy's example for illustrating the divergence of dynamic programming with least-squares linear function approximation. 34

3.2 An illustrative example for the problematic STD(λ) algorithm. 37

4.1 CS: Cost-Sensitive batch RL based on classification. CS-Learn is a sub-routine that induces a cost-sensitive classifier from the input training data. 42

4.2 CS-cRL: Cost-Sensitive classification-based RL. CS-Learn is a sub-routine that induces a cost-sensitive classifier from the input training data. 46

4.3 A typical distribution of immediate rewards in the two-dimensional grid-world used for our empirical evaluation. The rewards were randomly generated under a mixture distribution scheme consisting of a uniform distribution and a Gaussian distribution given in Equation 4.24. 48

4.4 A typical distribution of importance values in the two-dimensional grid-world used for our empirical evaluation of focused learning. 49

4.5 Policy value and misclassification cost in the grid-world experiment. Standard deviations are plotted every 1000 learning trials. In each trial, one training state is drawn from $\mathcal{T}$ for updating the weights in the artificial neural network. 50

4.6 Policy values in the first ten policy iterations, averaged over 50 runs. 52

4.7 Policy values of the 50 runs. Average performances correspond to the solid circles in the figures. 53

4.8 (Weighted) classification errors on the training set, averaged over 50 runs. . . 54

4.9 (Weighted) classification errors over the entire state space, averaged over 50 runs. 55

5.1 Approximating the optimal value function more precisely can lead to policies with lower quality. The values were averaged over 20 runs of experiments with variances plotted. 64

5.2 Comparison of the three batch value function learning methods. VF0 can decrease RMSE consistently but the policy value may not increase correspondingly; in contrast, VF1 and VF2 can compute a better policy although the value function is less accurate. 66

5.3 Comparison of the three batch value function learning methods. VF1 and VF2 can produce better policies although the value functions are less accurate than the one computed by VF0. 67

5.4 A simple 2-state MDP. 68

5.5 The three different performances of different values of θ for the policy prediction problem on the 2-state MDP (Figure 5.4). The bottom graph shows an enlarged fragment of the top graph. 69

5.6 An architecture combining policy gradient and pure value function learning methods, which is flexible enough to allow a tradeoff between the policy value and value function approximation accuracy. 71
5.7 An illustration of the arbitrator function defined in Equation 5.9. 73
5.8 An instance implementing the PGVF architecture. 75
5.9 A comparison between SARSA(0) and PG-SARSA(0) using Ar_1 on the 2-state MDP. 76
5.10 Different values of η controls the tradeoff between value function accuracy and the policy value in PG-SARSA(0) using Ar_3 on the 2-state MDP. 77
5.11 The converged values of θ with different η in PG-SARSA(0) and PG-SARSA(1) using Ar_3 on the 2-state MDP. 77

A.1 The mathematical model of a neuron. 96
A.2 The Sigmoid function (Equation A.9). 96
A.3 A multi-layer feed-forward neural network with one hidden layer. 97

List of Tables

1.1 A summary of the work in the thesis categorized along two dimensions. . . . 4

2.1 Optimal and two approximate policies for the car-shopping problem. 32

4.1 Statistics of the policy value improvements of CS-cRL obtained from 50 runs of experiment. 52

5.1 Features used in the grid-world domain. A state s is represented by its coordinates, (x, y). 63

Chapter 1

Introduction

This thesis is about *how a reinforcement learning agent can improve its policy by focusing attention on more important states.* As an introduction, this chapter will first give a high-level overview of the main topic, including sequential decision making, reinforcement learning, and function approximation. Then we will summarize the motivation and contributions of the thesis, followed by a thesis reading guide. Formal definitions of the terminology are found in the next chapter.

1.1 Sequential Decision Making in Artificial Intelligence

Sequential-decision making [Littman, 1996] is one of the key problems in artificial intelligence [**?**]. In this framework, the task of the intelligent agent cannot be accomplished in a single step; instead, it has to take multiple actions before the goal is achieved. Such a problem is very common in many control problems such as bicycle riding, as well as animal behavior and human activities such as game playing, inventory management, and market prediction.

For example, imagine an agent playing the game of Tic-Tac-Toe. The agent and its opponent take turns to place X's or O's on a 3×3 board. A player wins if she manages to place three of her marks in a row, either horizontally, vertically, or diagonally. This is a typical sequential decision making problem, where each agent has to take a number of moves before the game ends with a winner or a draw, and each move she takes depends on the current situation of the game board.

A common interface exists in all sequential decision-making problems: the agent-environment interface (Figure 1.1), in which the world consists of the agent and the environment. In artificial intelligence, an agent is an entity, either a software or hardware implementation, that has a goal and can perceive and take actions. The agent has full control over its knowledge

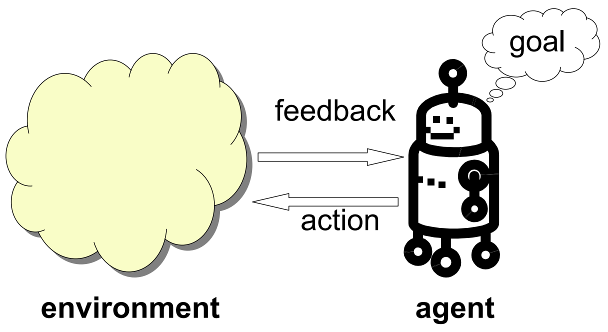


Figure 1.1: The agent-environment interface.

about the world, as well as its behavior which is determined by its policy. Basically, a policy answers the question "what shall I do now?". Everything outside the agent is considered as part of the environment which the agent may not have a full control over or observation of. The agent and the environment then *interact* as follows: in each cycle, the agent perceives the state of the world, and takes an action according to its policy; in response to each action, the environment reaches a new state and at the same time provides the agent with some feedback; then the next cycle begins. An environment is *deterministic* if the same action in the same state always leads to the same feedback and the same new state; otherwise, it is *stochastic*.

1.2 Reinforcement Learning: Learning in Sequential Decision Making

Reinforcement learning (RL) [Sutton and Barto, 1998], or neuro-dynamic programming (NDP) [Bertsekas and Tsitsiklis, 1996], is the most studied general framework for learning optimal sequential decision-making policies, through interaction with a possibly unknown and stochastic environment. Having succeeded in a number of important and difficult applications, such as the game of backgammon [Tesauro, 1992; 1995], job-shop scheduling [Zhang and Dietterich, 1995], elevator dispatching [Crites and Barto, 1996], dialogue policy learning [Singh *et al.*, 2002], power control in wireless transmitters [Berenji and Vengerov, 2003], and helicopter control [Ng *et al.*, 2004], reinforcement learning has attracted a great deal of research interest and become one of the central topics in machine learning and artificial intelligence. This section highlights several key characteristics of reinforcement learning without delving into technical details.

The first key idea of reinforcement learning is *learning from interaction*. This is in contrast to supervised learning, a large subfield of machine learning [Mitchell, 1997], which assumes the existence of an external teacher who tells the agent the correct decisions[1]. Then a supervised learning agent attempts to *emulate* the teacher. In reinforcement learning, however, a teacher is not provided. Consequently, the RL agent has to do *trial-and-error* search in the state space and *actively* collect information for learning the optimal policy, *only* by interacting with the environment.

Another key characteristic of reinforcement learning, also common in other sequential decision-making problems, is the *delayed* rewards. In real life, it is common that the long-term consequences of an action are not reflected by the immediate feedback of the environment. For example, an investor does not know exactly whether his investment in a stock will bring him profit or loss, or how much profits there will be, until the outcome has come true. Further, the outcome may also depend on a number of factors that occur after the investment is made, which makes it even more difficult to evaluate whether the investment is good. Therefore, the investor (agent) has to solve the well-known *temporal credit assignment problem* — a problem of appropriately apportioning credit and blame to the states and actions that lead to the final outcome [Sutton, 1984].

Third, reinforcement learning has to balance the exploitation and exploration of the environment. Specifically, in order to maximize the rewards, the agent will evaluate how good an action or state is, and prefers the action or state that seems best according to its current evaluation. However, it is possible that the evaluation is inaccurate, or the environment is non-stationary. Thus, it is beneficial for the agent to try suboptimal actions or states *occasionally* in the hope of potentially better policies and higher long-term rewards.

1.3 Thesis Overview and Contributions

A number of reinforcement learning algorithms exist. They either attempt to learn the optimal policy *directly*, or to learn an evaluation function from which the optimal policy can be derived. Almost all real-world problems of interest are so complex and large that some *compact* representation has to be used to *approximate* the target policy or evaluation function. These include a number of widely studied supervised learning techniques such as neural networks [Haykin, 1999], decision trees [Breiman *et al.*, 1984; Quinlan, 1993], and support vector machines [Vapnik, 1999]. Two problems exist in the presence of approximation.

First, when function approximation is used, the convergence property of evaluation function learning is guaranteed only under some very limited conditions. In fact, examples of

[1]NB: Generally speaking, instructions from the teacher may be noisy, meaning that they may not always be correct. But it is still different from the reinforcement learning problem.

	Batch RL	Online RL
Classification-based Policy Search Method (Ch. 4)	Sec. 4.2	Sec. 4.3
Value-Function Method (Ch. 5)	Sec. 5.1	Sec. 5.2

Table 1.1: A summary of the work in the thesis categorized along two dimensions.

divergence have been shown even when both the problem and the function approximation are astonishingly simple. In the last decade, a large amount of efforts have been invested in studying this problem. This is an important issue, but will not be discussed in detail in the thesis.

Second, even if the approximated evaluation function converges, it may converge to a suboptimal solution due to the *inaccuracy* of approximation or the inability to represent the optimal policies or value functions *exactly*. Consequently, the RL agent may try to approximate the target function or policy as accurately as possible, by using advanced supervised learning techniques, with the hope that a more *accurate* evaluation function results in a *better* policy. However, the ultimate goal of reinforcement learning is to compute a good policy, in the sense of high rewards. We present empirical evidence that these two metrics (function approximation performance and policy quality) can conflict in practice. In other words, supervised learning should be applied to reinforcement learning problems with caution for better outcomes.

Thesis Contributions

We show that, in sequential decision making, not all states are equally important in terms of preferring one action to another. Therefore, it is beneficial for the agent to achieve optimally when making an important decision thereby increasing higher rewards. *The thesis investigates the problem of focused learning and aims at (i) defining an effective metric for measuring state importance, and (ii) utilizing such information to learn a better policy in the sense of sequential decision making.* Specifically, two primary classes of RL approaches are investigated: the policy-search methods that attempt to optimize the policy directly, and the value-function methods that learn an evaluation function first and then derive a policy from it. For each class of methods, two settings of reinforcement learning problems are examined: the batch setting where learning occurs offline using a fixed set of interaction experiences, and the online setting (also known as the full reinforcement learning problem) where the agent learns and acts at the same time. Table 1.1 summarizes the work in the thesis along these two dimensions:

For the first class of approaches, we will consider the classification-based methods, where a policy is represented as a classifier mapping states to actions, and the objective is to induce a classifier mapping states to the optimal actions. We propose a measure of state importance that is suitable for sequential decision making, and connect the importance directly to the policy performance. Advantages of focused learning are shown both theoretically and empirically.

For the second class of approaches, we analyze how individual states contribute to changes of the global policy quality when the function approximation parameter is updated. An architecture is proposed by considering this information to avoid the policy degradation that has been observed in other direct value-function methods. Furthermore, this architecture is shown to be flexible in handling the tradeoff between two possibly conflicting metrics: the policy quality and the function approximation performance.

1.4 Thesis Reading Guide

Chapter 2 provides the necessary background and notation that will be used throughout the rest of the thesis. It first gives the notation for sequential decision making and reinforcement learning, as well as their mathematical model, Markov decision processes. Some technical details necessary for later chapters will be provided. Then the motivation for the thesis research is discussed. We show that a more accurate approximation can actually translate into worse policies. Examples include both toy problems and two large, real-life systems. Finally, we will summarize the objectives of this thesis research based on these observations.

Chapter 3 reviews the related work. Two major problems in function approximation for reinforcement learning will be introduced: the divergence problem and the sub-optimality problem. The limitations in the previous study of the second problem will also be discussed.

Chapter 4 investigates the classification-based methods, where a policy is a classifier labelling states with actions directly. We propose a definition of state importance which can be incorporated in existing algorithms to result in better policies. The advantages are supported by both theoretical and empirical studies in a 2D grid-world domain.

Chapter 5 investigates the value-function methods. An architecture called PGVF (policy-gradient-based value-function learning) learns a value function while at the same time considers the policy quality. In addition, we show that PGVF is flexible in dealing with the tradeoff between policy quality and function approximation performance.

Chapter 6 concludes the thesis and discusses several directions for future research.

Appendix A briefly discusses supervised learning with a focus on techniques related to function approximation for reinforcement learning. It will also introduce several learning algorithms used in our experiments.

Chapter 2

Background and Motivation

This chapter first provides background including the notation and necessary technical details for later chapters. Then the motivation for this thesis research is discussed with support of empirical evidence.

2.1 Sequential Decision Making and Markov Decision Processes

The thesis only considers the *discrete-time* sequential decision-making problems. More specifically, we denote the set of states of the environment by $\mathcal{S}$, and the set of actions of the agent by $\mathcal{A}$. The agent starts from an initial state $s_0 \in \mathcal{S}$; at each time step $t = 0, 1, 2, \cdots$, the state of the environment is denoted by $s_t \in \mathcal{S}$, the agent chooses action $a_t \in \mathcal{A}$ to execute, resulting in an immediate feedback in the form of a real-valued reward $r_{t+1} \in \mathbb{R}$ and the next state $s_{t+1} \in \mathcal{S}$. The task is *episodic* if the process terminates after a finite number T of steps; otherwise, it is *continual* (i.e., $T = \infty$).

Formally, the goal of the agent is to maximize the cumulative rewards, $r_1 + r_2 + r_3 + \cdots$, over time. In the case of continual tasks, the infinite sum of rewards can tend to infinity. For this reason, a discount factor $\gamma \in [0, 1)$ is introduced to place a decaying weight on each immediate reward, and the cumulative rewards becomes $r_1 + \gamma r_2 + \gamma^2 r_3 + \gamma^3 r_4 + \cdots$. If the reward signal is bounded:

$$\exists R_{\mathrm{M}} \in \mathbb{R}, \forall t \in \mathbb{N}, \ |r_{t+1}| < R_{\mathrm{M}},$$

then the discounted sum is also bounded. Note that this sum can also be used in non-discounted ($\gamma = 1$), episodic ($T < \infty$) tasks by letting r_k be zero for all $k > T$. Therefore, the discounted cumulative rewards can be unified for the two types of tasks, episodic and continual.

Definition 1 *Given a fixed discount factor γ between* 0 *and* 1*, the* return *of a sequential decision-making problem from time t is defined as:*

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}.$$

Another key concept in sequential decision making is the *policy* mapping states to actions, which the agent uses to select actions based on the current state. Two types of policies exist—deterministic policies and stochastic policies. A deterministic policy specifies the action for each state; in contrast, a stochastic policy assigns a probability distribution over the set of actions conditional on the current state. Clearly, deterministic policies are a special case of stochastic policies.

Definition 2 *A* policy *is a probability distribution over the action set conditional on states:* $\pi : \mathcal{S} \times \mathcal{A} \mapsto [0, 1]$*, where*

$$\forall s \in \mathcal{S},\ \forall a \in \mathcal{A},\ \ \pi(s, a) \geq 0 \text{ and } \sum_{b \in \mathcal{A}} \pi(s, b) = 1.$$

When a policy is deterministic, we also use $\pi(s)$ to denote the *unique* action selected by policy π in state s.

In general, the action a_t selected at time t also depends on the previous history of the agent: $s_0, a_0, r_1, s_1, a_1, r_2 \cdots, s_{t-1}, a_{t-1}, r_t, s_t$. However, if the environment's state contains *sufficient* information of the previous history, the agent can just decide a_t based on s_t without considering previous state transitions or rewards. This is called the *Markovian* assumption and the resulting model, although simplifies the problem greatly, is surprisingly useful for modelling many sequential decision-making problems. A formal definition of this model, Markov decision processes [Puterman, 1994], is given below.

Definition 3 *A* Markov decision process *(MDP) is a six-tuple* $(\mathcal{S}, \mathcal{A}, D, P, \gamma, R)$ *where:*

- $\mathcal{S}$ *is a set of states;*
- $\mathcal{A}$ *is a set of actions;*
- D *is the start-state distribution from which s_0 is drawn;*
- P *is the state transition distribution with $P_{sa}(s')$ denoting the next-state distribution after taking action a in state s:*

$$P_{sa}(s') = \Pr\{s_{t+1} = s' \mid s_t = s, a_t = a\};$$

- $\gamma \in [0, 1]$ *is the discount factor;*

- *$R : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ is a bounded reward function with $R(s,a)$ denoting the expected immediate reward collected by taking action a in state s:*

$$R(s,a) = \mathbf{E}\{r_{t+1} \mid s_t = s, a_t = a\}.$$

It is possible that a task is non-Markovian, the state of the task is not fully observable, or the state/action spaces are continuous. However, in this thesis we only consider only the most widely studied MDPs, which satisfy the Assumption 1 below. This class of MDPs, although simplest in the form, represents a broad range of practical domains.

Assumption 1 *The MDPs considered in this thesis satisfy the following assumptions:*

1. *Both $\mathcal{S}$ and $\mathcal{A}$ are finite sets: $|\mathcal{S}| = n$ and $|\mathcal{A}| = m$;*

2. *The value of each component in $(\mathcal{S}, \mathcal{A}, D, P, \gamma, R)$ does not depend on time t.*

Henceforth, we will use the term *MDP model* to refer to the reward and transition function (R and P) of an MDP. For some problem (e.g., the game of backgammon), the MDP model is completely known to the agent. For other problems (e.g., helicopter control), however, this model is unavailable and the agent has to compute the optimal policy through interaction with the environment (the MDP). If an MDP terminates in a finite number of steps, it is called *finite-horizon* MDP; if it continues forever, it is called *infinite-horizon* MDP. Clearly, the two types corresponds to the episodic and continual tasks, respectively. Below we state an important assumption that is usually made in the the research of reinforcement learning.

Assumption 2 *Given a fixed policy π, assume the* stationary state distribution *and the* stationary state-action distribution *exist, which are denoted by $\mu_\infty^\pi(s)$ and $\mu_\infty^\pi(s,a)$, respectively:*

$$\begin{aligned} \mu_\infty^\pi(s) &= \lim_{t\to\infty} \Pr\{s_t = s \mid \pi\}, \\ \mu_\infty^\pi(s,a) &= \lim_{t\to\infty} \Pr\{s_t = s, a_t = a \mid \pi\}. \end{aligned}$$

2.2 Reinforcement Learning

From this section, we start the discussion of reinforcement learning which addresses the learning problem in solving MDPs [Bertsekas and Tsitsiklis, 1996; Littman, 1996; Singh, 1994; Sutton and Barto, 1998]. The learning problem is to compute a policy π to maximize the return R_t, either based on a model of the MDP or on interactions with the environment.

In this section, we do not consider the problem of function approximation. Instead, we assume that the state space is tractable so that all functions and policies can be stored in a lookup table.

2.2.1 Notation

In making decisions, the RL agent needs to *evaluate* the utility/merit of each action or state, and select either the optimal one to maximize the long-term return, or occasionally a suboptimal alternative to explore the state space. The evaluation is done by maintaining a value function:

Definition 4 *The* state-value function *for policy π, denoted by $V^{\pi}(s)$, is the expected return by starting from state s and following π thereafter:*

$$V^{\pi}(s) = \mathbf{E}\{R_t \mid s_t = s, \pi\}.$$

Definition 5 *The* action-value function *for policy π, denoted by $Q^{\pi}(s, a)$, is the expected return by taking action a in state s and following π thereafter*[1]*:*

$$Q^{\pi}(s, a) = \mathbf{E}\{R_t \mid s_t = s, a_t = a, \pi\}.$$

Clearly, $V^{\pi}(s) > V^{\pi}(s')$ indicates that by following policy π, a larger return is expected by acting from state s than from state s'. Likewise, $Q^{\pi}(s, a) > Q^{\pi}(s, a')$ indicates that taking action a in state s has a higher expectation of return than a' when following π. A similar comparison can be done between two policies. For example, if

$$\forall s \in \mathcal{S},\ V^{\pi}(s) \geq V^{\pi'}(s),$$

then π is said to be better than π'. Given a policy, the problem of calculating $V^{\pi}(s)$ or $Q^{\pi}(s, a)$ is called *policy evaluation*; the problem of improving π to a better policy is called *policy improvement*.

The *policy improvement theorem* [Bellman, 1957] guarantees that in any MDPs satisfying Assumption 1, there always exists an optimal policy π^* whose value functions are no lower than the value functions of any other policy, i.e., $\exists \pi^*$, $\forall \pi$:

$$\begin{aligned} \forall s \in \mathcal{S}, \qquad & V^{\pi^*}(s) \geq V^{\pi}(s) \\ \forall s \in \mathcal{S}, \forall a \in \mathcal{S}, \qquad & Q^{\pi^*}(s, a) \geq Q^{\pi}(s, a). \end{aligned}$$

Clearly, $V^{\pi^*}(s) = \max_{\pi} V^{\pi}(s)$ and $Q^{\pi^*}(s, a) = \max_{\pi} Q^{\pi}(s, a)$. Note that there may be more than one optimal policies, but they share the same optimal value functions.

[1]Sometimes $Q^{\pi}(s, a)$ is also called the *value of the state-action pair* (s, a).

Definition 6 *The value functions of an optimal policy are called the* optimal value functions*:*

$$
\begin{aligned}
V^*(s) &= V^{\pi^*}(s), \\
Q^*(s,a) &= Q^{\pi^*}(s,a).
\end{aligned}
$$

Given the definitions of (optimal) value functions, a set of *recursive* relations among these value functions exist, which plays a fundamental role in almost all popular RL algorithms.

- The *Bellman equations* [Bellman, 1957] establish a relation between the values of state s and its successor states s', given any policy π:

$$
\begin{aligned}
V^\pi(s) &= \sum_a \pi(s,a)\left(R(s,a) + \gamma \sum_{s'} P_{sa}(s')V^\pi(s')\right), && (2.1) \\
Q^\pi(s,a) &= R(s,a) + \gamma \sum_{s'} \left(P_{sa}(s') \sum_{a'} \pi(s',a')Q^\pi(s',a')\right); && (2.2)
\end{aligned}
$$

- The *Bellman optimality equations* [Bellman, 1957] expresses a relation between the optimal values of state s and its successor states s':

$$
\begin{aligned}
V^*(s) &= \max_a \left(R(s,a) + \gamma \sum_{s'} P_{sa}(s')V^*(s')\right), && (2.3) \\
Q^*(s,a) &= R(s,a) + \gamma \sum_{s'} \left(P_{sa}(s') \max_{a'} Q^*(s',a')\right); && (2.4)
\end{aligned}
$$

- The following four equations show how the (optimal) state-value and action-value functions are related [Sutton and Barto, 1998]:

$$
\begin{aligned}
V^\pi(s) &= \sum_a \pi(s,a)Q^\pi(s,a), \\
Q^\pi(s,a) &= R(s,a) + \gamma \sum_{s'} P_{sa}(s')V^\pi(s'), \\
V^*(s) &= \max_a Q^*(s,a), \\
Q^*(s,a) &= R(s,a) + \gamma \sum_{s'} P_{sa}(s')V^*(s').
\end{aligned}
$$

Definition 7 *Let $V(s)$ or $Q(s,a)$ be the value function estimation of the agent, then it can derive a deterministic* greedy policy *with respect to the value function by one-step lookahead into the future:*[2]

$$
\begin{aligned}
\pi_V(s) &= \arg\max_a \left(R(s,a) + \gamma \sum_{s'} P_{sa}(s')V(s')\right), && (2.5) \\
\pi_Q(s) &= \arg\max_a Q(s,a). && (2.6)
\end{aligned}
$$

[2]If several actions appear to lead to the same expected return, then they are treated as one action, and the $\arg\max_a$ operator just returns one of them randomly.

The action selected by the greedy policy is called a greedy action.

Therefore, if an RL agent has estimated the optimal value function, it can derive a greedy policy by Equations 2.5 and 2.6. The greedy policy is close to the optimal policy π^* if the estimated optimal value function is accurate enough. However, deterministic policies do not encourage exploration which is important to an RL agent. We introduce a family of commonly used stochastic policies called Gibbs softmax policies.

Definition 8 *The* Gibbs softmax *policy with respect to the value function $Q(s,a)$ is defined as:*

$$\pi(s,a) = \frac{\exp(Q(s,a)/\tau)}{\exp(\sum_b Q(s,b)/\tau)}, \tag{2.7}$$

where τ is the temperature *parameter controlling the exploitation/exploration tradeoff.*

The Gibbs softmax policies are useful in that the exploitation/exploration tradeoff is conveniently controlled by the temperature parameter: when $\tau \to \infty$, $\pi(s,a)$ tends to be a random policy; when $\tau \to 0$, $\pi(s,a)$ tends to be a pure greedy policy.

Throughout the thesis, we will use the policy value or policy loss, defined below, as the performance metric for evaluating a policy. It is clear that maximizing the policy value is equivalent to minimizing the policy loss. Therefore, these two metrics are equivalent.

Definition 9 *The* policy value *of a policy π with respect to a state distribution μ is defined as:*

$$\mathcal{V}_\mu(\pi) = \sum_s \mu(s) V^\pi(s). \tag{2.8}$$

Similarly, the policy loss *of π with respect to μ is defined as:*

$$\mathcal{L}_\mu(\pi) = \mathcal{V}(\pi^*) - \mathcal{V}(\pi) = \sum_s \mu(s)\big(V^*(s) - V^\pi(s)\big). \tag{2.9}$$

In the rest of the thesis, we will simplify the notation by omitting the μ in $\mathcal{V}_\mu(\pi)$ and $\mathcal{L}_\mu(\pi)$ when μ is the probability of visiting s or (s,a) by following policy π.

2.2.2 Basic Algorithms

This section presents some of the more important basic algorithms for solving reinforcement learning problems. Dynamic programming is useful when a complete and accurate model of the MDP is given; Monte Carlo does not require the MDP model, and is conceptually simple and relatively easy to analyze, but lacks efficiency; temporal difference learning does not require the model, is efficient and incremental, but much more difficult to analyze. A comprehensive introduction and analysis can be found in [Sutton and Barto, 1998].

Dynamic Programming

With the assumption that the MDP model is known, dynamic programming (DP) [Bellman, 1957] can be used to solve the optimal value function by making use of the Bellman (optimality) equations. Algorithms with a time complexity polynomial in the number of states exist, and their convergence to the optimal solution is guaranteed.

Policy iteration (PI) [Howard, 1960] finds the optimal value function by iteratively performing two sub-tasks: policy evaluation and policy improvement (Figure 2.1). An agent starts with a random policy. In each iteration, the agent first evaluates the value function of the current policy π, such as $Q^\pi(s, a)$. Then it computes the *greedy policy* π' from π:

$$\forall s \in \mathcal{S},\ \pi'(s) = \arg\max_a Q^\pi(s, a). \tag{2.10}$$

A theorem called the *policy improvement theorem* [Sutton and Barto, 1998] asserts that $\mathcal{V}(\pi') > \mathcal{V}(\pi)$, unless $\pi' = \pi = \pi^*$ in which case $\mathcal{V}(\pi') = \mathcal{V}(\pi) = \mathcal{V}(\pi^*)$. Policy iteration always terminates in a finite number of iterations. In fact, empirical studies found that it often converges to the optimal policy very quickly.

Value iteration (VI) [Puterman and Shin, 1978] proceeds by merging the policy evaluation and improvement steps, and thus can be viewed as *generalized policy iteration*. In particular, it updates the value function using the right-hand-side of Equations 2.3 or 2.4, which forces the value function to converge to the optimal value function (Figure 2.2).

Monte Carlo

Monte Carlo (MC) is a conceptually simple method that estimates the value of a state or a state-action pair by averaging the *actual* returns starting from it. Besides the simplicity, another advantage of MC is that it does not require the MDP model. Figure 2.3 shows one MC solution to the policy evaluation problem. Such a routine can be conveniently embedded in the framework of policy iteration in Figure 2.1.

Note that the algorithm in Figure 2.3 is for episodic tasks, namely the finite-horizon MDPs; for infinite-horizon MDPs (continual tasks), however, there is only one "episode" without termination, which renders the algorithm inapplicable at once. For such a case, however, γ has to be strictly less than one. Therefore, distant rewards do not affect the return much and thus can be ignored. This observation forms a basis for a technique called *rollout* (Figure 2.4), which is easy to parallelized and has been applied successfully to the game of backgammon [Tesauro and Galperin, 1997]. By using a *reset*[3], rollout estimates $Q^\pi(s, a)$ by *repeatedly* executing action a in state s and then following π thereafter until a

[3]A reset is a "button" that resets the agent to any desired state.

```
**Policy-Iteration**
INPUT:
    $\epsilon_\Delta$:       threshold
    $\gamma$                 discount factor
    $R(s,a)$:                reward function
    $P_{sa}(s')$:            state transition probabilities
OUTPUT: $V \approx V^*$
initialization: $\pi \leftarrow$ random policy
repeat
    // Policy Evaluation: to compute $V \approx V^\pi$
    repeat
        $\Delta \leftarrow 0$
        for each $s \in \mathcal{S}$
            $v_{\text{old}} \leftarrow V(s)$
            $V(s) \leftarrow \sum_a \Big(\pi(s,a)\big(R(s,a) + \gamma \sum_{s'} P_{sa}(s')V(s')\big)\Big)$
            $\Delta \leftarrow \max\{\Delta, |v_{\text{old}} - V(s)|\}$
    until $\Delta < \epsilon_\Delta$
    // Policy Improvement: to compute $\pi'$
    changed $\leftarrow$ false
    for each $s \in \mathcal{S}$
        $\pi'(s) \leftarrow \arg\max_a \{R(s,a) + \gamma \sum_{s'} P_{sa}(s')V(s)\}$
    If $\pi \equiv \pi'$, then changed $\leftarrow$ true
    $\pi \leftarrow \pi'$
until changed = false
return $V(s)$
```

Figure 2.1: Policy iteration.

given horizon. The returns of all runs are then averaged and the mean value becomes the action value estimate, $\widehat{Q}^\pi(s,a)$. An implementation of rollouts is illustrated in Figure 2.4. Simulate is a procedure that uses the generative model M to determine the next state and the immediate reward. Indeed, if $\gamma < 1$, rollouts can approximate the true return to any desired precision: it is shown [Kearns *et al.*, 2000] that returns after the first H_ϵ steps contribute at most $\epsilon/2$ to the total return where

$$H_\epsilon = \log_\gamma \frac{\epsilon(1-\gamma)}{2R_{\text{M}}}.$$

Temporal Difference Learning

Temporal difference (TD) learning [Sutton, 1988] is the most popular RL algorithm [Crites and Barto, 1996; Singh and Bertsekas, 1997; Tesauro, 1995; Zhang and Dietterich, 1995]. It combines the strengths of MC and DP: it does not require the MDP model (similar to MC),

```
Value-Iteration

INPUT:
    $\epsilon_\Delta$:      threshold
    $\gamma$:       discount factor
    $R(s,a)$:   reward function
    $P_{sa}(s')$:  state transition probabilities

OUTPUT: $V \approx V^*$

initialization: $V(s) \leftarrow$ random function
repeat
    $\Delta \leftarrow 0$
    for each $s \in \mathcal{S}$
        $v_{\text{old}} \leftarrow V(s)$
        $V(s) \leftarrow \max_a\{R(s,a) + \gamma \sum_{s'} P_{sa}(s')V(s')\}$
        $\Delta \leftarrow \max\{\Delta, |v_{\text{old}} - V(s)|\}$
until $\Delta < \epsilon_\Delta$
return $V(s)$.
```

Figure 2.2: Value iteration.

and can efficiently utilize the structure of the MDP by updating value functions using the Bellman (optimality) equations (similar to DP). The basic idea underlying TD learning is that learning can be achieved from the information called *temporal difference errors*.

Suppose, for example, that the agent is following a policy π and has an estimate of the state-value function. At time step t, the estimate is $V_t(s)$, and by taking an action a_t according to π a transition is experienced: $s_t \to s_{t+1}$ with an immediate reward of r_{t+1}. Recall that on average, $V^\pi(s_t)$, $V^\pi(s_{t+1})$, and r_{t+1} should satisfy Equation 2.1. If the estimate $V_t(s)$ is not accurate or the MDP is stochastic, then the two sides of the Bellman equation may not equal and the difference is:

$$\delta_t^{\text{TD}} = r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t). \tag{2.11}$$

According to Equation 2.1, the term $r_{t+1} + \gamma V_t(s_{t+1})$ above can be seen as an "updated" estimate of $V(s_t)$ at time $t+1$, while $V_t(s_t)$ is the original estimate at time t. Therefore, the difference δ_t^{TD} between these two estimates is called the temporal difference. And the value function can be updated using this information:

$$V_{t+1}(s_t) \leftarrow V_t(s_t) + \alpha\delta_t^{\text{TD}}$$

where α is the step-size parameter. The update process above is also called a *backup*.

More generally, a broader family of algorithms called TD(λ) [Sutton, 1988] computes the temporal differences in a more complicated way controlled by a parameter $\lambda \in [0,1]$. In

```
**MonteCarlo-PolicyEvaluation**
INPUT: the policy $\pi$ to be evaluated
OUTPUT: $V(s) \approx V^{\pi}(s)$
initialize: $V \leftarrow$ random function
$\forall s, ReturnList(s) \leftarrow 0$
repeat until $V(s)$ converges
    for each episode generated by $\pi$
        $R \leftarrow$ return following the first occurrence of $s$ in this episode
        Append $R$ to $ReturnList(s)$
        $V(s) \leftarrow Average(ReturnList(s))$
end repeat
return $V(s)$
```

Figure 2.3: The Monte Carlo algorithm for policy evaluation.

particular, Equation 2.11 corresponds to TD(0); and the aforementioned MC can be seen as a special case of TD(1). A number of other RL algorithms are also based on the idea of learning from TD errors, such as actor-critic [Barto *et al.*, 1983; Konda and Tsitsiklis, 2000].

The most well-known TD algorithms for solving MDPs are SARSA [Sutton, 1996] and Q-learning [Watkins, 1989], shown in Figures 2.5 and 2.6, respectively, for the case of $\lambda = 0$. Both algorithms are based on the Bellman optimality equation 2.4 and attempt to compute $Q^*(s, a)$. Note that if $Q(s, a) = Q^*(s, a)$, then the expected TD error is 0. Therefore, these two algorithms can be seen as performing gradient descent on the TD errors. Eventually, when the TD error reaches 0, $Q(s, a)$ converges to $Q^*(s, a)$. In fact, for finite MDPs, if $Q(s, a)$ is stored in a lookup table with one entry for one (s, a) pair and each (s, a) is experienced infinitely often, then both SARSA(0) and Q-learning converge to the optimal action-value function $Q^*(s, a)$ [Singh *et al.*, 2000; Watkins, 1989].

2.2.3 Two Classes of General Reinforcement Learning Algorithms

A number of modern RL algorithms have been developed since the early 1980's. These algorithms can be categorized along different dimensions (e.g., see Chapter 10 in [Sutton and Barto, 1998]), such as (i) on-policy vs. off-policy, (ii) model-based vs. model-free, (iii) sample backups vs. full backups, (iv) shallow backups vs. deep backups, etc. In this thesis, we will focus on two types of RL algorithms: the *value-function* methods and the *policy-search* methods.

Value-function methods [Baird, 1995; Bertsekas and Tsitsiklis, 1996; Sutton, 1988; 1996;

```
Rollout
INPUT:
    $M$:                   generative model
    $\langle s_0, a_0 \rangle$:   the state-action pair to evaluate
    $\pi$:                 the policy to be evaluated
    $\gamma$:              discount factor
    $K$:                   number of trajectories
    $H$:                   maximum length of trajectories
OUTPUT: an estimate of $Q^\pi(s_0, a_0)$
for $k = 1$ to $K$
    $s \leftarrow s_0$, $a \leftarrow a_0$
    $(s', r) \leftarrow \texttt{Simulate}(M, s, a)$
    $\widehat{Q}^\pi_k \leftarrow r$
    $s \leftarrow s'$
    for $h = 1$ to $H - 1$
        $a \leftarrow \pi(s)$
        $(s', r) \leftarrow \texttt{Simulate}(M, s, a)$
        $\widehat{Q}^\pi_k \leftarrow \widehat{Q}^\pi_k + \gamma^h r$
        $s \leftarrow s'$
$\widehat{Q}^\pi \leftarrow \frac{1}{K} \sum_{k=1}^{K} \widehat{Q}^\pi_k$
return $\widehat{Q}^\pi$
```

Figure 2.4: An implementation of the rollout technique. Simulate is a sub-routine that generates the next state and immediate reward using a generative model.

Sutton and Barto, 1998; Watkins, 1989] are the most popular. Recall that the goal of reinforcement learning is to acquire a policy. Value function methods, however, do not compute a policy directly. Instead, they attempt to learn a value function from which a policy is derived. The algorithms described in the previous subsection belong to this family.

Policy-search methods, as the name suggests, seek the desired policy in a fixed policy space Π [Bagnell *et al.*, 2004; Kakade, 2002; Kearns *et al.*, 2000; Ng and Jordan, 2000; Ng *et al.*, 1999; Williams, 1992]. They may or may not make use of the value functions. For example, genetic algorithms [Holland, 1975] and simulated annealing [Kirkpatrick *et al.*, 1983] can be used to perform stochastic search in the policy space and hopefully, the set of policies they find converge to a good solution.

Both value-function and policy-search methods have pros and cons of their own, and have remained a topic of active research. The dominant approach in the last decade has been the former family of methods. By using value functions (often called critics), these methods are usually more efficient because the structure of the agent's interaction history can be better utilized to compute a good policy. On the other hand, pure policy-search

```
Sarsa
INPUT:
    $\gamma$:    discount factor
OUTPUT: $Q(s,a) \approx Q^*(s,a)$
for each $(s,a) \in \mathcal{S} \times \mathcal{A}$
    $Q(s,a) \leftarrow$ random value
repeat until $Q(s,a)$ converges
    initialize $s \sim D$
    $a \leftarrow \pi_Q(s)$ (greedy with exploration)
    repeat
        Take action $a$, observe $r$ and $s'$
        $a' \leftarrow \pi_Q(s')$ (greedy with exploration)
        $Q(s,a) \leftarrow Q(s,a) + \alpha\big(r + \gamma Q(s',a') - Q(s,a)\big)$
        $s \leftarrow s'$
        $a \leftarrow a'$
    until $s$ is the terminal state
end repeat
return $Q(s,a)$
```

Figure 2.5: The SARSA algorithm.

methods ignore such useful information. But as we will discuss in the later sections, policy-search methods are easier to analyze and their convergence guarantees are stronger when function approximation is used. Also, policy-search has a lower computational cost when the set $\mathcal{A}$ of actions is large or even continuous. For these reasons, there has been a growing interest in this class of methods. In this thesis, two types of policy-search methods will be considered:

- the classification-based methods [Fern *et al.*, 2004; Lagoudakis and Parr, 2003b; Langford and Zadrozny, 2003; Yoon *et al.*, 2002] in Chapter 4;
- policy gradient (including actor-critic) methods [Barto *et al.*, 1983; Baxter and Bartlett, 1999; Konda and Tsitsiklis, 2000; Sutton *et al.*, 2000] in Chapter 5.

2.2.4 Two Types of Reinforcement Learning Problems

So far, we have considered the *full* reinforcement learning problem. That is, the agent acts online and learns a policy or a value function at the same time. There is another class of RL problems called *batch reinforcement learning*. In batch reinforcement learning, the agent is presented with a *fixed* set of experiences from which a policy has to be computed. That is, the learning process occurs *offline*. For this reason, batch reinforcement learning is sometimes called *offline reinforcement learning*.

```
Q-Learning
INPUT:
    $\gamma$:  discount factor
OUTPUT: $Q(s,a) \approx Q^*(s,a)$
for each $(s,a) \in \mathcal{S} \times \mathcal{A}$
  initialize: $Q(s,a) \leftarrow$ random value
repeat until $Q(s,a)$ converges
  initialize $s \sim D$
  repeat
    $a \leftarrow \pi_Q(s)$
    Take action $a$, observe $r$ and $s'$
    $Q(s,a) \leftarrow Q(s,a) + \alpha\big(r + \gamma \max_{a'} Q(s',a') - Q(s,a)\big)$
    $s \leftarrow s'$.
  until $s$ is the terminal state
end repeat
return $Q(s,a)$
```

Figure 2.6: The Q-learning algorithm.

The batch/offline reinforcement learning framework is important whenever online learning is not feasible (e.g., when the reward data are limited), and therefore a fixed set of experiences has to be acquired and used for offline policy learning [Draper *et al.*, 2000; Levner and Bulitko, 2004]. A related technique called *experience replay* has been employed in robotics and was shown to speed up TD-learning and reduce possible damage to the learning robot [Lin, 1992]. An experience-replay RL agent simply remembers its past online experiences and then repeatedly updates its value function or policy using these offline experiences. A recently proposed method called LSPI [Lagoudakis and Parr, 2003a] was shown to make efficient use of data by applying the least-square technique offline on reusable sampled experience. Another similar idea has been adopted in the Dyna-Q architecture [Sutton, 1990], in which the agent improves its policy or value function from both the "real" online experience and the "imaginary" experience generated by a model. Thus, both experience replay and Dyna-Q can be viewed as combinations of online and offline/batch reinforcement learning. Another advantage of batch reinforcement learning is that sometimes it facilitates theoretical analysis such as the sample complexity problem [Kakade, 2003; Kearns *et al.*, 2000], as well as applications of advanced supervised learning algorithms [Dietterich and Wang, 2002].

In this thesis, we consider a special case of batch RL assuming that the state space is sparsely *sampled* and the optimal action values for these sampled states are computed or at least estimated. The sampled states together with their optimal action values form the training set for supervised learning. For this reason, these sampled states are called *training states*. Formally, we will consider the training data provided in the form of

$$T_{Q^*} = \{\langle s, a, Q^*(s,a)\rangle \mid \forall s \in \mathcal{T}, \forall a \in \mathcal{A}\}, \tag{2.12}$$

where $\mathcal{T} \subset \mathcal{S}$ is a sparsely sampled state space.

Knowing the optimal action values may at first seem unrealistic. In practice, however, a technique called *full-trajectory-tree expansion* can be used to compute or estimate such values. Using this technique, all possible action sequences are applied to each training state, and in this way the optimal action values are computed or estimated. Note that in the infinite-horizon case where the action sequences can be infinitely long, the discount factor γ has to be strictly less than one, which implies that the optimal action values can be estimated to any desired precision by considering action sequences up to a limited depth (cf. the discussion of MC in Section 2.2.2).

Full-trajectory-tree expansion is especially useful for deterministic domains where good policies generalize well across problems of different sizes. Then the agent can start with problems of tractable state space and apply the expansion efficiently to obtain the information needed for batch reinforcement learning. Once a good policy is computed, it can generalize to problems with larger state spaces. There have been several successful applications of this technique including [Draper *et al.*, 2000; Levner and Bulitko, 2004; Wang and Dietterich, 1999].

For the value-function methods, once the training data T_{Q^*} are acquired, an optimal value function approximation $\widehat{Q}^*(s,a)$ can be computed using the standard supervised learning techniques [Dietterich and Wang, 2002]. For the classification-based methods, the class labels (optimal actions) can be computed:

$$\forall s \in \mathcal{T},\ a^*(s) = \arg\max_a Q^*(s,a),$$

and the training data for computing the optimal policy approximation (which is a classifier) are formed:

$$T_{\mathsf{CI}} = \{\langle s, a^*(s)\rangle \mid s \in \mathcal{T}\}. \tag{2.13}$$

The subscript CI (*cost-insensitive*) is in contrast to its *cost-sensitive* counterpart that will be introduced later in Chapter 4.

2.3 Function Approximation for Reinforcement Learning

Function approximation [Boyan *et al.*, 1995] in reinforcement learning is the theme of the thesis. In this section, we will first define the problem as well as the notation, and then present several important results in this line of research.

2.3.1 Why Function Approximation?

So far, we have considered solving the reinforcement learning problem using a tabular representation for policies and/or value functions. Good theoretical results as well as impressive empirical performance on small problems have been obtained. In solving real-world problems, however, function approximation becomes critical for the following reasons.

First, most real-world problems of practical interest have very large state spaces which render the tabular representation infeasible. For example, the size of the state space of backgammon is estimated to be over 10^{20} [Tesauro, 1995]. For such large problems, compact representations (or approximation) have to be used, including decision trees, artificial neural networks, etc.

Second, even if we can afford to use a lookup table, function approximation is still important for generalization. Note that many theoretical results are based on an assumption that the MDP is ergodic and every state is visited infinitely often. But in practice, "infinity" can never be achieved. Instead, the agent can first compute a good policy for a subspace of $\mathcal{S}$, and then generalize the policy to other states that are visited less frequently.

Third, for problems with a continuous state space, a lookup table cannot be used because (i) it is impossible to enumerate all states in a table, and (ii) it is unlikely that a state will be visited more than once. In such a case, function approximation is even more critical to the generalization ability to *unseen* states.

2.3.2 Reinforcement Learning with Function Approximation

Instead of lookup tables, function approximation (see Appendix A.4 for several examples) can be used as a compact representation parameterized by a k-dimensional vector $\theta \in \mathbb{R}^k$: the value functions are denoted by $V(s, \theta)$ or $Q(s, a, \theta)$, and the policy by $\pi(s, a, \theta)$ or more compactly, $\pi(\theta)$. During the learning process, θ is optimized according to some performance metric. A natural choice is to minimize the average value function approximation error or the temporal differences.

In particular, for the agent solving the policy evaluation problem using TD, the target error function can be the mean squared error of $V(s,\theta)$ or $Q(s,a,\theta)$:

$$\mathrm{Err}^{V}_{\mathrm{TD},\mu}(\theta) \quad = \quad \sum_{s\in\mathcal{S}} \mu(s)\big(V^{\pi}(s) - V(s,\theta)\big)^2,$$

or,

$$\mathrm{Err}^{Q}_{\mathrm{TD},\mu}(\theta) \quad = \quad \sum_{s\in\mathcal{S}} \mu(s,a)\big(Q^{\pi}(s,a) - Q(s,a,\theta)\big)^2,$$

where $\mu(s)$ and $\mu(s,a)$ are some distributions weighting the errors of different states or state-action pairs. Usually, they are the stationary distributions $\mu^{\pi}_{\infty}(s)$ and $\mu^{\pi}_{\infty}(s,a)$. A Sarsa agent may minimize the mean squared error of its optimal value function approximation:

$$\mathrm{Err}_{\textsc{Sarsa}}(\theta) \quad = \quad \sum_{s\in\mathcal{S}} \mu^{\pi}_{\infty}(s,a)\big(Q^{*}(s,a) - Q(s,a,\theta)\big)^2,$$

Correspondingly, it updates the parameter according to the stochastic gradient descent rule (Equation A.8):[4]

$$\theta_{t+1} = \theta_t + \alpha\big(Q^{*}(s_t,a_t) - Q(s_t,a_t,\theta_t)\big)\cdot\nabla Q(s_t,a_t,\theta_t).$$

Note that the state-action pairs (s_t,a_t) are sampled from a stationary distribution $\mu^{\pi}_{\infty}(s,a)$. If the step sizes α_t satisfy Assumption 4 (cf. Appendix A.4), then the θ_t updated by the rule above are guaranteed to converge to a local minimum of $\mathrm{Err}_{\textsc{Sarsa}}$. Typically, $Q^{*}(s_t,a_t)$ is unknown. An alternative is to use the TD error in place of the true error $Q^{*}(s_t,a_t) - Q(s_t,a_t,\theta_t)$:

$$\begin{aligned} \theta_{t+1} \quad &= \quad \theta_t + \alpha\cdot\delta^{\mathrm{TD}}_t\cdot\nabla Q(s,a,\theta) \\ &= \quad \theta_t + \alpha\big(r_{t+1} + \gamma Q(s_{t+1},a_{t+1},\theta_t) - Q(s_t,a_t,\theta_t)\big)\cdot\nabla Q(s_t,a_t,\theta_t). \qquad (2.14) \end{aligned}$$

2.3.3 Classification-based Approximate Policy Iteration

Recently, modern classification techniques have been successfully applied to the approximation framework of policy iteration using rollouts.

Approximate Policy Iteration

When function approximation is used, computing the *exact* greedy policy π' in policy iteration is infeasible. In this case, we may use a general framework of approximation within policy iteration known as *approximate policy iteration* (API) [Bertsekas and Tsitsiklis, 1996], which is similar to policy iteration, except that (i) the policy (or the value function) is represented by function approximators (in contrast to the lookup tables in the original policy iteration), and/or (ii) the value function is not computed exactly but is estimated.

[4]Throughout the thesis, the ∇ denotes the gradient with respect to the k-dimensional vector $\theta\in\mathbb{R}^k$.

```
**CI-cRL**
INPUT:
    $M$:   generative model
    $\mathcal{T}$:   training states
    $\gamma$:   discount factor
    $K$:   number of trajectories
    $H$:   maximum length of trajectories
OUTPUT: $\pi \approx \pi^*$
$\widehat{\pi}' \leftarrow$ random policy
repeat
    $\pi \leftarrow \widehat{\pi}'$
    $T \leftarrow \emptyset$
    for each $s \in \mathcal{T}$
        for each $a \in \mathcal{A}$
            $\widehat{Q}^{\pi}(s,a) \leftarrow \mathsf{Rollout}(M, s, a, \gamma, \pi, K, H)$
        $\hat{a}^*_{\pi} \leftarrow \arg\max_{a \in \mathcal{A}} \widehat{Q}^{\pi}(s,a)$
        for each $a \in \mathcal{A}$
            if $\widehat{Q}^{\pi}(s, \hat{a}^*_{\pi}) > \widehat{Q}^{\pi}(s,a)$
                $T \leftarrow T \cup \{\langle s, \hat{a}^*_{\pi} \rangle\}$
    $\widehat{\pi}' \leftarrow \mathsf{Learn}\ (T)$
until $\pi \approx \widehat{\pi}'$
return $\pi$
```

Figure 2.7: CI-cRL: Cost-insensitive classification-based RL. Learn is a sub-routine that induces a classifier from the input training data.

Unlike in the case of policy iteration, the approximate greedy policy $\widehat{\pi}'$ computed in API may be worse than the original policy π, due to the errors introduced by approximation. Such a problem is referred to as the *policy degradation problem* in the thesis.

Online Reinforcement Learning as Classification

Recent developments in the class of policy-search methods include the classification-based reinforcement learning where a policy π is a classifier mapping states to actions [Langford and Zadrozny, 2003]. Specifically, each state is labeled with the action $\pi(s)$. Then the task of learning the policy π is reduced to learning a classifier labeling the states. By using classifiers to represent policies and rollouts to estimate value functions, API can be implemented in a natural way. Figure 2.7 shows an example of API based on classification and rollouts [Lagoudakis and Parr, 2003b]. Learn is a sub-routine that induces a classifier from the input training data. Henceforth, we will call this algorithm CI-cRL (Cost-Insensitive classification-based RL).

2.3.4 Policy Gradient and Actor-Critic Method

Recently, a class of policy-search methods called *policy gradient* are receiving growing attention [Baxter and Bartlett, 1999; Baxter *et al.*, 1999; Berenji and Vengerov, 2003; Kakade, 2001; 2002; Williams, 1992]. Such methods perform gradient ascent on the policy value in the parameter space, and an immediate advantage is that they directly optimize the policy value. In addition, convergence is easier to guarantee.

In this section, we will introduce two such methods in more details: the policy gradient by Sutton et al. [Sutton *et al.*, 2000], and the actor-critic method by Konda & Tsitsiklis [Konda and Tsitsiklis, 2000; Konda, 2002]. Since these two pieces of work are very similar, both of them will be referred to as policy gradient in the thesis. When policy gradient is discussed (here and in Chapter 5), we always assume the policies satisfy the following assumption [Sutton *et al.*, 2000].

Assumption 3 *The* parameterized stochastic polices $\pi(s, a, \theta)$ *satisfy the following assumptions:*

1. *The probability of selecting any action under any policy is always nonzero:*

$$\forall s \in \mathcal{S},\ \forall a \in \mathcal{A},\ \forall \theta \in \mathbb{R}^k,\ \pi(s, a, \theta) > 0;$$

2. $\forall s, a,\ \nabla\pi$ *exists;* $\forall s, a, i, j,\ |\frac{\partial^2 \pi}{\partial \theta^i\, \partial \theta^j}|$ *is bounded. Furthermore,* $\forall s, a,$ *the* $\mathbb{R}^k$*-valued function* $\theta \to \nabla \ln \pi(s, a, \theta)$ *exists and is bounded;*

3. $\forall \theta \in \mathbb{R}^k$*, the Markov chains*[5] $\{s_t\}$ *and* $\{(s_t, a_t)\}$ *have stationary probabilities* $\mu^{\theta}_{\infty}(s)$ *and* $\mu^{\theta}_{\infty}(s, a) = \mu^{\infty}(s)\theta_{\infty}(s)\pi(s, a, \theta)$*, respectively;*

The assumption contains three parts. The first two are easily satisfied by designing $\pi(s, a, \theta)$ appropriately. For example, the Gibbs softmax policy is a choice often used (cf. Sections 2.2.1 and 5.2.4). The third part in the assumption is automatically satisfied if Assumption 2 holds. In fact, for an MDP with a fixed policy π, there exist at least two Markov chains: $\{s_t\}$ and $\{(s_t, a_t)\}$. A well-known result for Markov chains with a finite state space is that the stationary distribution μ_∞ exists if (i) every state is visited infinitely often, and (ii) the probability of visiting any state at time step t is eventually non-zero as $t \to \infty$. Further, the distribution μ_∞ is independent of the start-state distribution D. The reader is referred to the texts (e.g., [Isaacson and Madsen, 1976]) for formal definition and details of Markov chains. Assumptions of policies for more general MDPs with continuous state space is found in [Konda, 2002].

[5]A Markov chain can be seen as an MDP with a singleton set of actions, i.e., $|\mathcal{A}| = 1$.

The key step in the policy gradient method is to estimate the gradient of the policy value. A theorem [Konda and Tsitsiklis, 2000; Sutton *et al.*, 2000] states that[6]:

$$
\begin{aligned}
\nabla\mathcal{V} &= \sum_{s\in\mathcal{S}} \mu_\infty^\theta(s) \sum_a \big(\nabla\pi(s,a,\theta)\cdot Q^\theta(s,a)\big) &\quad (2.15)\\
&= \sum_{s\in\mathcal{S}}\sum_{a\in\mathcal{A}} \Big(\big(\mu_\infty^\theta(s)\cdot\pi(s,a,\theta)\big)\cdot Q^\theta(s,a)\cdot\frac{1}{\pi(s,a,\theta)}\nabla\pi(s,a,\theta)\Big)\\
&= \sum_{s\in\mathcal{S}}\sum_{a\in\mathcal{A}} \big(\mu_\infty^\theta(s,a)\cdot Q^\theta(s,a)\cdot\psi(s,a,\theta)\big), &\quad (2.16)
\end{aligned}
$$

where

$$\psi(s,a,\theta) = \frac{1}{\pi(s,a,\theta)}\nabla\pi(s,a,\theta) = \nabla\ln\pi(s,a,\theta)$$

and

$$\mu_\infty^\theta(s,a) = \mu_\infty^\theta(s)\cdot\pi(s,a,\theta).$$

A problem in estimating the gradient in Equation 2.15 is that the true action-value function, $Q^\theta(s,a)$, is unknown. Fortunately, it is shown [Konda and Tsitsiklis, 2000] that although $Q^\theta(s,a)$ is a high-dimensional vector[7] that may be difficult to learn, it affects the gradient $\nabla\mathcal{V}$ only through an inner product in a space with lower dimension (cf. Equation 2.16). In particular, the inner product is defined as

$$(Q,\psi)_\theta = \sum_{s\in\mathcal{S}}\sum_{a\in\mathcal{A}} \mu_\infty^\theta\cdot Q(s,a,\theta)\cdot\psi(s,a,\theta),$$

and it suffices to learn the projection of $Q^\theta(s,a)$ in a linear subspace with a dimension of k:

$$\widetilde{Q}^\theta(s,a,w^*) = w^*\cdot\psi(s,a,\theta), \tag{2.17}$$

where $w^*\in\mathbb{R}^k$ and $\psi(s,a,\theta)$ are called *features* under the policy parameterization. Any TD method can be used to estimate w^*. After w^* is computed or estimated, the policy gradient can be computed by:

$$\nabla\mathcal{V} = \sum_{s\in\mathcal{S}}\sum_{a\in\mathcal{A}} \Big(\mu_\infty^\theta(s,a)\cdot\widetilde{Q}^\theta(s,a,w)\cdot\psi(s,a,\theta)\Big), \tag{2.18}$$

Based on this insight, a family of policy gradient methods are proposed. All of them estimate w^* first, and then compute $\nabla\mathcal{V}$ by Equation 2.18. This approach has recently been applied to power control in wireless transmitters [Berenji and Vengerov, 2003].

[6]To simplify the notation, we also use θ in replace of the policy $\pi(s,a,\theta)$ if there is no ambiguity. For instance, the action-value function $Q^{\pi(s,a,\theta)}$ will be simplified as Q^θ, and the policy value $\mathcal{V}(\pi(\theta))$ will be rewritten as $\mathcal{V}(\theta)$, etc. This convention will also be adopted in Chapter 5

[7]A function defined on a discrete set can always be imagined as a vector. For example, $Q^\theta(s,a)$ can be seen as a $|\mathcal{S}\times\mathcal{A}|$-dimensional vector with one entry for each (s,a) pair.

2.4 Motivation

The rest of this chapter discusses the motivation for the thesis research. We will start with the empirical evidence in a complex, real-world system, showing that making the function approximation more accurate may not result in a better policy for sequential decision making. Similar phenomena were also observed for the game of backgammon [Weaver and Baxter, 1999]. Finally, a discussion of the application of function approximation in reinforcement learning as well as the thesis research objectives conclude this chapter.

2.4.1 Empirical Results in A Real-World System: MR ADORE

Recently, reinforcement learning has been applied to the problem of automated image interpretation and has shown promising results. The system MR ADORE (Multi-Resolution ADaptive Object REcognition) [Bulitko *et al.*, 2003; Levner and Bulitko, 2004], as an extension to ADORE (ADaptive Object REcognition) [Draper *et al.*, 2000], models the image interpretation process as an MDP, where the states are image tokens (e.g., images, lines, etc.) and the actions are image processing operators (e.g., thresholding, smoothing, equalization, etc.) The learning problem in such a system is to acquire a good control policy that selects which operator to apply at each image level.[8]

MR ADORE was designed with the following objectives: (i) rapid system development for a wide class of image interpretation domains; (ii) low demands for subject matter, computer vision, and AI expertise on the part of the developers; (iii) accelerated domain portability, system upgrades, and maintenance; (iv) adaptive image interpretation wherein the system adjusts its operation dynamically to a given image; (v) user-controlled trade-offs between recognition accuracy and resources utilized (e.g., time required).

The objectives above favor the use of readily available off-the-shelf image processing operator libraries (IPL). However, it is difficult to learn a good control policy because the task of automated image interpretation is both complex and adaptive [Levner, 2003]:

- It is *complex* in the sense that there is rarely a one-step mapping from input images to their interpretations; instead, a series of operator applications are required to bridge the gap between raw pixels and semantic objects.
- It is *adaptive* in that there is no fixed sequence of actions that will work well for most images. For instance, the steps required to locate and identify isolated trees are different from the steps required to find connected stands of trees.

[8]At the current stage, the policy is to choose fixed-length sequences of operators, which can be viewed as *macro-actions* [Hauskrecht *et al.*, 1998].

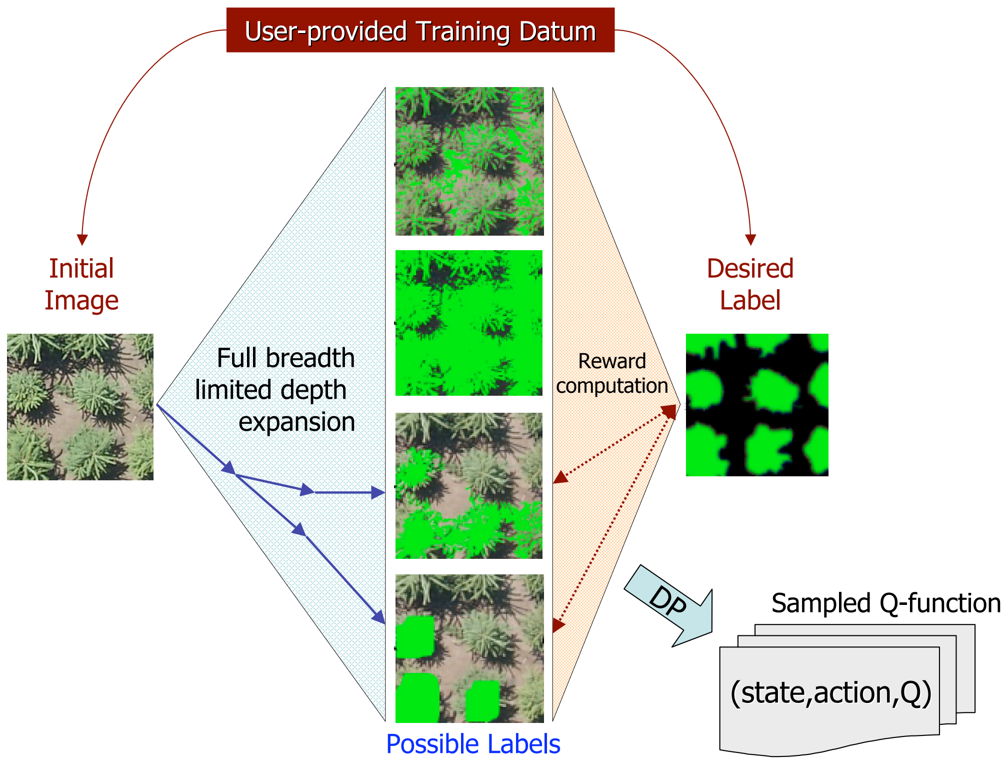


Figure 2.8: Offline operation of MR ADORE for policy acquisition.

Several challenges exist for MR ADORE, including:

- the raw state (image token) requires an order of 10^7 bytes for representation;
- the number of states is prohibitively large (e.g., the number of possible initial states is estimated to be up to $10^{7,200,000}$);
- the online actions (image processing) are expensive.

In response, the following techniques were taken (Figure 2.8 from [Levner, 2003]). First, MR ADORE uses training data (here, annotated images) to provide relevant domain information. Each training datum is a source image, annotated by an expert with the desired output. Figure 2.9 demonstrates a training datum in the forestry image interpretation domain.

Second, during the offline stage the state space is explored via limited depth expansions of training images. Within a single expansion, all sequences of actions (IPL operators) up to a certain user-controlled length are applied to the training image. Since training images are user-annotated with the desired output, terminal rewards can be computed based on

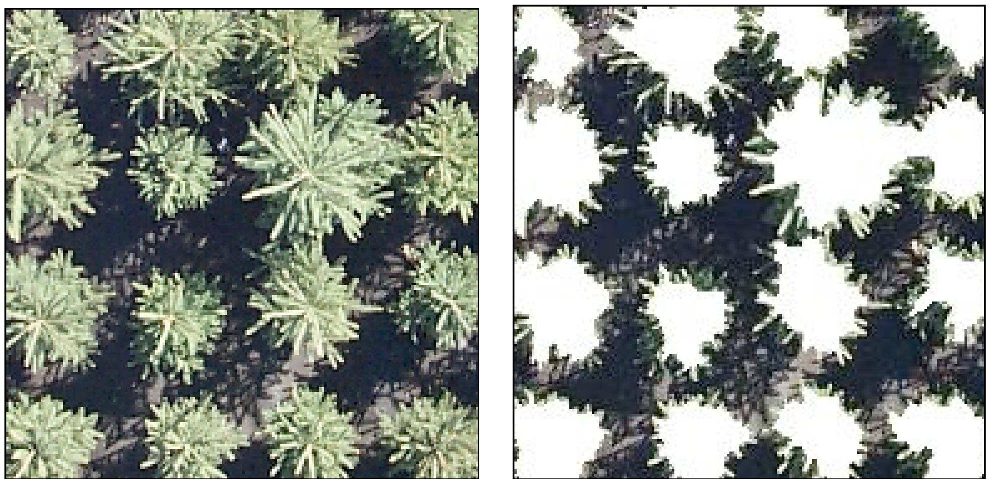

(a) Input image (b) Desired output

Figure 2.9: Training data used in MR ADORE. (a) An original photograph. (b) The corresponding desired labeling provided by an expert as a part of the training set.

the difference between the produced labeling and the desired labeling. Then, dynamic programming are used to compute the optimal value function for the explored parts of the state space. Note that MR ADORE does not use a discount factor, making the entire problem a non-discounted finite-horizon MDP.

Third, as the raw state descriptions are on the order of mega-bytes each, we first extract features $f(s)$ of each state s. Then supervised machine learning extrapolates the sampled Q^*-values, computed in the previous step, onto the entire state space. The resulting optimal value function approximation then is $Q(f(s), a, \theta)$.

Finally, when presented with a novel input image to interpret, MR ADORE first computes the abstracted state representation $f(s)$, and applies $Q(f(s), a, \theta)$ to estimate $Q^*(f(s), a)$ for each IPL operator a; then it performs the greedy action $a^* = \arg_a \max Q(f(s), a, \theta)$. The process terminates when the policy executes action `submit(⟨labeling⟩)` and the image token `⟨labeling⟩` becomes the system's output.

Experiments using Artificial Neural Networks

In the first set of experiments, multi-layer feed-forward neural networks were used as the regressors. Common sets of features including RGB-HISTOGRAM, HSV-HISTOGRAM, HSV-MEAN, textural features, etc. were used. Experiments were run with combinations of different features and neural network topologies. Thirty two forestry aerial images with user-annotated labeling were used. Since the training data are very limited, leave-one-out cross-validation was employed for evaluation. In each run, one image was selected for testing while the other thirty one images were used for training. Three performance metrics were measured[9]: the training error of $Q(s,a,\theta)$, the test error of $Q(s,a,\theta)$, and the relative value of the resulting control policy π_θ. The three performance metrics were then averaged over all the thirty two runs.

Experimental results showed that the approximated optimal policy achieved an average relative reward of over 85% on the forestry plantation data with HSV-histogram as features [Levner *et al.*, 2003]. It is indeed helpful to the success of MR ADORE that the function approximator approximates the optimal value function as accurately as possible. Therefore, we tried boosting to approximate $Q^*(s,a)$, hoping that obtaining a more accurate estimate $Q(s,a,\theta)$ results in a better control policy π_θ. However, this is not always the case [Li *et al.*, 2003], as shown in the next subsection.

Experiments using Boosted Artificial Neural Networks

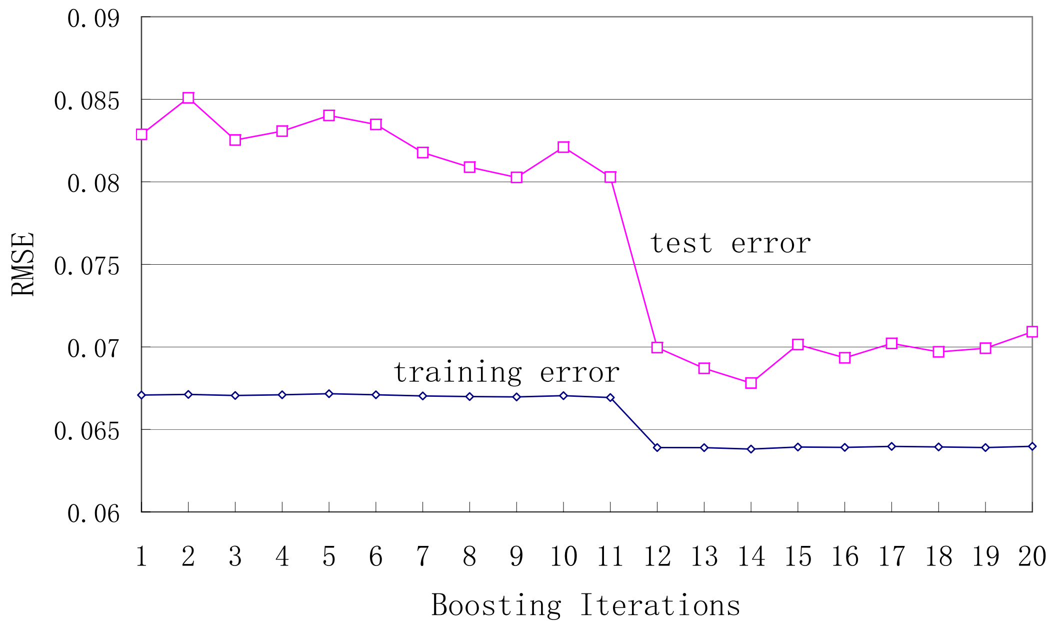
The second set of experiments is conducted using SquareLev.R [Duffy and Helmbold, 2002] on different sets of features, and with ANNs of different topologies. We observed that training and test errors were decreased significantly in almost all experiments. The resulting average relative policy value, however, did not increase for all features; sometimes we found that the relative reward even decreased. Figures 2.10 and 2.11 shows these two different cases.

These results show that boosting may not improve the value of a control policy even if the approximation error to the optimal value functions *does* decrease. Reinforcement learning problems behave differently from regression problems in terms of boosting/leveraging methods applied to the value function. Note that if the optimal value function can be learned with an arbitrary precision, then the resulting control policy can be made arbitrarily close to the optimal policy (cf. Section 3.1.2). In practice, however, the complexity of the problems frequently does not allow learning the optimal value function arbitrarily well. This section demonstrates that in such cases boosting methods can have an *opposite* effect on the policy value.

2.4.2 A Discussion of Supervised Learning in Reinforcement Learning

The problem discussed in the previous section was also noticed by other researchers. Weaver et al. [Weaver and Baxter, 1999] gave a two-state example to show that even if the linear function approximation in TD(λ) converges to a near-optimal solution in minimizing the temporal difference error of the value function approximation, it could converge to a

[9]Formal definitions of the common terms in supervised learning are found in Appendix A.



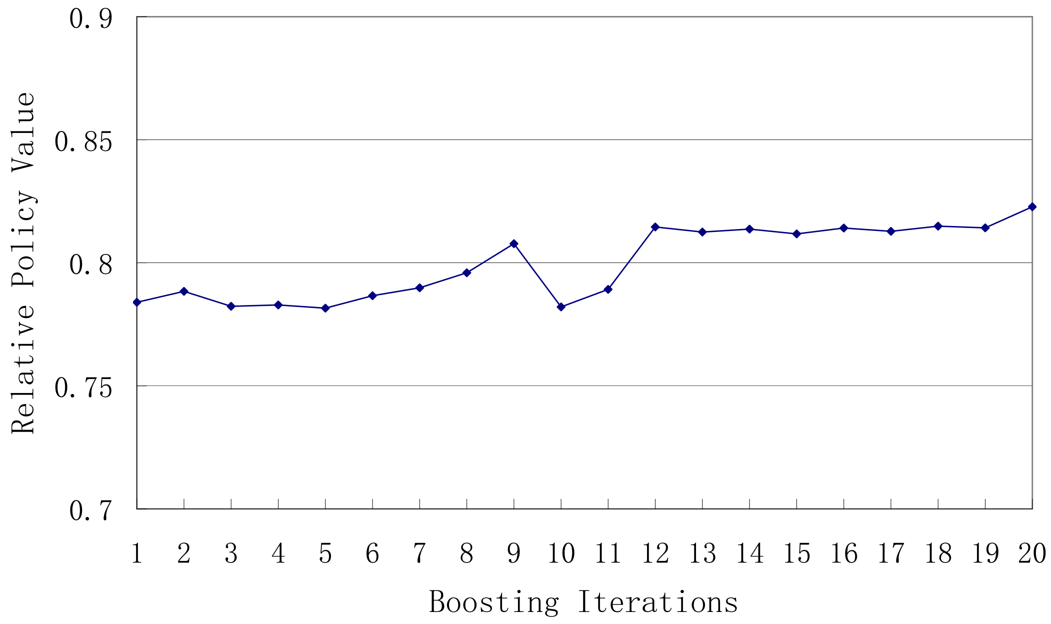
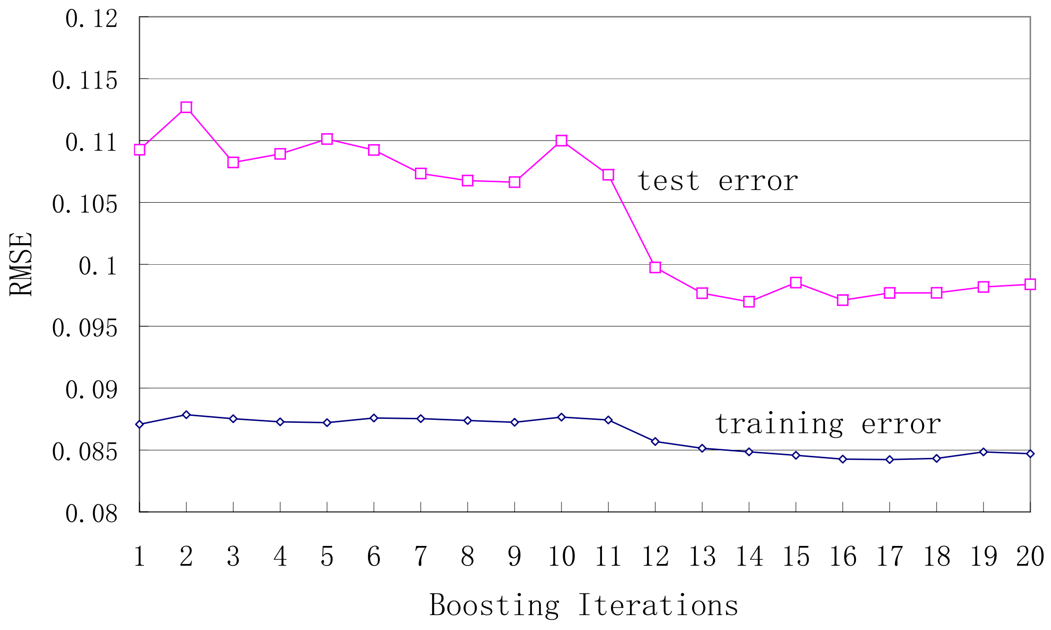


Figure 2.10: Empirical results of SquareLev.R applied to MR ADORE. When the training and test errors decrease, the relative policy value increases. See Appendix A.1 for a formal definition of RMSE (root mean squared error).



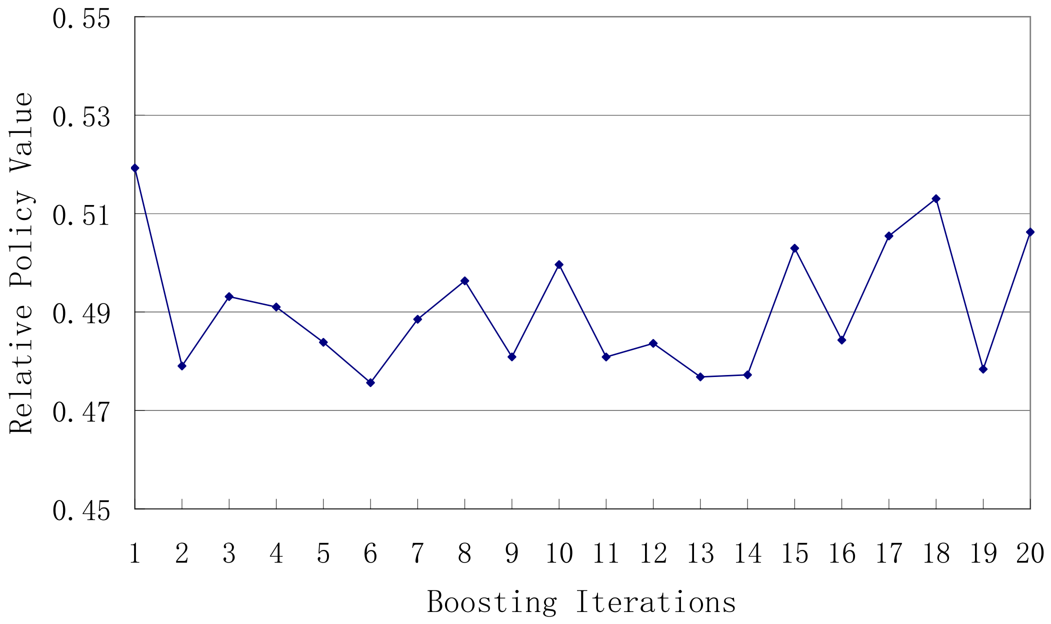


Figure 2.11: Empirical results of SquareLev.R applied to MR ADORE. Although the training and test errors decrease, the relative policy value can decrease slightly. See Appendix A.1 for a formal definition of RMSE (root mean squared error).

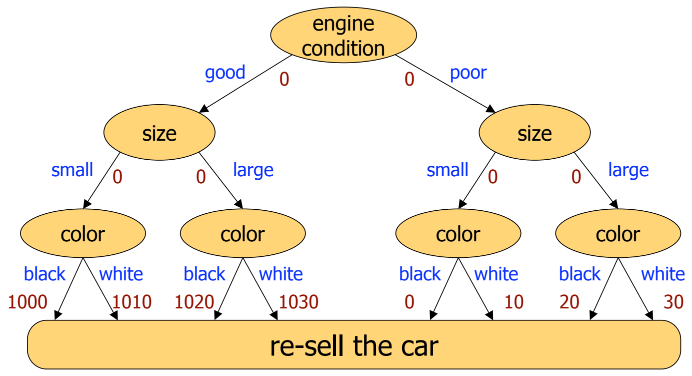


Figure 2.12: The car-shopping problem.

suboptimal policy although the optimal policy can be represented by the linear function approximator. More importantly, they observed a similar phenomenon when training their backgammon player program online. They found that although the estimated error in approximating the value function decreases by about 50% during training, the winning proportion of their game playing program, however, decreases significantly from 0.46 to 0.30 — a decline of about one-third. This suggests that the commonly used metrics, function approximation accuracy, TD errors, and the policy value, can conflict in practice. And this problem is *not* just hypothetical, bur rather, it does occur in real-life domains.

It would be helpful to look into a small and concrete example to understand why naive applications of function approximation in reinforcement learning can be problematic. We consider the classification-based policy search methods in the batch learning setting, and the agent is presented with a set of training data T_{Cl} (Equation 2.13). High classification accuracy is usually deemed to correlate with high policy value. But this is not true even for the toy problem below.

Figure 2.12 shows the car-shopping problem modelled as a three-step (finite-horizon), non-discounting MDP. Non-terminal states are labeled with the decisions the user is to make. The edges are labeled with the actions and the immediate rewards. The agent starts by choosing the engine condition and finishes with the need to re-sell the car. Starting with the state engine condition, she has two choices: good and poor. Then she decides on the size of the car small/large, and finally on its color black/white. After the choices are made, the agent buys the car and collects the final rewards by reselling the car.

The optimal policy π^* is shown in Table 2.1. If the agent uses π^*'s choices of action $a^*(s)$ in the training data set T_{Cl}, it may learn two approximate policies π_1 and π_2 (also shown in the table). Policy π_1 has the classification accuracy of 86 % and the policy value

	color	size	engine condition
optimal policy π^*	white	large	good
policy π_1	white	large	poor
policy π_2	black	small	good

Table 2.1: Optimal and two approximate policies for the car-shopping problem.

$V(\pi_1)$ of 30. Policy π_2, on the other hand, is considerably less accurate (14%) but enjoys a much higher policy value of $1,000$.

2.4.3 Research Objectives

An intuitive explanation for the observed phenomenon in the car-shopping problem is that not all states are equally important in terms of affecting the policy value. It is beneficial for the agent to increase the classification accuracy and agree with the optimal policy in more states. However, it can be more crucial to agree with the optimal policy in states that are more *important* in affecting the policy value.

The goal of this thesis is to investigate the policy value by *focusing* the learning process on *more important* states. In doing so, we:

- introduce a precise definition of decision-making importance of a state;
- propose a novel family of algorithms that focus learning on more important states;
- evaluate importance-sensitive learning empirically and theoretically.

Chapter 3

Related Work

This chapter reviews some of the more important related work on function approximation for reinforcement learning. Two problems will be introduced: the divergence problem and the sub-optimality problem. Then we will discuss the limitations in previous work, followed by some concluding remarks.

3.1 Two Problems with Function Approximation in Reinforcement Learning

Function approximation is critical to the application of reinforcement learning in many real-world problems with large or even continuous state spaces. However, two problems arise in the presence of function approximation.

3.1.1 The Divergence Problem

Several theoretical results about convergence to the optimal solution apply to the case of tabular representation [Bertsekas and Tsitsiklis, 1996; Singh *et al.*, 2000; Sutton and Barto, 1998]. Unfortunately, these properties may not be guaranteed when function approximation is used. Several simple counter-examples were found where dynamic programming or TD learning do not converge [Baird, 1995; Boyan and Moore, 1995; Gordon, 1996; 2001; Tsitsiklis and Van Roy, 1997], even with very simple function approximators. Although in practice, function approximation for RL can work well (e.g., [Crites and Barto, 1996; Sutton, 1996; Tesauro, 1995; Zhang and Dietterich, 1995]), it becomes much less assuring without such a convergence guarantee.

The simplest divergence example using dynamic programming in policy evaluation was given by [Tsitsiklis and Van Roy, 1997]. The example is illustrated in Figure 3.1. This is a discounting MDP with three states: the grayed square is the terminal state, and the

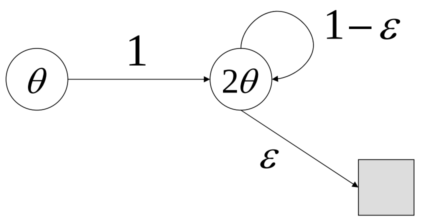


Figure 3.1: Tsitsiklis and van Roy's example for illustrating the divergence of dynamic programming with least-squares linear function approximation.

other two (circles) are non-terminal. Actions (edges) are labeled with the state transition probabilities. Immediate rewards are all zero on all transitions and therefore, the value function $V^\pi(s) = 0$ for any policy π. A linear function approximation is used, and the only parameter for tuning is a scalar θ. Clearly, if $\theta = 0$, then the function approximation represents the true value function exactly. However, when a simple least-squares technique is combined with Monte Carlo to update the parameter, the sequence $\{\theta_t\}$ diverges if $\gamma > 5/(6 - 4\varepsilon)$ and $\theta_0 \neq 0$.

In order to address this problem, Baird & Moore [Baird, 1995] proposed the *residual gradient algorithm*, and more generally, the family of *residual algorithms*. The target of residual gradient algorithm is to perform gradient descent on the *mean squared Bellman residual* (the next state is denoted by s'):

$$\begin{aligned} & Err_{\text{MSBR}}(\theta) \\ = & \sum_{s\in\mathcal{S}} \Big(\mu_\infty^\pi(s) \cdot \big(\mathbf{E}_{a,r,s'\sim\pi}\{r + \gamma V(s',\theta)\} - V(s,\theta)\big)^2\Big) \quad (3.1) \\ = & \sum_{s\in\mathcal{S}} \left(\mu_\infty^\pi(s) \cdot \left(\sum_{a\in\mathcal{S}} \pi(s,a) \left(R(s,a) + \gamma \sum_{s'\in\mathcal{S}} P_{sa}(s')V(s',\theta)\right) - V(s,\theta)\right)\right). \quad (3.2) \end{aligned}$$

The corresponding online update rule is:

$$\begin{aligned} \theta_{t+1} &= \theta_t - \alpha_t \cdot \nabla Err_{\text{MSBR}} \\ &= \theta_t - \alpha_t \cdot \big(r_{t+1} + \gamma V(s_{t+1},\theta_t) - V(s_t,\theta_t)\big) \cdot \big(\gamma \nabla V(s_{t+1},\theta_t) - \nabla V(s_t,\theta_t)\big). \quad (3.3) \end{aligned}$$

Since this update rule performs stochastic gradient descent on $Err_{\text{MSBR}}(\theta)$, it is guaranteed to converge to a local minimum as long as the step-size parameters α_t satisfy Assumption 4. However, residual gradient may be slow in learning [Baird, 1995]. A possible improvement is the residual algorithm family which is a linear combination of Equations 3.3 and 2.14.

One improvement was made by Tsitsiklis and van Roy [Tsitsiklis and Van Roy, 1997], who proved that TD with linear function approximation is guaranteed to converge to a near-optimal solution. Indeed, they showed that in the limit, the TD update rule is a contraction operation in a linear space spanned by the state features. With such a theoretical guarantee, linear function approximations appear suitable for reinforcement learning.

There have been other attempts to tackle the divergence problem in reinforcement learning. For example, stability (non-divergence) is guaranteed if the function approximation does not extrapolate from observed target values. These methods such as the nearest neighbor technique [Cover and Hart, 1967; Gordon, 1995] are not that popular as neural networks or linear function approximations for real-valued function learning.

3.1.2 The Suboptimality Problem

Another potential problem in using value function approximation is that the function approximator cannot be expected to represent the target policy or value function exactly. Therefore, it remains important to investigate how the approximation errors affect the resulting policy value.

Singh and Yee [Singh and Yee, 1994] proved that if a good approximation of the optimal value function can be obtained for a stationary MDP with stationary deterministic policies, then a reasonable policy performance can be guaranteed. In particular, they proved that if

$$\exists\epsilon > 0,\ \forall s \in \mathcal{S},\ \ |V^*(s) - V(s,\theta)| \leq \epsilon, \tag{3.4}$$

then

$$\mathcal{L}(\pi_\theta) < \frac{2\gamma\epsilon}{1-\gamma}. \tag{3.5}$$

Williams and Baird [Williams and Baird, 1993] gave another theoretical bound on the policy loss based on the Bellman residual/error defined as:

$$\mathrm{Br}(s,\theta) = \max_a \left\{ R(s,a) + \gamma \sum_{s'} P_{sa}(s')V(s',\theta) \right\} - V(s,\theta). \tag{3.6}$$

Intuitively, $\mathrm{Br}(s,\theta)$ measures to what degree the Bellman optimality equation (Equation 2.3) is violated by $V(s,\theta)$. Note that if $V(s,\theta) = V^*(s)$, then $\mathrm{Br}(s,\theta) \equiv 0$, and vice versa. A theorem states that if

$$\exists\epsilon > 0,\ \forall s \in \mathcal{S},\ \ |\mathrm{Br}(s,\theta)| \leq \epsilon, \tag{3.7}$$

then

$$\mathcal{L}(\pi_\theta) \quad \leq \quad \frac{2\gamma\epsilon}{1-\gamma}. \tag{3.8}$$

These two bounds above suggest that the resulting policy loss is upper bounded by a simple function of the errors in Equations 3.4 or 3.7. Therefore, in order to maximize the policy value $\mathcal{V}(\pi)$, or equivalently, to minimize the policy loss $\mathcal{L}(\pi)$, the RL agent can instead try to minimize the upper bound of $|V^*(s) - V(s,\theta)|$ or $|\mathrm{Br}(s,\theta)|$.

3.2 Limitations of Previous Work

Although the previous work on the suboptimality problem introduced in the previous section is interesting and important in furthering understanding of function approximation in reinforcement learning, it is limited for two reasons.

First, the two bounds in Equations 3.5 and 3.8 depend on the $\mathcal{L}_\infty$-norm (max-norm) of $V^*(s) - V(s, \theta)$ and $\mathrm{Br}(s, \theta)$. However, a great majority of supervised learning algorithms for regression aims at reducing the $\mathcal{L}_2$-norm of the error (cf. Appendix A). An inconsistency exists between the norm required by the error bounds and the norm that popular supervised learning techniques minimize. A tractable solution to this *norm incompatibility* problem was proposed by Guestrin, Koller and Parr [Guestrin *et al.*, 2001] for factored MDPs, a restricted class of MDPs. But a general efficient approach remains unknown.

Second, even if a supervised learning algorithm for minimizing the $\mathcal{L}_\infty$-norm error is used, this max-norm can be quite large in real-world problems where the true function is too complex to be captured by the function approximator. Therefore, as the agent is increasing the accuracy of value function approximation, the resulting policy value can still degrade.

An important issue closely related to the motivation of the thesis (cf. Section 2.4) is the difference between supervised learning and reinforcement learning. Supervised learning focuses on one-stage decision making. The performance of a learning algorithm can be defined straightforwardly, such as the classification error or the (root) mean squared error. For sequential decision-making problems, however, the target performance is the policy value. If supervised learning methods such as boosting are applied *naively*, the learning agent can end up with a worse policy, in the sense of sequential decision making, by spending more resources in computing a more *accurate* value function or policy approximation, in the sense of supervised learning. Such an observation also explains the paradox in the car-shopping problem as well as the findings in MR ADORE and the game of backgammon (cf. Section 2.4).

STD(λ)

After observing the policy degradation phenomena in backgammon, Weaver et al. argued that this problem comes from the fact that TD(λ) takes no account of the value of the policy derived from the function approximation. Thus, they proposed STD(λ) (state temporal difference learning), which is a variation to the classical TD(λ). The idea of STD is that the agent learns the state value differences, $V^\pi(s_1) - V^\pi(s_2)$, instead of the state values $V^\pi(s)$. More specifically, they considered only the binary MDPs in which at most two successor

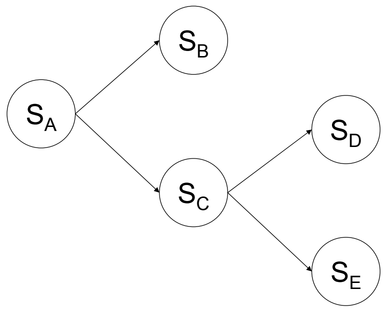


Figure 3.2: An illustrative example for the problematic STD(λ) algorithm.

states are possible at any time. Then STD(λ) tries to minimize:

$$\mathrm{Err}_{\mathrm{STD}} = \sum_{s,s'} \mu^{\theta}_{\infty}(s,s')\Big(\big(V(s,\theta)-V(s',\theta)\big)-\big(V^{\pi}(s)-V^{\pi}(s')\big)\Big)^2, \tag{3.9}$$

where $\mu^{\theta}_{\infty}(s,s')$ is the stationary probability that state s is visited with s' being its sibling state by following policy π. In Figure 3.2, for instance, states s_{B} and s_{C} are sibling states. Specifically, STD depends on the following Bellman equation:

$$V^{\pi}(s_t)-V^{\pi}(s'_t) = \big(R(s_t,a_t)-R(s'_t,a'_t)\big)+\gamma\big(V^{\pi}(s_{t+1})-V^{\pi}(s'_{t+1})\big). \tag{3.10}$$

Note that the actions a_t and a'_t above are chosen according to the policy π; s'_t and s'_{t+1} are sibling states of s_t and s_{t+1}, respectively.

A closer examination of Equation 3.10 may find it problematic. STD adopts the idea from differential training [Bertsekas, 1997]. Suppose two state transitions are observed:

$$\begin{aligned} s_t &\rightarrow r_{t+1}, s_{t+1}, \\ s'_t &\rightarrow r'_{t+1}, s'_{t+1}, \end{aligned}$$

and the respective Bellman equations are:

$$\begin{aligned} V^{\pi}(s_t) &= r_{t+1}+\gamma V^{\pi}(s_{t+1}) \\ V^{\pi}(s'_t) &= r'_{t+1}+\gamma V^{\pi}(s'_{t+1}). \end{aligned}$$

Subtracting these two equations yields:

$$V^{\pi}(s_t)-V^{\pi}(s'_t) = (r_{t+1}-r'_{t+1})+\gamma(V^{\pi}(s_{t+1})-V^{\pi}(s'_{t+1})). \tag{3.11}$$

Equation 3.11 can be viewed as the Bellman equation for a new MDP. But in STD a similar form of Equation 3.11 does not hold. Consider the example in Figure 3.2. STD attempts to update $V(s_{\mathrm{B}},\theta)-V(s_{\mathrm{C}},\theta)$ by using $V(s_{\mathrm{D}},\theta)-V(s_{\mathrm{E}},\theta)$. But it is clear that these two quantities do not have any relation in general and thus the Bellman equations underlying STD(λ) are not justified and the algorithm is problematic.

3.3 Conclusions

In this chapter, we briefly reviewed the related work on approximating the value function in solving complex reinforcement learning problems. We also discussed the limitations of previous work. The most related problem is that they have not taken into account the policy value. STD(λ), which aims at improving the policy value, is limited to binary MDPs, and more importantly, its underlying Bellman equation seems incorrect. In the next two chapters, we will consider how policy values can be improved by focusing attention in more important states, for the case of classification-based and value-function methods.

Chapter 4

Focus of Attention in Classification-based Reinforcement Learning

This chapter formalizes the idea of focused learning and investigates how to improve policy value by focusing attention in classification-based policy learning [Li *et al.*, 2004a; 2004b]. We will first give a theorem that forms the foundation for later development. Then we examine the batch reinforcement learning and online reinforcement learning cases, respectively. For each case, we define a metric for measuring the *sequential decision-making importance* of a state; several theorems explains why utilizing state importance is helpful. Following are experimental results for both cases in a grid-world domain. Proofs of all theorems will be provided in Section 4.5.

For simplicity, only binary-action problems with deterministic policies are examined. A short discussion of the extension to MDPs with stochastic policies and multiple actions is found at the end of the chapter, but a detailed treatment will be in the next chapter.

4.1 The Policy Switching Theorem

Theorem 1 below forms the basis of our proposed methods. It establishes how the policy value changes after a switch between two policies. For simplicity, we only give the results and proofs for discounted, infinite-horizon MDPs with binary actions, but the theorems and methodologies still apply to non-discounted finite-horizon MDPs with multiple actions.

Theorem 1 *Let ν and τ be two arbitrary policies for a discounted, infinite-horizon MDP, and an agent changes its policy from ν to τ. Define $G^{\nu}(s, \nu \to \tau) = Q^{\nu}(s, \tau(s)) - Q^{\nu}(s, \nu(s))$,* *then*

$$\mathcal{V}(\tau) - \mathcal{V}(\nu) = \sum_{s \in \mathcal{S}} \left(G^{\nu}(s, \nu \to \tau) \cdot \sum_{t=0}^{\infty} \gamma^t \mu_t^{\tau, D}(s) \right), \tag{4.1}$$

where the state visitation distribution, $\mu_t^{\tau,D}(s)$, is the probability that state s is visited at time t by following policy τ with start states drawn randomly according to D.

This theorem parallels a result in [Kakade and Langford, 2002], and a concept similar to $G^{\pi}(s, \pi \to \tau)$ was introduced in [Baird, 1993] and called *advantage*. To simplify the notation, let

$$d^{\tau,D}(s) = \sum_{t=0}^{\infty} \gamma^t \cdot \mu_t^{\tau,D}(s).$$

An interesting observation is that the policy improvement theorem (cf. Section 2.2.2) can be easily derived from Theorem 1. Namely, let ν and τ be a policy π and its greedy policy π' computed by Equation 2.10. Then $G^{\pi}(s, \pi \to \pi') = Q^{\pi}(s, \pi'(s)) - Q^{\pi}(s, \pi(s))$, and Equation 4.1 becomes:

$$\mathcal{V}(\pi') - \mathcal{V}(\pi) = \sum_{s \in \mathcal{S}} \left(\left(Q^{\pi}(s, \pi'(s)) - Q^{\pi}(s, \pi(s)) \right) \cdot d^{\pi',D}(s) \right).$$

Since π' is the greedy policy and $\mu_t^{\pi',D}$ is a state visitation distribution, it always holds for all states $s \in \mathcal{S}$ that $Q^{\pi}(s, \pi'(s)) - Q^{\pi}(s, \pi(s)) \geq 0$ and $d^{\pi',D}(s) \geq 0$. For ergodic MDPs where $d^{\pi',D}(s) > 0$, $\mathcal{V}(\pi') - \mathcal{V}(\pi) > 0$ unless $G^{\pi}(s, \pi \to \pi') \equiv 0$ in which case π satisfies the Bellman optimality equation 2.4 and thus is the greedy policy π^*.

4.2 Focusing Attention: Batch Reinforcement Learning

In Chapter 2, we demonstrated that not all states are equally important in the sense of sequential decision making. In this section, we will propose a novel batch reinforcement learning algorithm based on *cost-sensitive* classification that focuses on more important states. More specifically, the decision-making importance of states is used as the misclassification cost. As a result, the learning algorithm is able to focus on more important states thereby improving the convergence speed and the policy value. The exposition is done in two steps as follows. First, we show that the global metric of policy loss has a strong relation to the importance of individual states. Then an approximation step is taken to make the algorithm practical.

State Importance in Batch Reinforcement Learning

The first step is to introduce a formal measure of state *importance*. Intuitively, a state is important from the decision-making point of view if making a wrong decision in it can have significant repercussions. Therefore, the *importance* of a state s, $G^*(s)$, can be defined as the difference in the optimal values of $a^*(s)$ and $\bar{a}(s)$, where $a^*(s)$ is the optimal action and $\bar{a}(s)$ is the other (sub-optimal) action.[1]

Definition 10 *The* importance *of a state s is defined as:*

$$G^*(s) = Q^*(s, a^*(s)) - Q^*(s, \bar{a}(s)), \tag{4.2}$$

Likewise, $G^*(s, \pi)$ is defined as the difference in the optimal values of $a^*(s)$ and $\pi(s)$:

$$G^*(s, \pi) = Q^*(s, a^*(s)) - Q^*(s, \pi(s)) = \left\{ \begin{array}{ll} G^*(s), & \pi(s) \neq \pi^*(s) \\ 0, & \pi(s) = \pi^*(s) \end{array} \right. . \tag{4.3}$$

Expressing Policy Loss Through State Importance

Like the policy value, the policy loss (Equation 2.9) is a *global* attribute of a policy insomuch as it is computed over a distribution μ of start states, and for each state s, computing $V^\pi(s_0)$ involves the decisions made in other states as well as rewards gathered at each time step along the trajectory. This fact limits the use of policy loss in incremental policy learning since $\mathcal{L}(\pi)$ is not conveniently evaluated. Consequently, a direct use of the policy loss as the optimization criterion in classification learning would require recomputing it over all states every time a change to the classifier is made. Without further approximation this procedure is intractable.

The non-locality of policy loss can be addressed by approximating it with a local and computationally feasible metric. We develop such an approximation in two steps. First, the policy loss will be expressed through the importance of individual states. We have the following result.

Theorem 2 *The policy loss can be expressed through policy importance as follows:*

$$\mathcal{L}(\pi) = \sum_{s \in \mathcal{S}} \left(G^*(s, \pi) \cdot \sum_{t=0}^{\infty} \gamma^t \, \mu_t^{\pi, D}(s) \right) = \sum_{s \in \mathcal{S}} \left(G^*(s, \pi) \cdot d^{\pi, D}(s) \right). \tag{4.4}$$

Approximating Policy Loss

So far, we have represented the global metric of policy loss via policy importance of individual states. The representation, however, is still of a limited practical value since $d^{\pi,D}(s)$ is

[1]NB: we are considering the binary action case in this chapter.

```
CS
INPUT:
    $T_{Q^*}$:   training data
OUTPUT: $\widehat{\pi}^* \approx \pi^*$
$T_{\mathsf{CS}} \leftarrow \emptyset$
for each $s \in \mathcal{T}$
    $a^* \leftarrow \arg\max_a Q^*(s,a)$
    $\bar{a} \leftarrow$ the other (suboptimal) action
    $g \leftarrow Q^*(s,a^*) - Q^*(s,\bar{a})$
    $T_{\mathsf{CS}} \leftarrow T_{\mathsf{CS}} \cup \{\langle s, a^*, g\rangle\}$
$\widehat{\pi}^* \leftarrow$ CS-Learn($T_{\mathsf{CS}}$)
return $\widehat{\pi}^*$
```

Figure 4.1: CS: Cost-Sensitive batch RL based on classification. CS-Learn is a sub-routine that induces a cost-sensitive classifier from the input training data.

usually not available to the agent. Therefore, we propose an approximation to minimizing the policy loss by minimizing the *upper bound* of $\mathcal{L}(\pi)$:

$$\mathcal{L}(\pi) \leq \sum_{s\in\mathcal{S}} \left(G^*(s,\pi) \cdot \sum_{t=0}^{\infty} \gamma^t \right) = \frac{1}{1-\gamma} \sum_{s\in\mathcal{S}} G^*(s,\pi). \tag{4.5}$$

Correspondingly, a practical approach is as follows:

$$\begin{aligned} \widehat{\pi}^*_{\mathsf{CS}} &= \arg\min_{\widehat{\pi}^*\in\Pi} \sum_{s\in\mathcal{S}} G^*(s,\widehat{\pi}^*) \\ &= \arg\min_{\widehat{\pi}^*\in\Pi} \frac{1}{1-\gamma} \sum_{s\in\mathcal{S}} G^*(s,\widehat{\pi}^*) \\ &\approx \arg\min_{\widehat{\pi}^*\in\Pi} \mathcal{L}(\widehat{\pi}^*). \end{aligned} \tag{4.6}$$

Using the definition of $G^*(s,\pi)$ in (4.2), computing $\widehat{\pi}^*_{\mathsf{CS}}$ becomes the cost-sensitive classification problem with the *misclassification costs* conditional on individual cases [Turney, 2000]. Indeed, s is the attribute, $a^*(s)$ is the desired class label, and $G^*(s)$ is the misclassification cost. Thus, given a set of training data T_{Q^*} described in (2.12), the agent can first compute $G^*(s)$ for all states $s \in \mathcal{T}$ by (4.2) forming a training set:

$$T_{\mathsf{CS}} = \{\langle s, a^*(s), G^*(s)\rangle \mid s \in \mathcal{T}\} \tag{4.7}$$

and then solve the optimization problem (4.6). The corresponding importance-sensitive algorithm, called CS, is summarized in Figure 4.1. It calls a subroutine, CS-Learn, which is a cost-sensitive classification algorithm.

In contrast, a *naive*, importance-insensitive agent will solve the following classification problem based on T_{CI} (Equation 2.13):

$$\widehat{\pi}^*_{\mathsf{CI}} = \arg\min_{\widehat{\pi}^*} \sum_{s\in\mathcal{S}} \mathcal{I}(\widehat{\pi}^*(s) \neq \pi^*(s)), \tag{4.8}$$

where $\mathcal{I}$ is the indicator function defined in Equation A.3. The resulting algorithm, called CI, is the same as CS except that it uses a cost-insensitive classification algorithm Learn to induce the policy $\widehat{\pi}^*$.

A question of both theoretical and practical interest is therefore: Is it preferable to solve $\widehat{\pi}^*_{\mathsf{CS}}$ (4.6) as opposed to $\widehat{\pi}^*_{\mathsf{CI}}$ (4.8)? Theorem 3 below connects the quality of the classifier to the resulting policy value, and provides an upper bound on the policy loss of $\widehat{\pi}^*_{\mathsf{CS}}$. In contrast, Theorem 4 establishes that $\widehat{\pi}^*_{\mathsf{CI}}$ can be arbitrarily poor in the sense that the policy loss can be arbitrarily close to its upper bound given by Equation 4.5, as long as the classification error of $\widehat{\pi}^*_{\mathsf{CI}}$ is non-zero.

Theorem 3 *If $\widehat{\pi}^*_{\mathsf{CS}}$ has a sufficiently high quality:*

$$\exists\epsilon > 0, \quad \text{s.t.} \quad \frac{\sum_{s\in\mathcal{S}} G^*(s, \widehat{\pi}^*_{\mathsf{CS}})}{\sum_{s\in\mathcal{S}} G^*(s)} < \epsilon, \tag{4.9}$$

then

$$\mathcal{L}(\widehat{\pi}^*_{\mathsf{CS}}) \quad \leq \quad \frac{\epsilon}{1-\gamma} \sum_{s\in\mathcal{S}} G^*(s). \tag{4.10}$$

Theorem 4 *Let*

$$\epsilon = \frac{1}{|\mathcal{S}|} \sum_{s\in\mathcal{S}} \mathcal{I}(\widehat{\pi}^*_{\mathsf{CI}}(s) \neq \pi^*(s)) \tag{4.11}$$

be the classification error of $\widehat{\pi}^*_{\mathsf{CI}}$, then $\forall\epsilon, \xi > 0$, there exists an MDP and $\widehat{\pi}^*_{\mathsf{CI}}$, s.t.

$$\mathcal{L}(\widehat{\pi}^*_{\mathsf{CI}}) \quad > \quad \frac{1-\xi}{1-\gamma} \sum_{s\in\mathcal{S}} G^*(s). \tag{4.12}$$

4.3 Focusing Attention: Online Reinforcement Learning

In this section, we extend the idea of focused learning to the online reinforcement learning problem. Note that in the online RL setting, the optimal action values are unknown and therefore, the state importance cannot be computed by Equation 4.2. Below we will give another suitable definition of state importance. This extension is supported by several theorems demonstrating how $G^\pi(s)$ can be used as the misclassification costs to efficiently improve policy values in API. A novel algorithm is then proposed.

The classification-based API framework (cf. Section 2.3.3) will be considered. In the inner loop of CI-cRL (Figure 2.7), a high-accuracy classifier returned by Learn is used to approximate the greedy policy π'. If the classifier has a low classification error, the resulting policy $\widehat{\pi}'$ has a high probability of taking the same action as π'. Hence, it is expected that a more accurate classifier results in a better approximate greedy policy. In summary, the CI-cRL agent is importance insensitive and seeks:

$$\begin{aligned} \widehat{\pi}^*_{\mathsf{CI-cRL}} &= \arg\max_{\widehat{\pi}' \in \Pi} P(\widehat{\pi}'(s) = \pi'(s)) && (4.13) \\ &= \arg\min_{\widehat{\pi}' \in \Pi} P(\widehat{\pi}'(s) \neq \pi'(s)). && (4.14) \end{aligned}$$

In the following subsections, we will derive an importance-sensitive counterpart of CS-cRL that can result in better policies by focusing on important states.

State Importance in Online Reinforcement Learning

In order to analyze policy improvement within API formally, we define the importance of state s under policy π.

Definition 11 *In online reinforcement learning, the* importance *of a state s is defined as:*

$$G^\pi(s) = Q^\pi(s, a^*_\pi(s)) - Q^\pi(s, \bar{a}_\pi(s)), \tag{4.15}$$

where $a^*_\pi(s) = \arg\max_a Q^\pi(s,a)$ is the greedy action and $\bar{a}_\pi(s) \neq \arg\max_a Q^\pi(s,a)$ is the non-greedy action, with respect to the policy π. Likewise, for any policy τ, $G^\pi(s,\tau)$ is defined as the difference in the values of $a^*_\pi(s)$ and $\tau(s)$ with respect to policy π:

$$G^\pi(s,\tau) = Q^\pi(s, a^*_\pi(s)) - Q^\pi(s, \tau(s)) = \begin{cases} G^\pi(s), & \tau(s) \neq a^*_\pi(s) \\ 0, & \tau(s) = a^*_\pi(s) \end{cases}. \tag{4.16}$$

Intuitively, $G^\pi(s)$ measures how much additional return can be obtained by *switching* the action $\bar{a}_\pi(s)$ to $a^*_\pi(s)$ in state s, and then following the policy π thereafter. The following theorem relates the policy importance of $\widehat{\pi}'$ in each state to the difference of policy value between π' and $\widehat{\pi}'$.

Theorem 5 *If during policy improvement a policy π is changed to $\widehat{\pi}'$ which is an approximation to the greedy policy π' w.r.t. π, and*

$$\forall s \in \mathcal{S},\ |d^{\pi',D}(s) - d^{\widehat{\pi}',D}(s)| < \varepsilon, \tag{4.17}$$

then

$$\mathcal{V}(\pi') - \mathcal{V}(\widehat{\pi}') \leq \sum_{s\in\mathcal{S}} G^\pi(s,\widehat{\pi}') \cdot d^{\widehat{\pi}',D}(s) + \varepsilon \sum_{s\in\mathcal{S}} G^\pi(s). \tag{4.18}$$

CS-cRL: Cost-Sensitive Classification-based Reinforcement Learning

Theorem 5 bounds $\mathcal{V}(\pi') - \mathcal{V}(\widehat{\pi}')$ via the policy importance in each individual state. Note that π' and $\mathcal{V}(\pi')$ are determined if π is fixed. Therefore,

$$\arg\max_{\widehat{\pi}'} \mathcal{V}(\widehat{\pi}') \equiv \arg\min_{\widehat{\pi}'} \left\{ \mathcal{V}(\pi') - \mathcal{V}(\widehat{\pi}') \right\}.$$

In other words, in order to maximize $\mathcal{V}(\widehat{\pi}')$, we can instead minimize $\mathcal{V}(\pi') - \mathcal{V}(\widehat{\pi}')$ by (4.18). The representation is, again, still of a limited practical value since $d^{\widehat{\pi}',D}(s)$ and ε are usually unavailable to the agent. However, if ε is not large, then we can minimize the upper bound of the first term in the right-hand-side of Equation 4.18. A similar practical approximation is proposed as follows:

$$\begin{aligned} \widehat{\pi}^*_{\mathsf{CS-cRL}} &= \arg\min_{\widehat{\pi}'} \sum_{s\in\mathcal{S}} G^\pi(s, \widehat{\pi}') \\ &\approx \arg\min_{\widehat{\pi}'} \{\mathcal{V}(\pi') - \mathcal{V}(\widehat{\pi}')\} \\ &= \arg\max_{\widehat{\pi}'} \mathcal{V}(\widehat{\pi}'). \end{aligned} \tag{4.19}$$

Conveniently, (4.19) is exactly the cost-sensitive classification problem where the misclassification costs are conditional on individual cases [Turney, 2000]. In the RL settings, s is the attribute, $a^*_\pi(s)$ is the desired class label, and $G^\pi(s)$ is the misclassification cost. Based on the analysis above, an algorithm is proposed that focuses learning on more important states (Figure 4.2). Since costs are introduced, the algorithm is called CS-cRL (Cost-Sensitive classification-based Reinforcement Learning) as the learning procedure CS-Learn is cost-sensitive.

The following two theorems describe a key difference between CI-cRL and CS-cRL. Theorem 6 states that if the cost-sensitive classifier in CS-cRL has a sufficiently high quality, and if ε (the $\mathcal{L}_\infty$-norm difference between the two state visitation distributions in Equation 4.17) is not large, then the approximate greedy policy $\widehat{\pi}'$ will be close to π' in terms of policy value. In contrast, Theorem 7 establishes that, even if $\varepsilon = 0$, we can always construct an MDP so that $V(\pi') - V(\widehat{\pi}')$ is arbitrarily close to its upper bound as long as the cost-insensitive classifier in CI-cRL has a non-zero classification error.

Theorem 6 *If $\widehat{\pi}'_{\mathsf{CS-cRL}}$ has a sufficiently high quality, i.e.,*

$$\exists \epsilon > 0, \ \frac{\sum_{s\in\mathcal{S}} G^\pi(s, \widehat{\pi}'_{\mathsf{CS-cRL}})}{\sum_{s\in\mathcal{S}} G^\pi(s)} < \epsilon, \tag{4.20}$$

and assume the notation in (4.17), then

$$V(\pi') - V(\widehat{\pi}'_{\mathsf{CS-cRL}}) \ \leq \ (\frac{\epsilon}{1-\gamma} + \varepsilon) \sum_{s\in\mathcal{S}} G^\pi(s). \tag{4.21}$$

```
CS-cRL
INPUT:
    $M$:  generative model
    $\mathcal{T}$:  training states
    $\gamma$:  discount factor
    $K$:  number of trajectories
    $H$:  maximum length of trajectories
OUTPUT: $\pi \approx \pi^*$
$\widehat{\pi}' \leftarrow$ random policy
repeat
    $\pi \leftarrow \widehat{\pi}'$
    $T \leftarrow \emptyset$
    for each $s \in \mathcal{T}$
        for each $a \in \mathcal{A}$
            $\widehat{Q}^\pi(s,a) \leftarrow \mathsf{Rollout}(M, s, a, \gamma, \pi, K, H)$
        $\hat{a}^*_\pi \leftarrow \arg\max_{a \in \mathcal{A}} \widehat{Q}^\pi(s,a)$
        if $\widehat{Q}^\pi(s,\hat{a}^*_\pi) > \widehat{Q}^\pi(s,a), \forall a \neq \hat{a}^*_\pi$
            $g \leftarrow \widehat{Q}^\pi(s,\hat{a}^*_\pi) - \widehat{Q}^\pi(s,a)$
            $T \leftarrow T \cup \{\langle s, \hat{a}^*_\pi, g\rangle\}$
    $\widehat{\pi}' \leftarrow$ CS-Learn $(T)$
until $\pi \approx \widehat{\pi}'$
return $\pi$
```

Figure 4.2: CS-cRL: Cost-Sensitive classification-based RL. CS-Learn is a sub-routine that induces a cost-sensitive classifier from the input training data.

Theorem 7 *Let π' be the greedy policy of π and*

$$\epsilon = \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} \mathcal{I}\big(\widehat{\pi}'(s) \neq \pi'(s)\big) \tag{4.22}$$

be the classification error of $\widehat{\pi}'$, then $\forall \epsilon, \xi > 0$, there exists an MDP, π, π' and $\widehat{\pi}'$, s.t.

$$\mathcal{V}(\pi') - \mathcal{V}(\widehat{\pi}') > \frac{1-\xi}{1-\gamma} \sum_{s \in \mathcal{S}} G^\pi(s). \tag{4.23}$$

4.4 Empirical Evaluation

In this section, we report the empirical results in a 2D grid-world domain to evaluate the advantages of importance-sensitive classification-based reinforcement learning. This domain is adopted from Dietterich & Wang [Dietterich and Wang, 2002], which can be thought of as a simplified version of some discrete problems such as SAT [Hopcroft *et al.*, 2000] being solved by the Davis-Putnam-Logemann-Loveland (DPLL) procedure [Davis *et al.*, 1962].

4.4.1 Experimental Domain

We consider grid-worlds of N by N cells with no walls. Each state is represented as a tuple, (x, y) where $x, y \in \{1, 2, \cdots, N\}$. Start states are randomly chosen in the leftmost column, $(1, y)$. At every step, the agent has two possible actions, north-east and south-east. The two actions deterministically take the agent from the position (x, y) to $(x+1, y+1)$ or $(x+1, y-1)$, respectively. If the agent attempts to step out of the grid-world (i.e., $y-1$ or $y+1$ exceeds the range $[1, N]$), it will continue to move along the boundary (i.e., go towards east). An episode is terminated when the agent reaches the rightmost column.

4.4.2 Experiment I: Batch Reinforcement Learning

In the first experiment, $N = 100$. In order to make the optimal policy have a positive value (which is more intuitive in our evaluation below), we assign a small positive reward of 2 to each action (note that this does not change any policy). Additionally, 3000 units of negative rewards with a value of -1 each are randomly positioned in the grid-world according to some distribution scheme. If more than one reward is placed in the same cell, then the rewards are accumulated. The distribution scheme used in our experiment was a mixture of a uniform distribution and a two-dimensional Gaussian distribution centered at the cell $(50, 50)$ with the variance $\sigma = 10$ in each dimension. The uniform and Gaussian distributions carried the weights of 0.4 and 0.6, respectively. Formally, for all $x, y \in \{1, 2, \cdots, 100\}$, the weight[2] of cell (x, y) is computed by:

$$d(x,y) \quad = \quad \frac{0.4}{100 \times 100} + \frac{0.6}{2\pi\sigma} \exp\left\{ \frac{(x-50)^2 + (y-50)^2}{2\sigma^2} \right\}. \tag{4.24}$$

Figure 4.3 illustrates an example of the reward distribution in the grid world. This problem is a non-discounted finite-horizon MDP. The goal of the agent is to learn a policy to maximize its cumulative rewards by minimizing the negative rewards throughout the grid-world.

In each run of the batch RL experiments, a *random* set of training states, $\mathcal{T} \subset \mathcal{S}$, was generated, then full-trajectory-tree expansion was applied to each training state in $\mathcal{T}$ to compute its optimal action values. Finally, these optimal action values were used to construct T_{CI} (2.13) and T_{CS} (4.7). Figure 4.4 illustrates an example of the state importance values in the grid world.

We compared two different algorithms. One algorithm, called CI, is the baseline algorithm that solves the importance-insensitive optimization problem (4.8); the other, called CS, solves the importance-sensitive classification problem (4.6). We used a feed-forward multi-layer artificial neural network (ANN) as the classifier. The topology and parameters

[2]I.e., the probability of getting a negative reward at state (x, y).

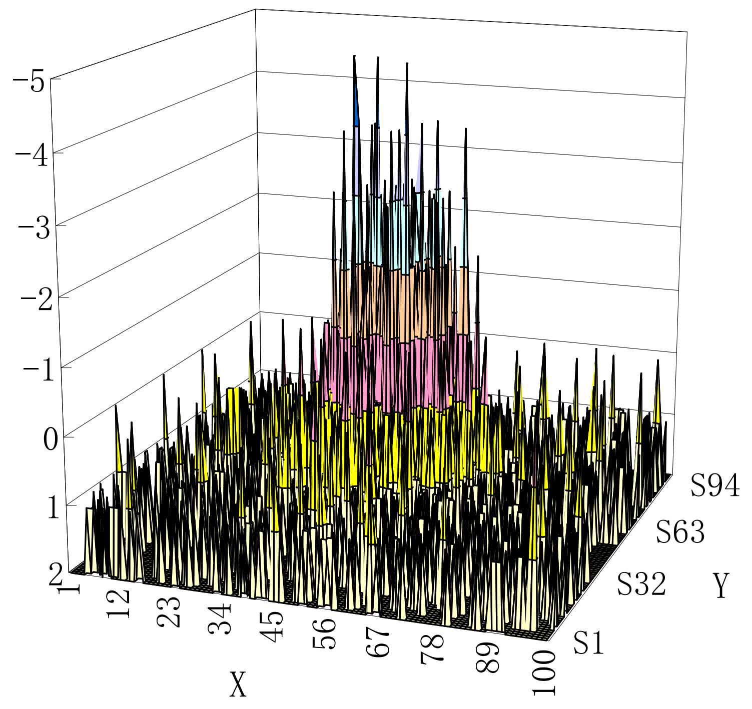


Figure 4.3: A typical distribution of immediate rewards in the two-dimensional grid-world used for our empirical evaluation. The rewards were randomly generated under a mixture distribution scheme consisting of a uniform distribution and a Gaussian distribution given in Equation 4.24.

of the ANN were fixed throughout the experiments, so that both classifiers, $\widehat{\pi}^*_{\mathsf{CI}}$ and $\widehat{\pi}^*_{\mathsf{CS}}$, have the same learning ability. Cost-sensitivity in classification can be achieved in different ways [Zadrozny and Langford, 2003]. We adopted simple resampling, where the training samples are drawn according to the following distribution:

$$\Pr(\text{select state } s \in \mathcal{T} \text{ for training}) = \frac{G^*(s)}{\sum_{s' \in \mathcal{T}} G^*(s')}.$$

Each of the algorithms was evaluated along the two performance measures:

- *Relative Policy Value* (RPV)[3], defined as:

$$\text{RPV}(\pi) = \frac{\mathcal{V}(\pi)}{\mathcal{V}(\pi^*)};$$

- *Average Misclassification Cost* (MCC) over the entire state space, defined as:

$$\text{MCC}(\pi) = \frac{\sum_s G^*(s,\pi)}{|\mathcal{S}|}.$$

[3]NB: since $V^*(\pi)$ is guaranteed to be positive, higher RPV indicates a better policy.

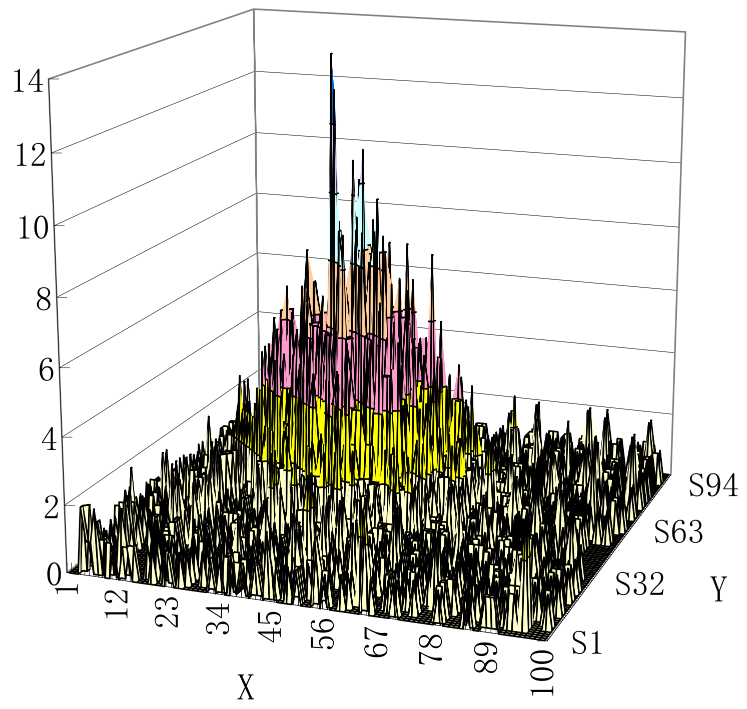
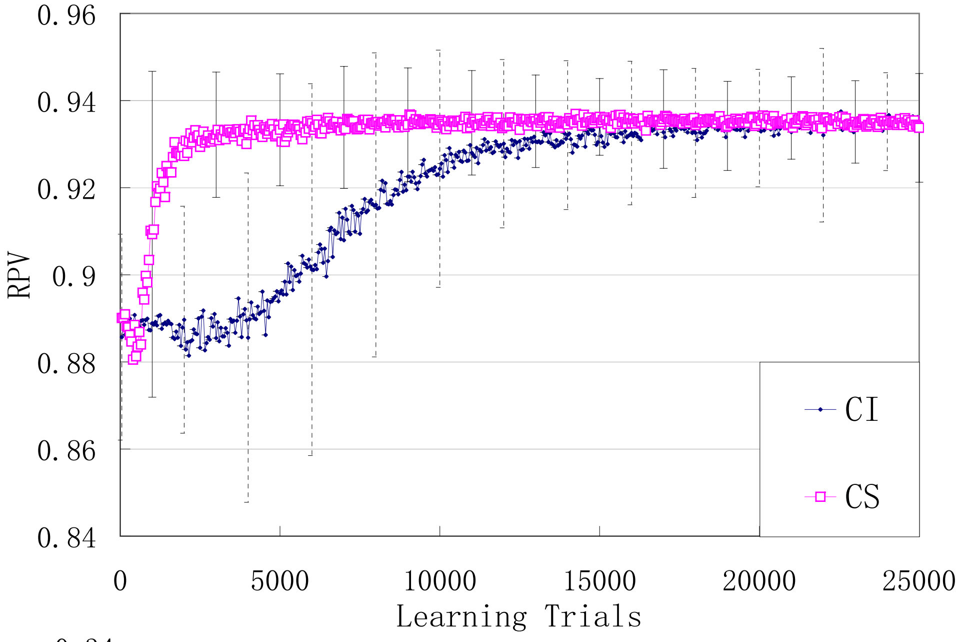


Figure 4.4: A typical distribution of importance values in the two-dimensional grid-world used for our empirical evaluation of focused learning.

The true objective of our reinforcement learning agent is to maximize RPV while the classifier's objective is to reduce MCC.

In the experiments, $\mathcal{T}$ contained 1000 states randomly sampled from the original state space: 700 of them were used for training, and the other 300 were used for validation to guard against overfitting. Ten random 100×100 grid-worlds were generated according to the reward distribution, and 20 learning sessions were conducted in each of them resulting in a total of 400 trials for each algorithm. The results are plotted in Figures 4.5 with standard deviation as the error bars. Several observations are in order.

First, note that the importance-sensitive algorithm CS increased the policy value substantially faster than the importance-insensitive CI. The significant advantage was observed early in the learning process. Note that the importance values of the grid-world states vary significantly, as shown in Figure 4.4. For many states, the importance is negligible or even zero; on the other hand, some of the states have much greater importance which makes them more significant in affecting the policy value and deserve more attention in learning. This observation leads to the conjecture that CS can learn even better when there is a high variance in the importance distribution over the state space.



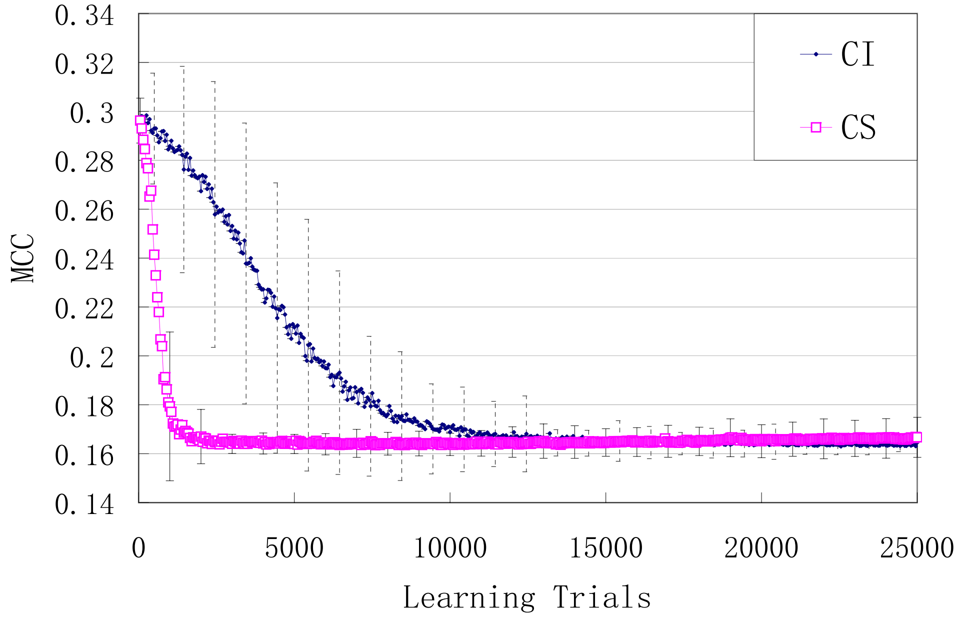


Figure 4.5: Policy value and misclassification cost in the grid-world experiment. Standard deviations are plotted every 1000 learning trials. In each trial, one training state is drawn from $\mathcal{T}$ for updating the weights in the artificial neural network.

Second, we note that the importance-sensitive learner CS was able to reduce the average misclassification cost (MCC) faster than CI [4]. The superior ability to reduce MCC appears to be the reason for a more rapid RPV improvement exhibited by CS. The results also suggest an efficiency of our approximation in (4.6).

Finally, we find that the standard deviation of the policies obtained by CS is lower (Figure 4.5(a)). This seems to be due to the fact that CS pays more attention to more important states. In turn, it is able to do better with the same amount of training time and data.

4.4.3 Experiment II: Online Reinforcement Learning

For the online learning experiment, we used support vector machines[5] as the classifiers. Smaller mazes[6] with a size of 50 by 50 were used. 2000 pieces of rewards each with a value of 1 were placed in the state space according to a similar reward distribution in the previous section. The center of the Gaussian distribution was moved to the cell $(25, 25)$ and the variance $\sigma = 5$. In summary, the reward distribution used was:

$$d(x, y) \quad = \quad \frac{0.4}{50 \times 50} + \frac{0.6}{2\pi\sigma} \exp\left\{ \frac{(x-25)^2 + (y-25)^2}{2\sigma^2} \right\}.$$

500 states were selected randomly from the entire state space as the training state set $\mathcal{T}$, which was then presented to CI-cRL and CS-cRL, respectively. Three metrics were used to evaluate the approximate greedy policy $\widehat{\pi}'$ in each iteration:

- *Policy Value* (PV) $\mathcal{V}(\widehat{\pi}')$;
- *Classification Error* (CE), defined as:

$$\text{CE}(\widehat{\pi}') = \frac{\sum_{s \in \mathcal{S}} \mathcal{I}(\widehat{\pi}' \neq \pi')}{|\mathcal{S}|};$$

- *Weighted Classification Costs* (WCE), the classification error weighted by the state importance $G^\pi(s)$, defined as:

$$\text{WCE}(\widehat{\pi}') = \frac{\sum_{s \in \mathcal{S}} G^\pi(s, \widehat{\pi}')}{\sum_{s \in \mathcal{S}} G^\pi(s)} = \frac{\sum_{s \in \mathcal{S}} G^\pi(s) \mathcal{I}(\widehat{\pi}' \neq \pi')}{\sum_{s \in \mathcal{S}} G^\pi(s)}.$$

In the original form, SVMs are cost insensitive and do not take the misclassification costs into account. We again used the simple resampling technique by making the number of occurrences of a training sample $\langle s, a, g \rangle$ in T proportional to its importance value g.

[4]Note that MCC is different from the standard uniform classification error insomuch as it is weighted by state importance (the misclassification cost).

[5]We used LS-SVMlab [Suykens *et al.*, 2002], a publicly available implementation of SVMs in the MATLAB environment.

[6]We used smaller mazes because running SVM implemented in MATLAB took a long time for a 100×100 maze.

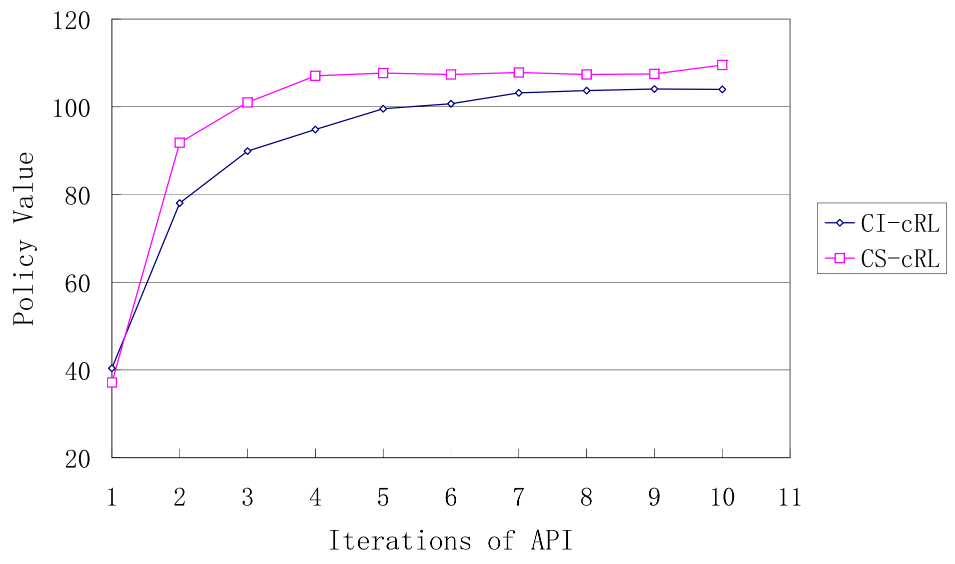


Figure 4.6: Policy values in the first ten policy iterations, averaged over 50 runs.

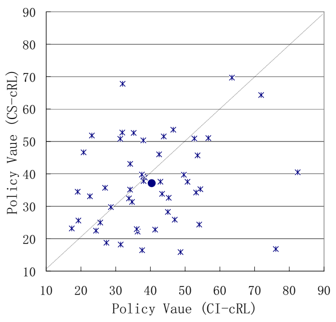
Iteration	1	2	3	4	5
Advantage	–	13.6±15.7	11.1±17.4	12.2±14.3	8.1±11.0
Prob(better)	–	81%	76%	86%	76%

Table 4.1: Statistics of the policy value improvements of CS-cRL obtained from 50 runs of experiment.

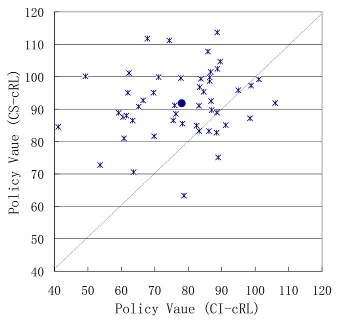
In the experiment, a maze was randomly generated and 50 runs of experiments were conducted. The average policy value of CI-cRL and CS-cRL in the first 10 API iterations are shown in Figures 4.6.

First, we observed that CS-cRL successfully increased the policy values by 10–20% in the second to fifth iterations[7], compared with CI-cRL. Although the advantage of improving policy values was not observed for each run of the experiments, CS-cRL did produce better policies in most cases. Statistics for the second to the fifth iterations are given in Table 4.1, including the average policy value improvement of CS-cRL (with variances) defined as $\mathcal{V}(\widehat{\pi}'_{\mathsf{CS-cRL}}) - \mathcal{V}(\widehat{\pi}'_{\mathsf{CI-cRL}})$, as well as the probability that CS-cRL did better than CI-cRL during the 50 runs. Figure 4.7 gives the complete comparison of policy values between these two algorithms. Each point in each graph corresponds to one of the 50 runs, and the large solid circle corresponds to the average performance. It shows that, on average, CS-cRL consistently works better than CI-cRL.

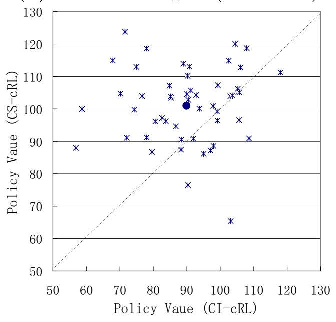
[7]Note that the performance of the first iteration will not be compared because the policies were random policies.



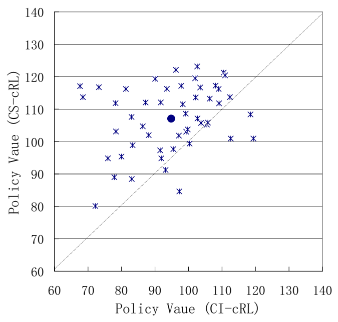
(a) Iteration #1 (random)



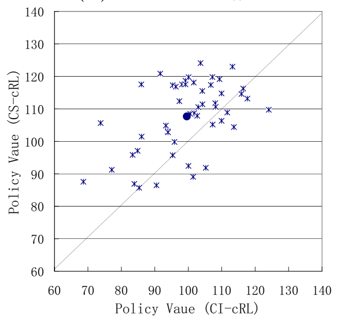
(b) Iteration #2



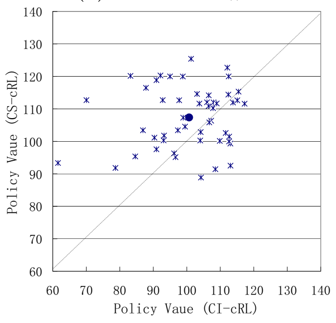
(c) Iteration #3



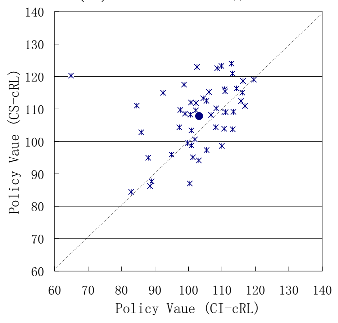
(d) Iteration #4



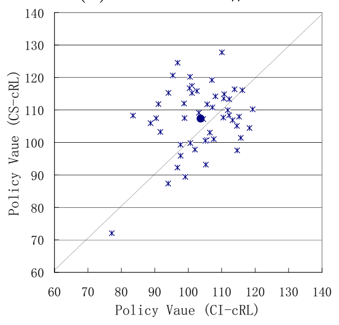
(e) Iteration #5



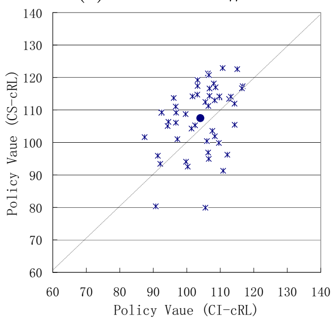
(f) Iteration #6



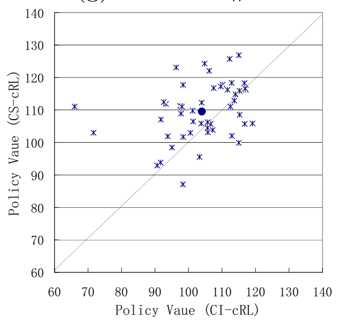
(g) Iteration #7



(h) Iteration #8

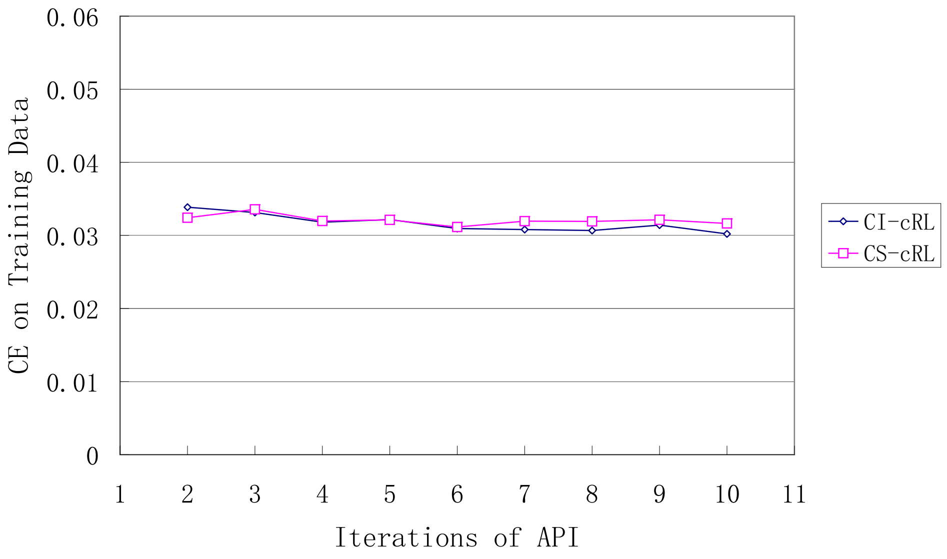


(i) Iteration #9

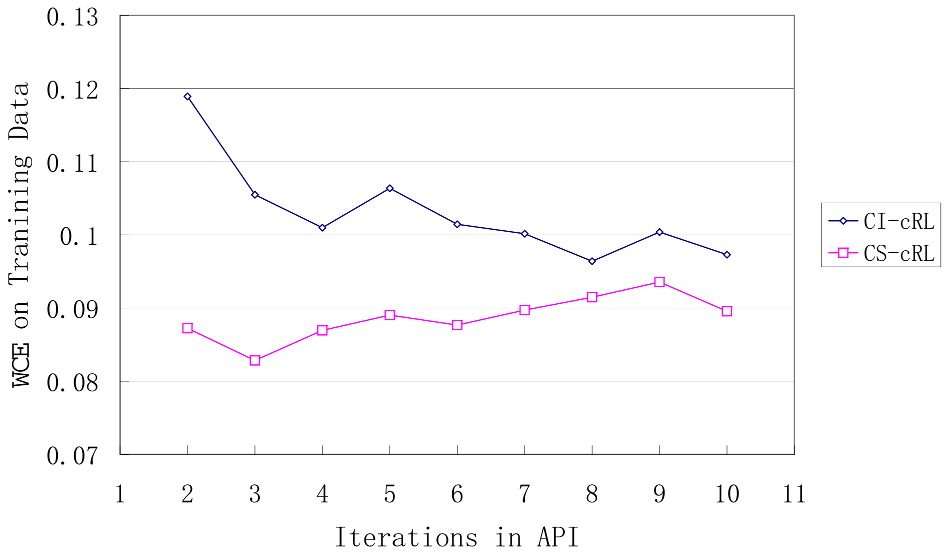


(j) Iteration #10

Figure 4.7: Policy values of the 50 runs. Average performances correspond to the solid circles in the figures.

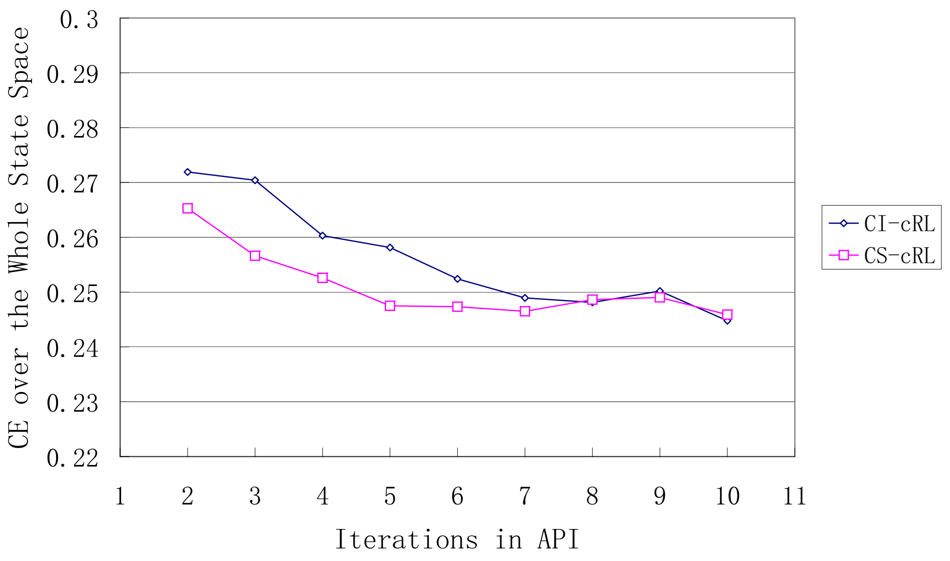


(a) CE

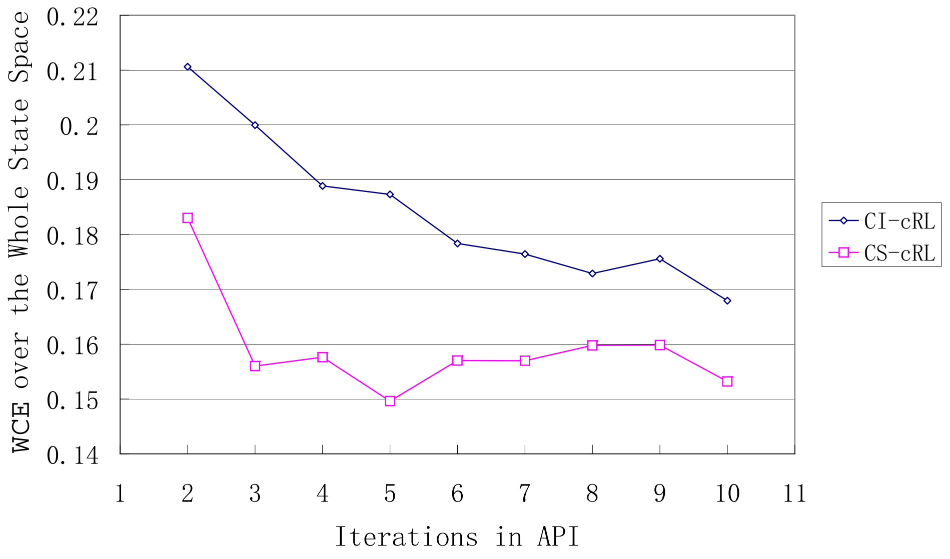


(b) WCE

Figure 4.8: (Weighted) classification errors on the training set, averaged over 50 runs.



(a) CE



(b) WCE

Figure 4.9: (Weighted) classification errors over the entire state space, averaged over 50 runs.

Next, we will examine the relations between CE/WCE and PV in the experiments to see whether it is helpful to focus on more important states. Figures 4.8 and 4.9 give the results (CE/WCE) on the training set and over the whole state space, respectively. It is clear from the results that CS-cRL managed to decrease the WCE while it has a performance comparable to CI-cRL in terms of reducing CE. These results together with Figure 4.6 suggest that the proposed method is able to focus on more important states, which appeared to be the reason why important-sensitive learning is helpful.

4.5 Proofs

This section provides detailed proofs of the Theorems 1–7 in the previous sections.

Proof of Theorem 1

Suppose the agent follows the policy τ from any start state $s_0 \sim D$. The trajectory is denoted by $s_0, a_0, r_1, s_1, a_1, \cdots, s_t, a_t, r_{t+1}, s_{t+1}, \cdots$. Then:

$$\begin{aligned} V^{\upsilon}(s_0) &= Q^{\upsilon}(s_0, \upsilon(s_0)) \\ &= Q^{\upsilon}(s_0, \tau(s_0)) - G(s_0, \upsilon \to \tau) \\ &= \mathop{\mathbf{E}}_{\tau}\{r_1 + \gamma V^{\upsilon}(s_1) - G(s_0, \upsilon \to \tau)\} \end{aligned} \tag{4.25}$$

Note that equation (4.25) is recurrent. In a similar fashion we derive:

$$\begin{aligned} & V^{\upsilon}(s_0) \\ = \ & \mathop{\mathbf{E}}_{\tau}\{r_1 + \gamma V^{\upsilon}(s_1) - G(s_0, \upsilon \to \tau)\} \\ = \ & \mathop{\mathbf{E}}_{\tau}\left\{r_1 + \gamma r_2 + \gamma^2 V^{\upsilon}(s_2) - G(s_0, \upsilon \to \tau) - \gamma G(s_1, \upsilon \to \tau)\right\} \\ = \ & \mathop{\mathbf{E}}_{\tau}\left\{r_1 + \gamma r_2 + \gamma^2 r_3 + \gamma^3 V^{\upsilon}(s_3) - G(s_0, \upsilon \to \tau) - \gamma G(s_1, \upsilon \to \tau) - \gamma^2 G(s_2, \upsilon \to \tau)\right\} \\ = \ & \cdots \\ = \ & \mathop{\mathbf{E}}_{\tau} \sum_{t=0}^{\infty} \gamma^t r_t - \mathop{\mathbf{E}}_{\tau}\left\{\sum_{t=0}^{\infty} \gamma^t G(s_t, \upsilon \to \tau)\right\} \\ = \ & V^{\tau}(s_0) - \mathop{\mathbf{E}}_{\tau}\left\{\sum_{t=0}^{\infty} \gamma^t G(s_t, \upsilon \to \tau)\right\}. \end{aligned}$$

Therefore,

$$V^{\tau}(s_0) - V^{\upsilon}(s_0) = \mathop{\mathbf{E}}_{\tau}\left\{\sum_{t=0}^{\infty} \gamma^t G(s_t, \upsilon \to \tau)\right\}. \tag{4.26}$$

Taking the expectation of equation (4.26) over all start states s_0 according to the distribution D, then:

$$\begin{aligned}
V(\tau) - V(\upsilon) &= \mathop{\mathbf{E}}_{s_0 \sim D} \{V^\tau(s_0) - V^\upsilon(s_0)\} \\
&= \mathop{\mathbf{E}}_{s_0 \sim D} \left\{ \mathop{\mathbf{E}}_{\tau} \left\{ \sum_{t=0}^{\infty} \gamma^t G(s_t, \upsilon \to \tau) \right\} \right\} \\
&= \sum_{t=0}^{\infty} \gamma^t \mathop{\mathbf{E}}_{s_0 \sim D, \tau} \{G(s_t, \upsilon \to \tau)\} \\
&= \sum_{t=0}^{\infty} \gamma^t \sum_{s \in \mathcal{S}} \Big(G(s, \upsilon \to \tau) \mu_t^{\tau, D}(s) \Big) \\
&= \sum_{t=0}^{\infty} \sum_{s \in \mathcal{S}} \Big(G(s, \upsilon \to \tau) \cdot \gamma^t \, \mu_t^{\tau, D}(s) \Big) \\
&= \sum_{s \in \mathcal{S}} \left(G(s, \upsilon \to \tau) \cdot \sum_{t=0}^{\infty} \gamma^t \, \mu_t^{\tau, D}(s) \right).
\end{aligned}$$

$\mathcal{Q.E.D}$.

Proof of Theorem 2

In Theorem 1, let $\tau = \pi$ and $\upsilon = \pi^*$, then

$$G^\upsilon(s, \upsilon \to \tau) = Q^*(s, \pi(s)) - Q^*(s, \pi^*(s)) = -G^*(s, \pi).$$

Then the policy loss $L(\pi)$ can be computed by Theorem 1,

$$\begin{aligned}
L(\pi) &= V(\pi^*) - V(\pi) \\
&= -\big(V(\tau) - V(\upsilon)\big) \\
&= -\sum_{s \in \mathcal{S}} \left(G^\upsilon(s, \upsilon \to \tau) \cdot \sum_{t=0}^{\infty} \gamma^t \mu_t^{\tau, D}(s) \right) \\
&= -\sum_{s \in \mathcal{S}} \left(-G^*(s, \pi) \cdot \sum_{t=0}^{\infty} \gamma^t \mu_t^{\tau, D}(s) \right) \\
&= \sum_{s \in \mathcal{S}} \left(G^*(s, \pi) \cdot \sum_{t=0}^{\infty} \gamma^t \mu_t^{\tau, D}(s) \right).
\end{aligned}$$

$\mathcal{Q.E.D}$.

Proof of Theorem 3

It always holds that $\mu_t^{\pi,D}(s) \in [0,1]$, $\forall \pi, D, s, t$. Therefore, by Theorem 2,

$$\begin{aligned} L(\pi) &= \sum_{s\in\mathcal{S}} \left(G^*(s,\pi) \cdot \sum_{t=0}^{\infty} \gamma^t \, \mu_t^{\pi,D}(s) \right) \\ &\leq \sum_{s\in\mathcal{S}} \left(G^*(s,\pi) \cdot \sum_{t=0}^{\infty} \gamma^t \right) \\ &= \frac{1}{1-\gamma} \sum_{s\in\mathcal{S}} G^*(s,\pi). \end{aligned}$$

On the other hand, $\widehat{\pi}^*_{\mathsf{CS}}$ satisfies Equation 4.10, therefore,

$$\begin{aligned} L(\widehat{\pi}^*_{\mathsf{CS}}) &\leq \frac{1}{1-\gamma} \sum_{s\in\mathcal{S}} G^*(s,\widehat{\pi}^*_{\mathsf{CS}}) \\ &= \frac{1}{1-\gamma} \frac{\sum_{s\in\mathcal{S}} G^*(s,\widehat{\pi}^*_{\mathsf{CS}})}{\sum_{s\in\mathcal{S}} G^*(s)} \sum_{s\in\mathcal{S}} G^*(s) \\ &< \frac{\epsilon}{1-\gamma} \sum_{s\in\mathcal{S}} G^*(s). \end{aligned}$$

$\mathcal{Q.E.D.}$

Proof of Theorem 4

We are going to prove this theorem by giving an MDP and a policy that has an arbitrary low classification error (bounded by ϵ), but has an arbitrarily poor policy value. For the MDP described below, we show that its parameters, p and R^*, can always be tuned to make (4.12) true even when (4.11) is guaranteed.

The MDP has N states ($\{s^1, s^2, \cdots, s^N\}$) and two actions for each state: one is the optimal action a^*, the other is the suboptimal action $\bar{a}$. Taking a^* in state s^1 lead to a positive reward R^* while the reward is r^* in all other states; taking $\bar{a}$ always results in a zero reward. The next state distribution is independent from the current state and actions taken: there is a probability p to go to s^1 and a probability $(1-p)/(N-1)$ to go to any other state. The start state distribution is:

$$D(s) = \begin{cases} p, & s = s^1 \\ \frac{1-p}{N-1}, & s \neq s^1 \end{cases}.$$

We let $R^* \gg r^* > 0$ and p being close to 1. Intuitively, s^1 is much more important than any other state from the sequential-decision-making point of view. Note that this is an ergodic, infinite-horizon MDP.

Assume there is a policy $\widehat{\pi}^*$ that selects the optimal actions for all states except for state s^1. That is,

$$\widehat{\pi}^*(s) = \begin{cases} a^*, & s \neq s^1 \\ \bar{a}, & s = s^1 \end{cases}. \qquad (4.27)$$

It is easy to verify that,

$$\begin{aligned} G^*(s) &= \begin{cases} r^*, & s \neq s^1 \\ R^*, & s = s^1 \end{cases}, \\ G^*(s,\pi) &= \begin{cases} 0, & s \neq s^1 \\ R^*, & s = s^1 \end{cases}. \end{aligned}$$

It can be computed that for all π, D, t:

$$\mu_t^{\pi,D}(s) = \begin{cases} p, & s = s^1 \\ \frac{1-p}{N-1}, & s \neq s^1 \end{cases},$$

and by Theorem 2, the policy loss of $\widehat{\pi}^*$ is:

$$L(\widehat{\pi}^*) = \sum_{s\in\mathcal{S}} \left(G^*(s,\pi) \sum_{t=0}^{\infty} \gamma^t \mu_t^{\pi,D}(s) \right) = \frac{pR^*}{1-\gamma} + \frac{(1-p)r^*}{1-\gamma}. \tag{4.28}$$

Note that the classification error of $\widehat{\pi}^*$ is $1/N$. By letting $1/N \leq \epsilon$, or equivalently, $N \geq \lceil \frac{1}{\epsilon} \rceil$, the policy has a classification error of at most ϵ. But for any positive real $\xi \in (0,1)$, let

$$L(\widehat{\pi}^*) \geq \frac{1-\xi}{1-\gamma} \sum_{s\in\mathcal{S}} G^*(s) = \frac{1-\xi}{1-\gamma} \big(R^* + (N-1)r^* \big).$$

By considering (4.28), we have

$$(p-1+\xi)R^* \quad \geq \quad \big((1-\xi)(N-1) - (1-p)\big) r^*.$$

Therefore, as long as $p-1+\xi > 0$, or $p > 1-\xi$, the appropriate range of R^* can be solved:

$$R^* \quad \geq \quad \frac{\big((1-\xi)(N-1) - (1-p)\big)}{p-1+\xi} r^*.$$

In summary, for any $\epsilon, \xi \in (0,1]$, we have solved the ranges for p, R^*, so that even if (4.11) is guaranteed, (4.12) can still be true.
$\mathcal{Q}.\mathcal{E}.\mathcal{D}.$

Proof of Theorem 5

In Theorem 1, let $\nu = \pi$. Then letting τ be π' and $\widehat{\pi}'$ yields the following equations, respectively:

$$\begin{aligned} \mathcal{V}(\pi') - \mathcal{V}(\pi) &= \sum_{s\in\mathcal{S}} \Big(G^{\pi}(s, \pi \to \pi') \cdot d^{\pi',D}(s) \Big), \\ \mathcal{V}(\widehat{\pi}') - \mathcal{V}(\pi) &= \sum_{s\in\mathcal{S}} \Big(G^{\pi}(s, \pi \to \widehat{\pi}') \cdot d^{\widehat{\pi}',D}(s) \Big). \end{aligned}$$

By subtracting these two equations and using Equation 4.17 we have:

$$\begin{aligned}
& \mathcal{V}(\pi') - \mathcal{V}(\widehat{\pi}') \\
= \ & \sum_{s\in\mathcal{S}} \Big(G^{\pi}(s, \pi \to \pi') \cdot d^{\pi',D}(s) - G^{\pi}(s, \pi \to \widehat{\pi}') \cdot d^{\widehat{\pi}',D}(s) \Big) \\
= \ & \sum_{s\in\mathcal{S}} g(s),
\end{aligned}$$

where,

$$g(s) = G^{\pi}(s, \pi \to \pi') \cdot d^{\pi',D}(s) - G^{\pi}(s, \pi \to \widehat{\pi}') \cdot d^{\widehat{\pi}',D}(s).$$

If $\pi(s) = a^*_{\pi}(s)$, then

$$\begin{aligned}
G^{\pi}(s, \pi \to \pi') &= 0, \\
G^{\pi}(s, \pi \to \widehat{\pi}') &= -G^{\pi}(s) \cdot \mathcal{I}(\widehat{\pi}'(s) \neq \pi'(s)).
\end{aligned}$$

Therefore,

$$\begin{aligned}
g(s) &= d^{\widehat{\pi}',D}(s) \cdot G^{\pi}(s) \cdot \mathcal{I}(\widehat{\pi}'(s) \neq \pi'(s)) \\
&= d^{\widehat{\pi}',D}(s) \cdot G^{\pi}(s, \widehat{\pi}').
\end{aligned}$$

If $\pi(s) \neq a^*_{\pi}(s)$, then

$$\begin{aligned}
G^{\pi}(s, \pi \to \pi') &= G^{\pi}(s), \\
G^{\pi}(s, \pi \to \widehat{\pi}') &= G^{\pi}(s) \cdot \mathcal{I}(\widehat{\pi}'(s) = \pi'(s)) \\
&= G^{\pi}(s) \cdot \big(1 - \mathcal{I}(\widehat{\pi}'(s) \neq \pi'(s))\big).
\end{aligned}$$

Therefore,

$$\begin{aligned}
g(s) &= G^{\pi}(s)\big(d^{\pi',D}(s) - d^{\widehat{\pi}',D}(s) + d^{\widehat{\pi}',D}(s) \cdot \mathcal{I}(\widehat{\pi}'(s) \neq \pi'(s))\big) \\
&= G^{\pi}(s)\big(d^{\pi',D}(s) - d^{\widehat{\pi}',D}(s)\big) + G^{\pi}(s) \cdot d^{\widehat{\pi}',D}(s) \cdot \mathcal{I}(\widehat{\pi}'(s) \neq \pi'(s)) \\
&\leq \varepsilon \cdot G^{\pi}(s) + G^{\pi}(s) \cdot d^{\widehat{\pi}',D}(s) \cdot \mathcal{I}(\widehat{\pi}'(s) \neq \pi'(s)) \\
&= \varepsilon \cdot G^{\pi}(s) + G^{\pi}(s, \widehat{\pi}') \cdot d^{\widehat{\pi}',D}(s)
\end{aligned}$$

In either case,

$$g(s) \leq \varepsilon \cdot G^{\pi}(s) + G^{\pi}(s, \widehat{\pi}') \cdot d^{\widehat{\pi}',D}(s).$$

Consequently,

$$\begin{aligned}
\mathcal{V}(\pi') - \mathcal{V}(\widehat{\pi}') &= \sum_{s} g(s) \\
&\leq \sum_{s} \Big(\varepsilon \cdot G^{\pi}(s) + G^{\pi}(s, \widehat{\pi}') \cdot d^{\widehat{\pi}',D}(s) \Big) \\
&= \sum_{s} left(G^{\pi}(s, \widehat{\pi}') \cdot d^{\widehat{\pi}',D}(s) + \varepsilon \cdot \sum_{s} G^{\pi}(s).
\end{aligned}$$

$\mathcal{Q.E.D}$.

Proof of Theorem 6

It always holds that $d^{\widehat{\pi}',D}(s) \in [0,1]$. Therefore, if Equation 4.20 is true, then by Theorem 5,

$$\begin{aligned} \mathcal{V}(\pi') - \mathcal{V}(\widehat{\pi}') &\leq \sum_{s \in \mathcal{S}} \left(G^{\pi}(s, \widehat{\pi}') \cdot \sum_{t=0}^{\infty} \gamma^t \right) + \varepsilon \cdot \sum_{s} G^{\pi}(s) \\ &= \frac{1}{1-\gamma} \sum_{s \in \mathcal{S}} G^{\pi}(s, \widehat{\pi}') + \varepsilon \cdot \sum_{s} G^{\pi}(s). \end{aligned}$$

On the other hand, $\widehat{\pi}'_{\mathsf{CS-cRL}}$ satisfies Equation 4.20, hence,

$$\begin{aligned} \mathcal{V}(\pi') - \mathcal{V}(\widehat{\pi}'_{\mathsf{CS-cRL}}) &\leq \frac{1}{1-\gamma} \frac{\sum_{s \in \mathcal{S}} G^{\pi}(s, \widehat{\pi}'_{\mathsf{CS-cRL}})}{\sum_{s \in \mathcal{S}} G^{\pi}(s)} \sum_{s \in \mathcal{S}} G^{\pi}(s) + \varepsilon \cdot \sum_{s} G^{\pi}(s) \\ &< \frac{\epsilon}{1-\gamma} \sum_{s \in \mathcal{S}} G^{\pi}(s) + \varepsilon \cdot \sum_{s} G^{\pi}(s) \\ &= (\frac{\epsilon}{1-\gamma} + \varepsilon) \sum_{s \in \mathcal{S}} G^{\pi}(s). \end{aligned}$$

$\mathcal{Q.E.D}$.

Proof of Theorem 7

This existence proof is the same as the proof of Theorem 4. The same example is used, and let $\pi = \pi' = \pi^*$ and $\widehat{\pi}' = \widehat{\pi}^*$ defined in Equation 4.27. Then following the same calculations in the proof, we can always find N, p, r^*, and R^* such that $\widehat{\pi}'$ has an ϵ-classification error but $\mathcal{V}(\pi') - \mathcal{V}(\widehat{\pi}')$ can be arbitrarily close to its upper bound, $\sum_{s \in \mathcal{S}} G^{\pi}(s)$.
$\mathcal{Q.E.D}$.

Chapter 5

An Extension to the Value-Function Methods

This chapter extends the idea of focused learning to the value-function methods to prevent the policy degradation problem (cf. Section 2.3.3). Two problem settings (batch and online) will be considered. More focus will be on the online learning problem, where the MDPs are ergodic and the policies satisfy Assumption 3.

5.1 Focusing Attention: Batch Reinforcement Learning

We start with the batch learning problem. Again, it is assumed that the training data for batch learning are provided in the form of Equation 2.12 which is repeated below for the reader's convenience:

$$T_{Q^*} = \{\langle s, a, Q^*(s,a)\rangle \mid s \in \mathcal{T}, a \in \mathcal{A}\}, \tag{5.1}$$

or

$$T_{V^*} = \{\langle s, V^*(s)\rangle \mid s \in \mathcal{T}\}. \tag{5.2}$$

A natural approach would be to train a regressor using standard supervised learning techniques to minimize the MSE, i.e., to solve the following optimization problem:

$$\theta^*_{\text{SL}} = \arg\min_\theta \sum_{s \in \mathcal{S}} \big(Q^*(s,a) - Q(s,a,\theta)\big)^2. \tag{5.3}$$

5.1.1 Inconsistency between Policy Values and Value Function Approximation Accuracy

In this section, we will present empirical evidence for the inconsistency between policy values and value function approximation accuracy in the batch reinforcement learning setting, by using the same 2D grid-world domain described in Section 4.4.1. We used 100

Feature Set	Features
#0	coarse tiling
#1	polynomial: $(x, y, x^2, y^2, \cdots, x^k, y^k)$
#2	$(x, y, d, d^2, \cdots, d^k)$ where $d = \sqrt{(x-50)^2 + (y-50)^2}$
#3	$(x, y, d_x, d_y, d_x^2, d_y^2, \cdots, d_x^k, d_y^k)$ where $d_x = \lvert x - 50\rvert$ and $d_y = \lvert y - 50\rvert$

Table 5.1: Features used in the grid-world domain. A state s is represented by its coordinates, (x, y).

by 100 grid-world with 3000 unit rewards randomly placed according to the mixture distribution of Equation 4.24. In the experiment, the training state set $\mathcal{T}$ consisted of 400 states drawn randomly whose optimal action values are known. The RL agent uses a linear combination of features as the value function approximation, and least-mean-square (LMS, cf. Appendix A.4.1) algorithm was used for training. For simplicity in constructing features, we used training data in the form of Equation 5.2 and assumed the agent has a world model including the reward function and transition matrices. The features are simple nonlinear functions defined on the 2D coordinates (x, y) (Table 5.1).

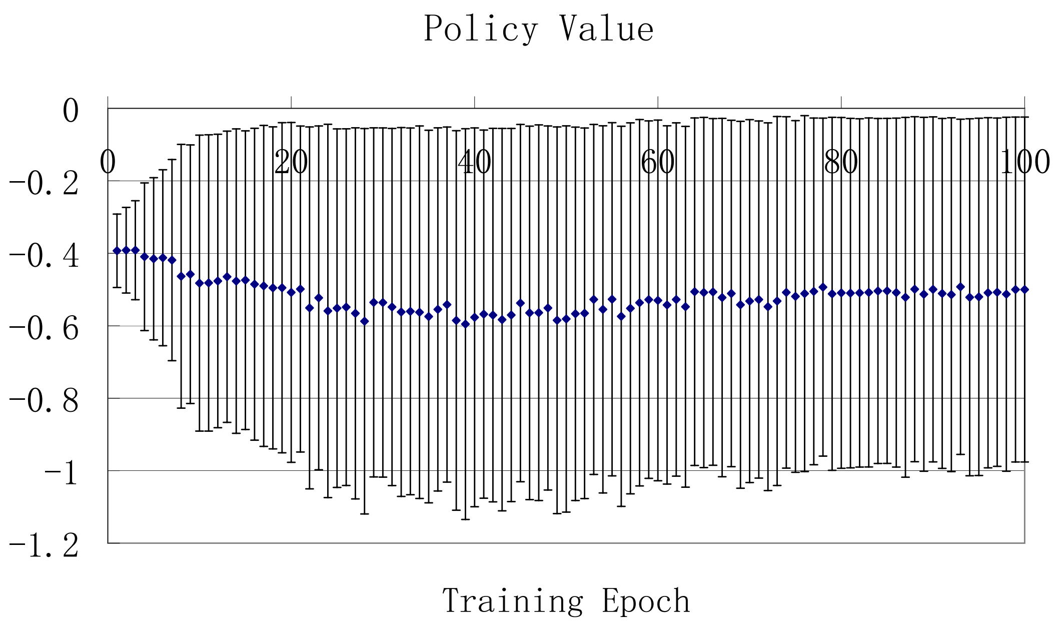
We ran the experiments 20 times solving the optimization problem in Equation 5.3. In each run of the experiments, two performance metrics were recorded: the policy value and the value function approximation accuracy (root mean squared error). Figure 5.1 gives the average performance with variance plotted. It shows that, as the training goes on, the policy quality degrades: the policies have smaller average value and greater variance (Figure 5.1(a)). On the other hand, the value function approximation was getting more and more accurate. This example demonstrates the inconsistency between the two metrics.

5.1.2 Focusing Attention on More Important States.

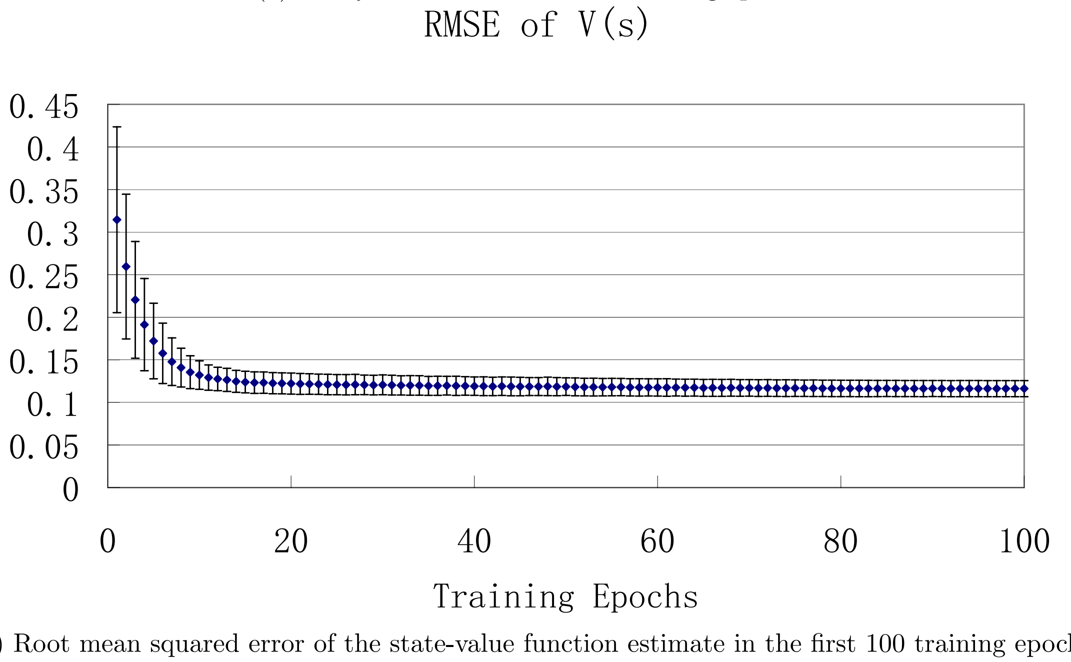
In this section, we used the measure of state importance defined in Equation 4.2. We compared three learning methods. VF0 is the baseline LMS without employing $G^*(s_i)$ in the training data; VF1 updates the weight vector only using states s with non-zero importance $G^*(s)$; VF2 re-samples the training states s_i in T according to the distribution:

$$\Pr(\text{select state } s_i \text{ for training}) \quad = \quad \frac{G^*(s_i)}{\sum_{j=1}^{m} G^*(s_j)}.$$

A large number of experiment settings were tried, including the size of input features (determined by k in Table 5.1), the learning rate, the number of training data. As shown by the results, VF1 and VF2 worked no worse than VF0 in terms of policy values most of the time. Figure 5.2 shows an example where VF1 and VF2 are better. VF0 decreased the



(a) Policy value in the first 100 training epochs



(b) Root mean squared error of the state-value function estimate in the first 100 training epochs

Figure 5.1: Approximating the optimal value function more precisely can lead to policies with lower quality. The values were averaged over 20 runs of experiments with variances plotted.

RMSE successfully, but did not work well in improving the resulting policy. In contrast, although the value function approximate produced by VF1 and VF2 had a larger RMSE, the resulting policies had higher values.

Even in the case where VF0 managed to improved the policy value, VF1 and VF2 can sometimes produce a policy with a higher value although their root mean squared errors of function approximation were larger (Figure 5.3).

5.1.3 Penalty for Making Suboptimal Actions

Another possible way to improve the naive approach in Equation 5.3 is to use a penalty to encourage the agent to make the right decision in training states, and when the value function approximation is generalized, the resulting policy will be more likely to agree with the optimal policy in other unseen states. In fact, this idea has been employed by Dietterich & Wang [Dietterich and Wang, 2002], where the optimal value function approximation is computed by solving three different optimization problems with the kernel tricks [Schölkopf and Smola, 2001]. Details are found in [Dietterich and Wang, 2002] and will not be discussed here.

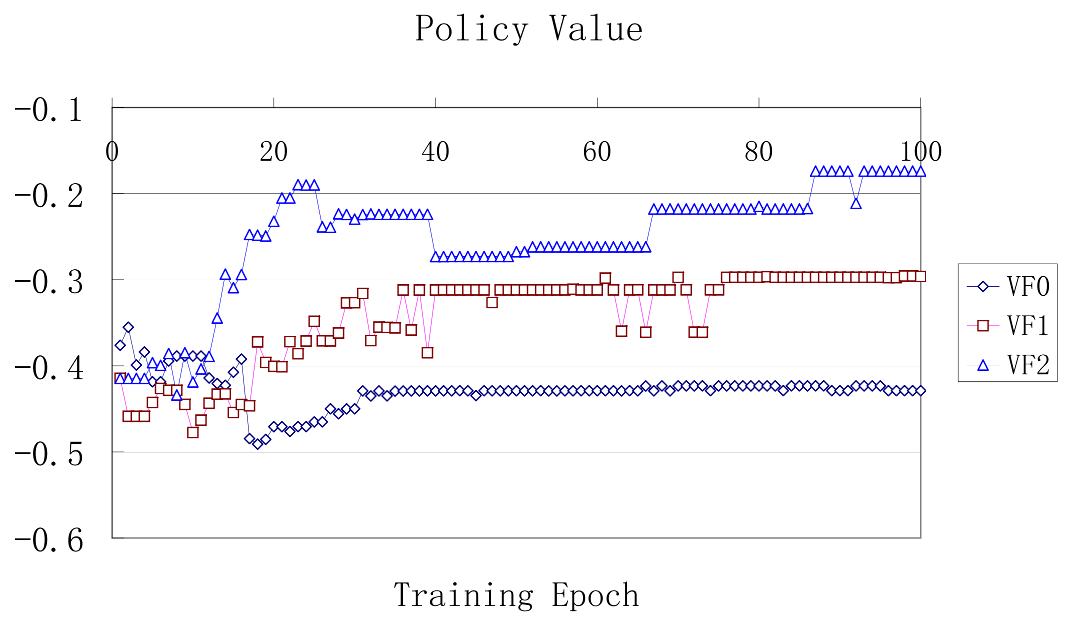
5.2 Focusing Attention: Online Reinforcement Learning

In considering online value-function methods, we assume the MDP has an infinite horizon and is ergodic. The policy is denoted by $\pi(s, a, \theta)$ where $\theta \in \mathbb{R}^k$ is the k-dimensional parameter vector. The distribution $\mu(s)$ in evaluating $\mathcal{V}(\pi)$ is fixed to be the on-policy distribution $\mu_\infty^\pi(s)$. To simplify the notation, $\pi(s, a, \theta)$ is replaced with θ when there is no ambiguity.

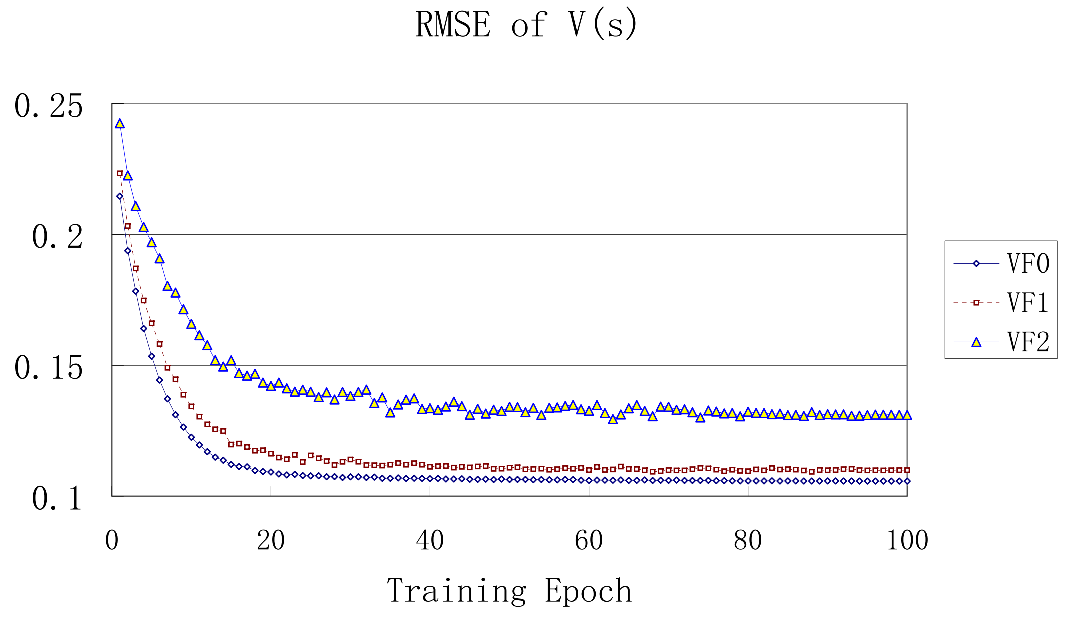
Note that the policy considered here always satisfies Assumption 3. In addition, we expect a policy to select the greedy action most frequently while having an exploration behavior. An often used form of such policies is the Gibbs softmax policy (Definition 8):

$$\pi(s, a) = \frac{e^{Q(s,a)/\tau}}{e^{\sum_b Q(s,b)/\tau}}, \tag{5.4}$$

where τ is the *temperature* parameter controlling the exploitation/exploration tradeoff: when $\tau \to \infty$, $\pi(s, a)$ tends to be a random policy; when $\tau \to 0$, $\pi(s, a)$ tends to be a pure greedy policy.

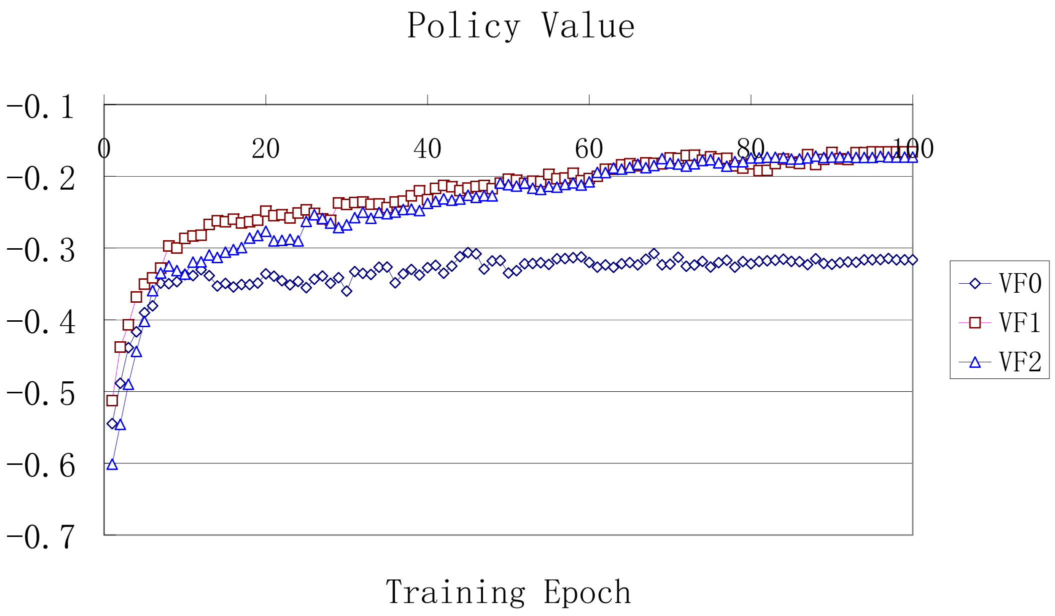


(a) Policy value of the three approaches

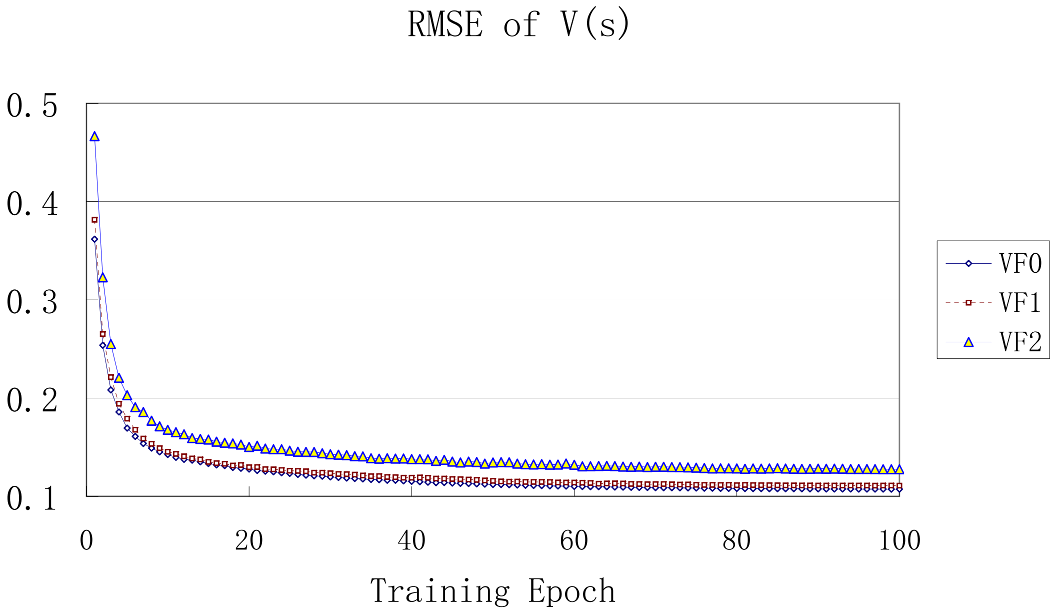


(b) Root mean squared error of the state-value function estimate

Figure 5.2: Comparison of the three batch value function learning methods. VF0 can decrease RMSE consistently but the policy value may not increase correspondingly; in contrast, VF1 and VF2 can compute a better policy although the value function is less accurate.



(a) Policy value of the three approaches



(b) Root mean squared error of the state-value function estimate

Figure 5.3: Comparison of the three batch value function learning methods. VF1 and VF2 can produce better policies although the value functions are less accurate than the one computed by VF0.

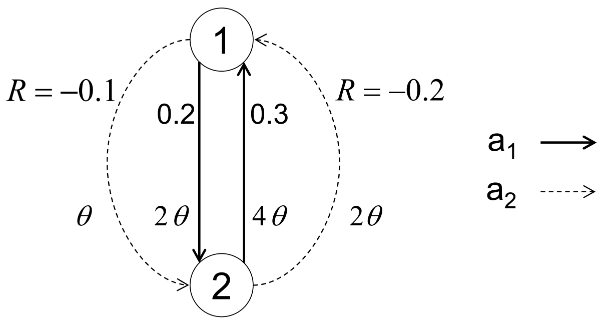


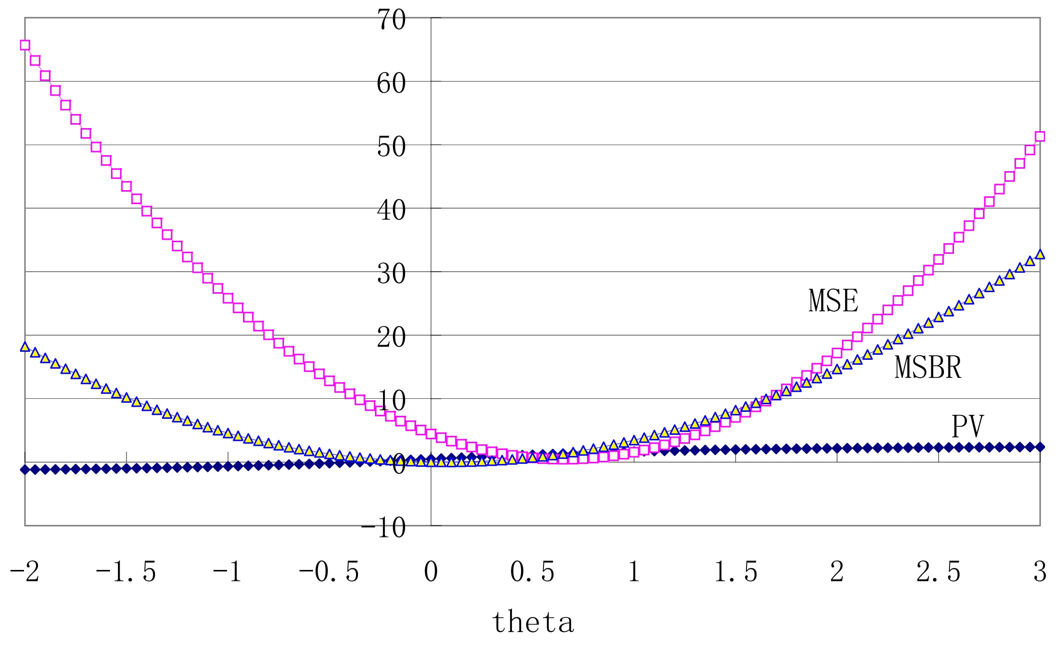
Figure 5.4: A simple 2-state MDP.

5.2.1 A Small Example

We will start with a small example that solves the policy evaluation problem in a toy MDP using TD(λ) with linear function approximation. The MDP in Figure 5.4 has two states (1 and 2), two actions (a_1 and a_2), and the discount factor $\gamma = 0.9$. As shown in the figure, the approximate action-value function is linear and the only parameter to adjust is a scalar θ. For a fixed θ, the policy $\pi(\theta)$ is the Gibbs softmax policy defined in Section 2.2.1. The agent will be taken to state 2 deterministically by taking either action in state 1, and vice versa. Actions are labeled with the immediate rewards.

Suppose the agent starts with one of the two states with the same probability, and is following the policy π which always selects a_1 with probability 0.9 and a_2 with probability 0.1, regardless of the current state. It is easy to verify that $\mathcal{V}(\pi) = 2.1$. Different policy evaluation algorithms are available: TD(1)/MC minimizes the mean squared error (MSE) of $Q(s, a, \theta)$; and residual gradient minimizes the mean squared Bellman residual (MSBR). Figure 5.5 plots the different performance metrics for different values of θ, where PV stands for the policy value. The two measures, MSE and MSBR, are minimized by different values of θ. Therefore, different algorithms are likely to end up with different solutions according to their own criteria. However, the minima found by TD(1) and residual gradient have policy values much lower than $\mathcal{V}(\pi)$. Numerical computation shows that the resulting policy value is 2.1 when $\theta \approx 1.75$. In contrast, the policy values of the TD(1) and residual gradient solutions are about 1.3 and 0.1, respectively.

The large gap in policy values can be a problem in approximate policy iteration, leading to the policy degradation problem, meaning that the greedy policy $\widehat{\pi}'$ w.r.t. $\widehat{Q}^{\pi}$ has a lower value than π. However, it is anticipated that $\widehat{\pi}'$ is no worse than π. Therefore, it is necessary to consider the policy value to avoid this undesired situation.



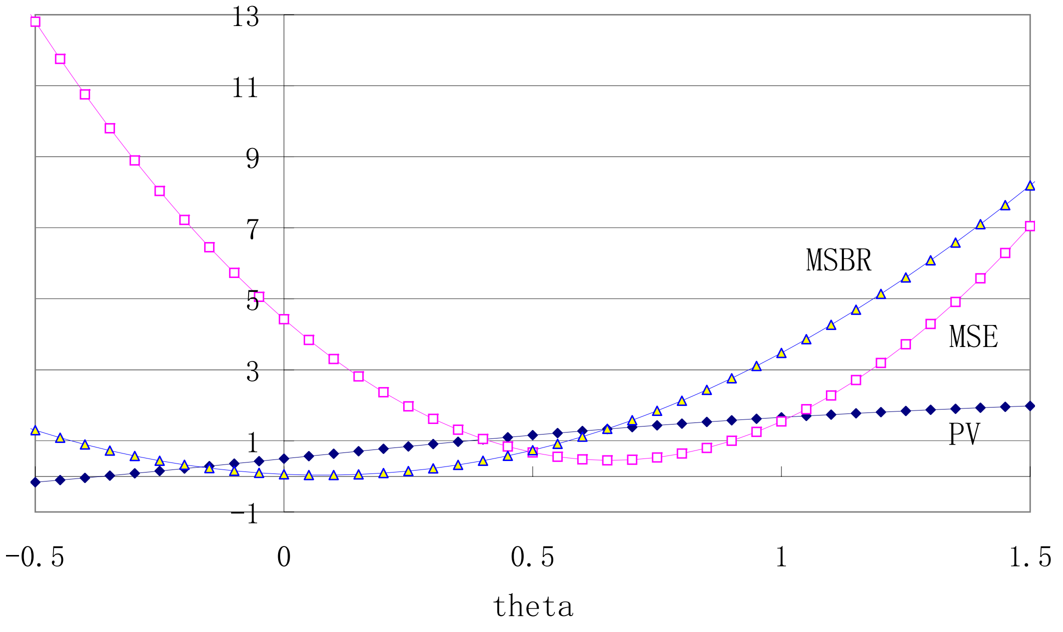


Figure 5.5: The three different performances of different values of θ for the policy prediction problem on the 2-state MDP (Figure 5.4). The bottom graph shows an enlarged fragment of the top graph.

5.2.2 Gradient of Policy Value in Changing Value Function Parameters

The policy degradation problem happens because the existing methods such as TD(λ), Sarsa, residual algorithms, do not take the policy value into account. In this section, we will study how policy value is affected by parameter updates in the value function $V(\theta)$ as $\theta \to \theta + \Delta\theta$. We denote the policy derived from $Q(s,a,\theta)$ by $\pi(s,a,\theta)$ and let the start-state distribution $D(s)$ be $\mu^{\pi}_{\infty}(s)$, then according to the Policy Switching Theorem (Section 4.1):

$$\mathcal{V}(\theta) - \mathcal{V}(\theta + \Delta\theta) = \sum_s \left(d^{\theta}(s) \cdot G^{\theta+\Delta\theta}(s, \theta + \Delta\theta \to \theta) \right).$$

Since

$$\begin{aligned} & G^{\theta+\Delta\theta}(s, \theta + \Delta\theta \to \theta) \\ = & \sum_a \left(\pi(s,a,\theta) \cdot Q^{\theta+\Delta\theta}(s,a) \right) - \sum_a \left(\pi(s,a,\theta+\Delta\theta) \cdot Q^{\theta+\Delta\theta}(s,a) \right) \\ = & \sum_a \left(Q^{\theta+\Delta\theta}(s,a) \cdot \left(\pi(s,a,\theta) - \pi(s,a,\theta+\Delta\theta) \right) \right), \end{aligned}$$

$$\mathcal{V}(\theta) - \mathcal{V}(\theta + \Delta\theta) = -\sum_s \left(d^{\theta}(s) \sum_a \left(Q^{\theta+\Delta\theta}(s,a) \cdot \left(\pi(s,a,\theta+\Delta\theta) - \pi(s,a,\theta) \right) \right) \right).$$

Therefore,

$$\begin{aligned} \nabla\mathcal{V} &= \lim_{\Delta\theta \to 0} \frac{\mathcal{V}(\theta + \Delta\theta) - \mathcal{V}(\theta)}{\Delta\theta} \\ &= \lim_{\Delta\theta \to 0} \sum_s \left(d^{\theta}(s) \sum_a \left(Q^{\theta+\Delta\theta}(s,a) \frac{\pi(s,a,\theta+\Delta\theta) - \pi(s,a,\theta)}{\Delta\theta} \right) \right) \\ &= \sum_s \left(d^{\theta}(s) \sum_a \left(Q^{\theta}(s,a) \cdot \nabla\pi(s,a,\theta) \right) \right). \end{aligned}$$

The last equation parallels the results in [Sutton *et al.*, 2000] and [Konda and Tsitsiklis, 2000], which have already been introduced in Section 2.3.4. This relation is important in that it expresses how each state contributes to the global policy gradient. In other words,

$$\nabla\mathcal{V} = \sum_s G^{\theta}(s),$$

where

$$G^{\theta}(s) = d^{\theta}(s) \sum_a \left(Q^{\theta}(s,a) \cdot \nabla\pi(s,a,\theta) \right)$$

can be viewed as a measure of generalized state importance in learning a value function.

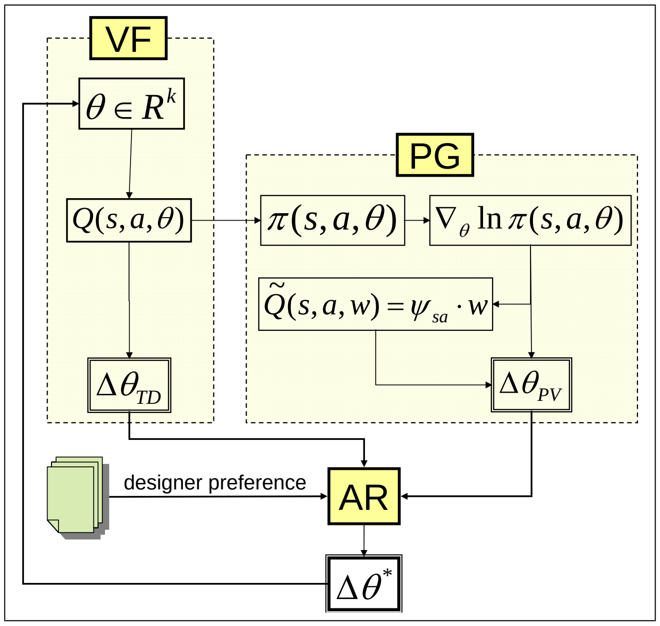


Figure 5.6: An architecture combining policy gradient and pure value function learning methods, which is flexible enough to allow a tradeoff between the policy value and value function approximation accuracy.

5.2.3 A Combination of Policy-Gradient and Value-Function Approach

In the previous section, we have computed the policy value gradient using the Policy Switching Theorem and obtained the same result as in the literature. Therefore, in updating the parameter θ, this gradient can be taken into account to prevent policy degradation. Here, we will describe a policy-gradient-based value-function method, henceforth called PGVF. It consists of three parts: (i) the value function learner (VF); (ii) the policy gradient estimator (PG); and (iii) the arbitrator (AR). Below is a description of these three parts. A concrete example instantiating this architecture is presented in the next section.

The Value Function Learner

The task of the value function learner is to approximate the value function accurately in terms of minimizing the MSE of the value function or the mean squared Bellman residual. Similar to other TD methods such as SARSA and Q-learning, the value function is parameterized by a k-dimensional vector $\theta \in \mathbb{R}^k$. The learner VF can apply any TD methods to update the parameter. We denote this update by $\Delta\theta_{\mathrm{TD}}$.

The Policy Gradient Estimator

The task of the policy gradient estimator is to estimate the direction of policy gradient. For any θ, a stochastic policy $\pi(s,a,\theta)$ is derived from the value function approximate $Q(s,a,\theta)$. We require that the policy satisfies Assumption 3. Then the features $\psi(s,a,\theta) = \nabla \ln \pi(s,a,\theta)$ can be extracted. As explained in Section 2.3.4, PG only needs to estimate the projection $\widehat{Q}(s,a,w)$ on a k-dimensional subspace Ψ^{θ}. Again, TD can be applied in computing the parameter w. The output of PG is an estimate of the policy gradient, denoted by $\Delta\theta_{\text{PV}}$.

The Arbitrator

The task of the arbitrator is to produce a final update of the parameter, $\Delta\theta^*$, by combing the output of VF ($\Delta\theta_{\text{TD}}$), the output of PG ($\Delta\theta_{\text{PV}}$), and designer-supplied preferences. This final update will be used to change the parameter θ.

5.2.4 An Instantiation of the PGVF Architecture

Figure 5.6 presents the abstract architecture of PGVF. In practice, the designer has the freedom in determining the representation of $Q(s,a,\theta)$ and the policy $\pi(s,a,\theta)$, as well as in incorporating preferences into the arbitrator. In this section, we propose an algorithm by instantiating the PGVF architecture as follows:

- The value function is a linear combination of state-action pair features;
- The stochastic policy is the Gibbs softmax distribution;

Selecting the Value Function Representation

In practice, linear function approximation is often used because it is simple enough to analyze and efficient to compute. In addition, it has been shown that TD learning with linear function approximation converges. In fact, linear function approximation can represent nonlinear functions by using nonlinear basis functions or employing the kernel tricks [Schölkopf and Smola, 2001]. Here, we will adopt this form of function approximation:

$$Q(s,a,\theta) = \varphi(s,a)\cdot\theta, \tag{5.5}$$

where $\varphi(s,a)$ is the k-dimensional feature vector extracted from the state-action pair (s,a).

Selecting the Stochastic Policy

In order to satisfy Assumption 3, we use the Gibbs softmax as the policy:

$$\pi(s,a,\theta) = \frac{e^{Q(s,a,\theta)/\tau}}{\sum_b e^{Q(s,b,\theta)/\tau}} = \frac{e^{\varphi(s,a)\cdot\theta/\tau}}{\sum_b e^{\varphi(s,b)\cdot\theta/\tau}}. \tag{5.6}$$

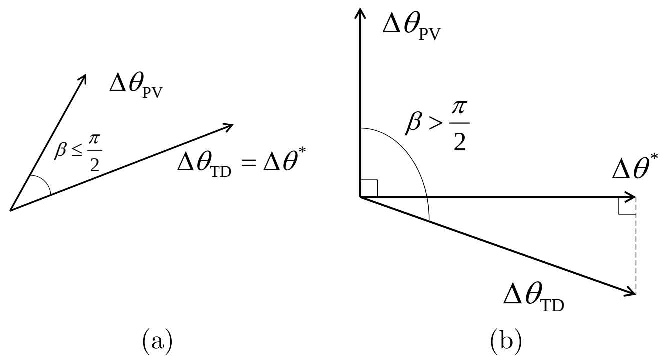


Figure 5.7: An illustration of the arbitrator function defined in Equation 5.9.

Then the features for the projection in Equation 2.17 can be computed by:

$$\psi(s,a,\theta) = \nabla \ln \pi(s,a,\theta) = \varphi(s,a) - \sum_b \pi(s,b)\varphi(s,b), \tag{5.7}$$

and the value function projection is

$$\widetilde{Q}(s,a) = \psi(s,a,\theta) \cdot w. \tag{5.8}$$

The policy gradient estimator is required to estimate w^* that satisfies Equation 2.17.

Selecting the Arbitrator

The final step is to select the arbitrator that combines the outputs of PG and VF according to some predefined human preferences and knowledge. If we want the value function to be as accurate as possible without degrading the policy value, then a natural way is to guarantee the angle between $\Delta\theta^*$ and $\nabla\mathcal{V}(\theta)$ is not blunt. Figure 5.7 illustrates the two cases of how $\Delta\theta^*$ can be computed. The corresponding arbitrator function is formally defined as:

$$\Delta\theta^* = \mathrm{Ar}_1(\Delta\theta_{\mathrm{TD}}, \Delta_{\mathrm{PV}}) = \begin{cases} \Delta\theta_{\mathrm{TD}}, & \Delta\theta_{\mathrm{TD}} \cdot \Delta_{\mathrm{PV}} \ge 0 \\ \Delta\theta_{\mathrm{TD}} - \frac{\Delta_{\mathrm{PV}} \cdot \Delta\theta_{\mathrm{TD}}}{|\Delta_{\mathrm{PV}}|^2} \Delta_{\mathrm{PV}}, & \Delta\theta_{\mathrm{TD}} \cdot \Delta_{\mathrm{PV}} < 0 \end{cases} . \tag{5.9}$$

Similarly, an alternative is to increase the policy value without degrading the accuracy of value functions:

$$\Delta\theta^* = \mathrm{Ar}_2(\Delta\theta_{\mathrm{TD}}, \Delta\theta_{\mathrm{PV}}) = \mathrm{Ar}_1(\Delta\theta_{\mathrm{PV}}, \Delta\theta_{\mathrm{TD}}). \tag{5.10}$$

The third way to compute $\Delta\theta^*$ is to use a scalar η to balance the tradeoff between value function accuracy and policy value:

$$\Delta\theta^* = \mathrm{Ar}_3(\Delta\theta_{\mathrm{TD}}, \Delta\theta_{\mathrm{PV}}) = \eta \cdot \Delta\theta_{\mathrm{TD}} + (1-\eta) \cdot \Delta\theta_{\mathrm{PV}}, \quad \eta \in [0,1]. \tag{5.11}$$

When $\eta = 0$, Ar_3 implements the policy gradient algorithm, as in [Sutton *et al.*, 2000]; when $\eta = 1$, it becomes the conventional TD methods.

It is worth mentioning a practical issue that will be used in our empirical study later. Suppose the agent computes $\Delta\theta_{\text{PV}}$ and adopts the first arbitrator function Ar_1. When the parameter θ reaches a local maximum of $\mathcal{V}(\theta)$, $\Delta\theta_{\text{PV}}$ is probably close to 0 rather than being exactly 0. In this case, the sign of $\Delta\theta_{\text{TD}} \cdot \Delta_{\text{PV}}$ is very noisy. Hence, a better way is to have a threshold $t_\theta > 0$, so that when $|\Delta\theta_{\text{PV}}| < t_\theta$, Ar_1 treats it as 0 and always outputs $\Delta\theta_{\text{TD}}$; when $|\Delta_{\text{PV}}| \geq t_\theta$, Ar_1 computes $\Delta\theta^*$ by Equation 5.9.

The Final Algorithm

More practical issues need to be solved before a final algorithm is available. Note that PG requires that θ is fixed when it estimates the gradient $\nabla\mathcal{V}$. If at each time step, $\Delta\theta^*$ is used to update the parameter θ using w for $\widetilde{Q}(s, a, w)$, then the policy $\pi(s, a, \theta)$ is changed, and therefore, the Ψ^θ is changed, resulting in w^* being changed. A similar problem is met in [Konda and Tsitsiklis, 2000] and [Sutton *et al.*, 2000]. The approach suggested by Konda & Tsitsiklis is to let the time scale for updating θ be slower than that for updating w [Konda and Borkar, 1999]. Formally, let α_t^θ and α_t^w be the step-size parameter on time step t for updating θ and w, respectively. If

$$\lim_{t\to\infty} \frac{\alpha_t^\theta}{\alpha_t^w} = 0,$$

then θ and w can be updated simultaneously.

Here we adopt a simpler approach more similar to that used by Sutton et al. [Sutton *et al.*, 2000]. We will use *batch update* for the parameters. In particular, θ will be fixed for a while, and the $\Delta\theta_{\text{TD}}$ computed at each time step will be accumulated but won't be used to update θ. When the value of w converges, it can be used to compute $\Delta\theta_{\text{PV}}$ by Equation 2.18. Finally, the true update $\Delta\theta^*$ is computed by a predefined arbitrator function. The algorithm is summarized in Figure 5.8.

5.3 Empirical Evaluations

We will use the same MDP in Figure 5.4. Two basic algorithms are studied: Sarsa(0) and Sarsa(1). The corresponding algorithms that implement the PGVF architecture are called PG-Sarsa(0) and PG-Sarsa(1), respectively. A threshold $t_\theta = 0.02$ was used.

Figure 5.9 compares Sarsa(0) and PG-Sarsa(0) with Ar_1. Without considering the policy value, the conventional Sarsa(0) converges to $\theta \approx 0.13$ which has a low policy value of around 0.7 (Figure 5.5). As expected, PG-Sarsa(0) managed to converge to $\theta \approx 0.95$

```
PGVF

INPUT:
    $\gamma$:                       discount factor
    $\alpha^w, \alpha^\theta$:      step sizes

OUTPUT: $Q(s, a, \theta) \approx Q^*(s, a)$

initialize $s \sim D$
$a \leftarrow \pi(s, a, \theta)$
$\theta \leftarrow \vec{0}$
repeat until $\theta$ converges
    $w \leftarrow \vec{0}$

    // Estimate $\widehat{Q}(s, a, w)$
    repeat
        Take action $a$ and observe $r$ and $s'$
        $a' \leftarrow \pi(s', a', \theta)$
        $\delta_{\text{TD}} \leftarrow r + \gamma \widetilde{Q}(s', a', w) - \widetilde{Q}(s, a, w)$
        $w \leftarrow w + \alpha^w \cdot \delta_{\text{TD}} \cdot \nabla_w \widetilde{Q}(s, a, w)$
        $s \leftarrow s'$
        $a \leftarrow a'$
    until $w$ converges

    // To estimate the policy gradient $\Delta\theta_{\text{PV}}$
    $\Delta\theta_{\text{PV}} \leftarrow \vec{0}$
    $\Delta\theta_{\text{TD}} \leftarrow \vec{0}$
    repeat
        Take action $a$ and observe $r$ and $s'$
        $a' \leftarrow \pi(s', a', \theta)$
        $\delta_{\text{TD}} \leftarrow r + \gamma Q(s', a', \theta) - Q(s, a, \theta)$
        $\Delta\theta_{\text{TD}} \leftarrow \Delta\theta_{\text{TD}} + \alpha^\theta \cdot \delta_{\text{TD}} \cdot \frac{\partial}{\partial\theta} Q(s, a, \theta)$
        $\Delta\theta_{\text{PV}} \leftarrow \Delta\theta_{\text{PV}} + \alpha^\theta \cdot \big(\widetilde{Q}(s, a, w) \cdot \psi(s, a, \theta) - \theta\big)$
        $s \leftarrow s'$
        $a \leftarrow a'$
    until $\Delta\theta_{\text{PV}}$ converges

    // To compute the final update $\Delta\theta^*$
    $\Delta\theta^* \leftarrow \text{Ar}(\Delta\theta_{\text{TD}}, \Delta\theta_{\text{PV}})$
    $\theta \leftarrow \theta + \Delta\theta^*$
end repeat
return $Q(s, a, \theta)$
```

Figure 5.8: An instance implementing the PGVF architecture.

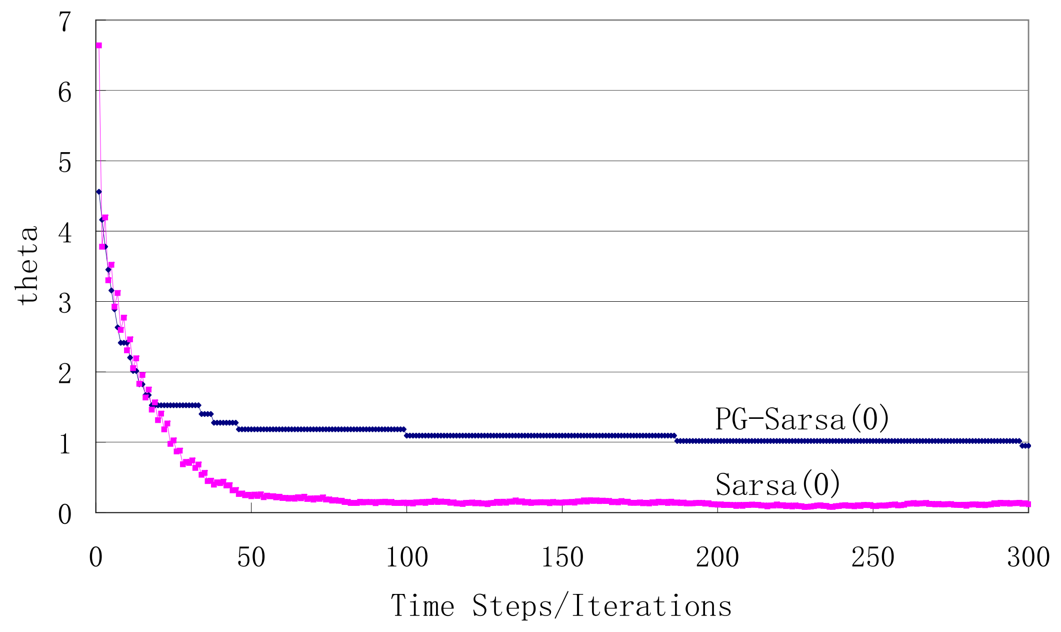


Figure 5.9: A comparison between SARSA(0) and PG-SARSA(0) using Ar_1 on the 2-state MDP.

with a policy value much closer to the optimal policy value ($\mathcal{V}(\pi^*) \approx 2.5$). As shown in Figure 5.5, $\nabla\mathcal{V} \approx 0$ when $\theta > 1.5$. In the first 20 iterations of Figure 5.9, therefore, PG-SARSA(0) behaves almost the same as SARSA(0) as $\Delta\theta_{\mathrm{PV}} < t_\theta$. When θ decreases, $\nabla\mathcal{V}$ increases. Consequently, PG-SARSA(0) eventually stops decreasing θ which converges to 0.95. The second arbitrator function Ar_2 is very similar to Ar_1 and leads to similar results.

Next, we investigate the third arbitrator function Ar_3 and study how η controls the tradeoff. Figure 5.10 show how θ is updated by PG-SARSA(0) using different values of η. The results is consistent with our analysis above: larger η indicates a smaller preference on the policy value. In particular, when $\eta = 1$, this is the conventional SARSA(0); when $\eta = 0$, the algorithm is equivalent to policy gradient, and since larger θ corresponds to greater policy value, PG-SARSA(0) with $\eta = 0$ tends to consistently increase θ. Figure 5.11 gives the converged values of θ given different η. Similar results were observed with PG-SARSA(1).

5.4 Further Discussion

In this section, we will give a comparison between PGVF and two closely related lines of work: policy gradient and VAPS [Baird, 1999]. On one hand, PGVF is able to consider the policy value by computing $\nabla\mathcal{V}$, which is similar to policy gradient. On the other hand, it explicitly handles the tradeoff between value function learning performance and policy

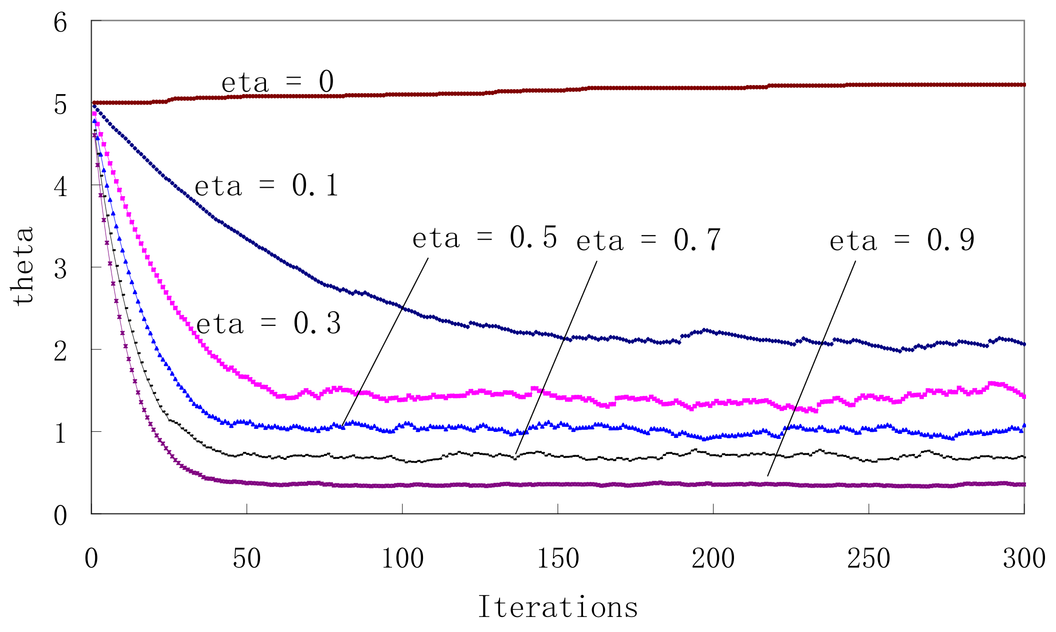


Figure 5.10: Different values of η controls the tradeoff between value function accuracy and the policy value in PG-Sarsa(0) using Ar$_3$ on the 2-state MDP.

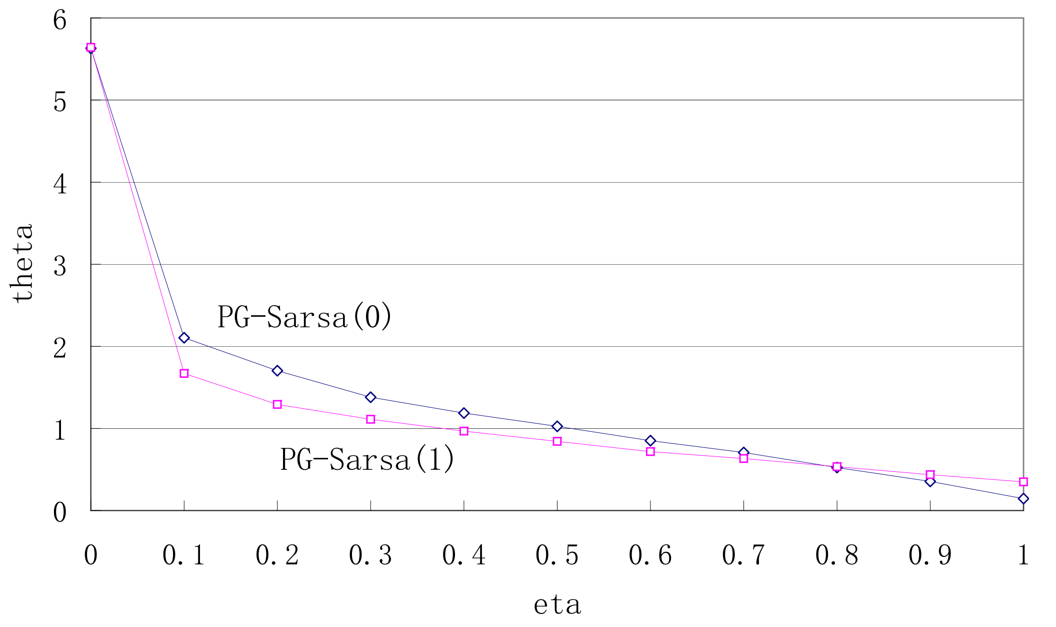


Figure 5.11: The converged values of θ with different η in PG-Sarsa(0) and PG-Sarsa(1) using Ar$_3$ on the 2-state MDP.

value, which is similar to VAPS in that it can perform value function learning and policy search simultaneously.

5.4.1 PGVF vs. Policy Gradient

Both PGVF and policy gradient can improve the policy value by computing the policy gradient $\nabla \mathcal{V}$. The only difference between them is that the parameter θ in policy gradient is the policy parameter, and the objective is to compute a locally optimal policy $\pi(s, a, \theta)$ directly. On the other hand, the θ in PGVF is used for the value function $Q(s, a, \theta)$, and the goal is to learn a value function from which a good policy is derived. There are reasons why PGVF may be better than the direct policy gradient method.

First, as discussed in Chapter 2, value functions are critical to reinforcement learning because they make use of the structure of the problem thereby making the learning process more efficient. In particular, value functions contain more information than policies alone. To illustrate this, consider the optimal policy π^* that selects a^* in state s. The only information extracted from π^* is that a^* is better than any other action in state s. Now consider a value function whose greedy action is also a^* in state s. In addition to the information that a^* is optimal, the function also predicts the quantity of how better it is, or its advantage [Baird, 1993] over other actions. This and possibly other kinds of information turn out to be very important in designing efficient algorithms as well as for multi-scale learning such as abstraction and hierarchical learning [Dietterich, 2000c; Kaelbling, 1993; Precup, 2000; Russell and Zimdars, 2003; Sutton *et al.*, 1999].

However, the policy gradient agent does maintain a critic $\widetilde{Q}(s, a, w)$, which can also be considered as an approximation to the true value function $Q^\theta(s, a)$. Can $\widetilde{Q}(s, a, w)$ be used to replace $Q(s, a, \theta)$? This leads to the second difference between PG and PGVF. Although both function approximations use a linear combination with k-dimensional parameters (θ and w, respectively), $Q(s, a, \theta)$ is expected to be more accurate than $\widetilde{Q}(s, a, w)$. Notice that the features $\varphi(s, a)$ used in $Q(s, a, \theta)$ are directly related to the state-action pairs; while the features $\psi(s, a) = \nabla \ln \pi$ used in $\widetilde{Q}(s, a, w)$ are less intuitive and depend on the policy parameter θ that is changed during learning. For this reason, it should be easier to incorporate expert knowledge into $\varphi(s, a)$ than into $\psi(s, a, \theta)$ thereby making $Q(s, a, \theta)$ more accurate than $\widetilde{Q}(s, a, w)$.

Third, PGVF explicitly considers two important, but potentially conflicting performance metrics: the policy value and the value function approximation performance (e.g., mean squared error, mean squared Bellman residual, or mean squared temporal difference). By doing this, PGVF provides a more flexible mechanism for using *a prior* preferences of the designers, and thus, has a more flexible tradeoff between these two metrics.

5.4.2 PGVF vs. VAPS

A similar idea of balancing the tradeoff between value-function learning and policy search is also implemented by VAPS (Value And Policy Search) [Baird and Moore, 1999]. VAPS starts with a definition of error $e(h_t)$ of a sequence h_t:

$$h_t = s_0, a_0, r_1, s_1, a_1, r_2, s_2, \cdots, s_{t-1}, a_{t-1}, r_t, s_t.$$

Then it performs gradient descent on the expected total error B during a period ended in time T where $T = 1, 2, 3, \cdots$. Under certain conditions, VAPS provably converges to a local minimum of B. By defining $e(h_t)$ in a similar way to Ar_3, VAPS is able to handle the tradeoff between value function learning and policy value.

PGVF is more flexible than VAPS in that it allows more complicated forms for handling the tradeoff. For example, the agent can make its value function more and more accurate without degrading the resulting policy simply by using Ar_1. In VAPS, however, if the agent tries not to degrade its policy, it has to put all weights on the policy performance without considering the value function accuracy. In such a case, VAPS degrades to pure policy-search methods without using a value function. In addition, other types of arbitrator functions other than Equations 5.9—5.11 can be used in PGVF to fit the designer's preferences.

Chapter 6

Conclusions and Future Work

6.1 Thesis Summary

This thesis investigated the problem of *improving the value of RL policies by focusing the learning process on more important states*. In particular, we examine two classes of RL algorithms: the policy-search methods based on classification, and the value-function methods. For each of them, we consider two settings of the RL problems: the batch and online learning.

- For the classification-based policy-search methods, we proposed a measure of the sequential decision-making importance of a state ($G^*(s)$ and $G^\pi(s)$ for the batch and online settings, respectively), which were shown to be closely related to the policy value. By focusing attention of learning in more important states, the agent can compute a better policy with less computation resources such as learning time. In addition to several theoretical results, the advantages of focusing attention are supported by empirical studies in a 2D grid-world domain.
- For the batch value-function methods, we discussed two ways of connecting the policy value to the error in function learning. One is to focus learning in more important states so as to make the function approximation more accurate at these states. Consequently, it is less likely that the agent will make suboptimal decisions in these states. Another idea is to add a penalty term to the value function errors, so that the agent is encouraged to agree with the optimal policy in the training states that it sees. In this way, when the value function is generalized to the whole state space, it is expected to result in a high-quality policy. The latter idea has also been studied by Dietterich and Wang [Dietterich and Wang, 2002].

- For online value-function methods, we proposed an architecture PGVF that combines the strengths of gradient descent and value function learning. This architecture consists of three parts. The PG element estimates the direction in the parameter space along which the policy value increases. The VF element implements the conventional value-function methods to minimize the MSE, MSBR, or temporal difference errors. Finally, the arbitrator AR decides on the final parameter updates based on the updates computed by PG and VF, as well as designer preferences and knowledge. We argue that such an architecture has the flexibility to handle the tradeoff between the possibly conflicting performance metrics that VF and PG attempt to optimize.

6.2 Future Work

The promising results open several avenues for future research. First, for classification-based focused learning methods, we can further prune down the training data set by placing a threshold on the importance level of a training state. A direct advantage is a reduction in training time. However, the extent to which such an *a priori* pruning may lead to overfitting needs to be explored.

Another area for future research is an investigation of the extent to which this approach depends on the cost-sensitive classifier used to represent a policy. In particular, it would be interesting to investigate the benefits of modern cost-sensitive classification methods (e.g., boosting [Fan *et al.*, 1999]) over the naive training data resampling we employed in the experiments.

Third, in designing approximations to the classification-based focused learning methods, we always approximated the original problem by minimizing the upper bound of the policy loss. However, if the state distribution is somehow known *a prior*, or can be estimated by an online agent, better approximations may be found.

Fourth, the PGVF architecture requires more time to compute the parameter updates than pure value function methods. Furthermore, each PGVF agent has to maintain two sets of parameters, θ and w, for the value functions $Q(s, a, \theta)$ and $\widetilde{Q}(s, a, w)$ respectively, which makes learning more complex. It would be helpful if the characteristics of the underlying MDP can be utilized to replace the PG in the architecture. In particular, we envision subclasses of MDPs that allows more efficient value function learning algorithms that are guaranteed not to degrade the policy.

Lastly, all the algorithms were only tested on small problems. Their demonstrated strengths need to be tested in complex, real-life applications. In addition, it is interesting to investigate under what conditions our proposed methods and focused learning are beneficial.

Bibliography

[Bagnell *et al.*, 2004] J Andrew Bagnell, Sham Kakade, Andrew Y. Ng, and Jeff Schneider. Policy search by dynamic programming. In *Advances in Neural Information Processing Systems (NIPS-03)*, volume 16, 2004.

[Baird and Moore, 1999] Leemon Baird and Andrew Moore. Gradient descent for general reinforcement learning. In David Cohn Michael Kearns, Sara Solla, editor, *Advances in Neural Information Processing Systems*, volume 11, pages 968–974. MIT Press, 1999.

[Baird, 1993] Leeman Baird. Advantage updating. Technical Report WL-TR-93-1146, Wright-Patterson Air Force Base, OH, 1993. Available from the Defense Technical information Center, Cameron Station, Alexandria, VA 22304-6145.

[Baird, 1995] Leemon Baird. Residual algorithms: Reinforcement learning with function approximation. In *the Twelfth International Conference on Machine Learning (ICML-95)*, pages 30–37, 1995.

[Baird, 1999] Leemon Baird. *Reinforcement Learning Through Gradient Descent.* PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, May 1999.

[Barto *et al.*, 1983] Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. Neuronlike elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 13:835–846, 1983. Reprinted in J. A. Anderson and E. Rosenfeld, Neurocomputing: Foundations of Research, MIT Press, Cambridge, MA, 1988.

[Baxter and Bartlett, 1999] Jonathan Baxter and Peter Bartlett. Direct gradient-based reinforcement learning: I. Gradient estimation algorithms. Technical report, Research School of Information Sciences and Engineering, Australian National University, July 1999.

[Baxter *et al.*, 1999] Jonathan Baxter, Lex Weaver, and Peter Bartlett. Direct gradient-based reinforcement learning: II. Gradient ascent algorithms and experiments. Technical report, Research School of Information Sciences and Engineering, Australian National University, September 1999.

[Bellman, 1957] Richard Bellman. *Dynamic Programming.* Princeton University Press, 1957.

[Berenji and Vengerov, 2003] Hamid R. Berenji and David Vengerov. A convergent actor critic based fuzzy reinforcement learning algorithm with application to power management of wireless transmitters. *IEEE Transactions on Fuzzy Systems*, 11(4):478–485, August 2003.

[Bertsekas and Tsitsiklis, 1996] Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming.* Athena Scientific, September 1996.

[Bertsekas, 1997] Dimitri P. Bertsekas. Differential training of rollout policies. In *Proceedings of the Thirty-Fifth Allerton Conference on Communication, Control, and Computing*, 1997.

[Boyan and Moore, 1995] Justin A. Boyan and Andrew W. Moore. Generalization in reinforcement learning: Safely approximating the value function. In *Advances in Neural Information Processing Systems (NIPS-94)*, volume 7, Cambridege, MA, 1995. MIT Press.

[Boyan *et al.*, 1995] Justin A. Boyan, Andrew W. Moore, and Richard S. Sutton, editors. *Proceedings of the Workshop on Value Function Approximation*, 1995. In the Twelfth International Conference on Machine Learning (ICML-95).

[Breiman *et al.*, 1984] Leo Breiman, Jerome Friedman, Richard Olshen, and Charles Stone. *Classification and Regression Trees.* Kluwer Academic Publishers, 1984.

[Breiman, 1996] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[Bulitko *et al.*, 2003] Vadim Bulitko, Lihong Li, Greg Lee, Russell Greiner, and Ilya Levner. Adaptive image interpretation : A spectrum of machine learning problems. In *ICML-03 Workshop on the Continuum from Labeled to Unlabeled Data in Machine Learning and Data Mining*, Washington D.C., August 2003.

[Cover and Hart, 1967] Thomas M. Cover and Peter E. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, IT-13(1):21–27, 1967.

[Crites and Barto, 1996] Robert H. Crites and Andrew G. Barto. Improving elevator performance using reinforcement learning. In *Advances in Neural Information Processing Systems (NIPS-95)*, volume 8, pages 1017–1023, 1996.

[Davis *et al.*, 1962] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem proving. *Communications of ACM*, pages 394–397, 1962.

[Dietterich and Wang, 2002] Thomas G. Dietterich and Xin Wang. Batch value function approximation via support vectors. In *Advances in Neural Information Processing Systems (NIPS-01)*, volume 14, 2002.

[Dietterich, 2000a] Thomas G. Dietterich. Ensemble methods in machine learning. *Lecture Notes in Computer Science*, 1857:1–15, 2000.

[Dietterich, 2000b] Thomas G. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning*, 40(2):139–3157, 2000.

[Dietterich, 2000c] Thomas G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.

[Draper and Baek, 1998] Bruce Draper and Kyungim Baek. Bagging in computer vision. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 144–149, Santa Barbara, CA, June 1998.

[Draper *et al.*, 2000] Bruce Draper, Jose Bins, and Kyungim Baek. ADORE: Adaptive object recognition. In *International Conference on Vision Systems*, Spain, 2000.

[Duffy and Helmbold, 2000] Nigel Duffy and David Helmbold. Potential boosters? In *Advances in Neural Information Processing Systems (NIPS-99)*, volume 12, pages 258–264. MIT Press, 2000.

[Duffy and Helmbold, 2002] Nigel Duffy and David Helmbold. Boosting methods for regression. *Machine Learning*, 47:153–200, 2002.

[Fan *et al.*, 1999] Wei Fan, Salvatore J. Stolfo, Junxin Zhang, and Philip K. Chan. AdaCost: Misclassification cost-sensitive boosting. In *Proceedings of the Sixteenth International Conference on Machine Learning (ICML-99)*, pages 97–105, Bled, Slovenia, 1999.

[Fern *et al.*, 2004] Alan Fern, SungWook Yoon, and Robert Givan. Approximate policy iteration with a policy language bias. In *Advances in Neural Information Processing Systems (NIPS-03)*, volume 16, 2004.

[Freund and Schapire, 1996] Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In *Proceedings of the Thirteenth International Conference on Machine Learning (ICML-96)*, pages 148–156, 1996.

[Gordon, 1995] Geoffrey J. Gordon. Stable function approximation in dynamic programming. In Armand Prieditis and Stuart Russell, editors, *Proceedings of the Twelfth International Conference on Machine Learning (ICML-95)*, pages 261–268, San Francisco, CA, 1995. Morgan Kaufmann.

[Gordon, 1996] Geoffrey J. Gordon. Chattering in SARSA(λ). Technical report, School of Computer Science, Carnegie Mellon University, April 1996.

[Gordon, 2001] Geoffrey J. Gordon. Reinforcement learning with function approximation converges to a region. In *Advances in Neural Information Processing Systems (NIPS-00)*, volume 12, 2001.

[Guestrin *et al.*, 2001] Carlos Guestrin, Daphne Koller, and Ronald Parr. Max-norm projections for factored MDPs. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01)*, Seattle, Washington, August 2001.

[Hagan *et al.*, 1996] Martin T. Hagan, Howard B. Demuth, and Mark Beale. *Neural Network Design*. Brooks Cole, 1996.

[Hauskrecht *et al.*, 1998] Milos Hauskrecht, Nicolas Meuleau, Craig Boutilier, Leslie Pack Kaelbling, and Tom Dean. Hierarchical solution of Markov decision processes using macro-actions. In *Proceedings of the Fourteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-98)*, pages 220–229, 1998.

[Haykin, 1999] Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall Inc., 2nd edition, 1999.

[Holland, 1975] John H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975. Reprinted in 1992 by MIT Press, MA.

[Hopcroft *et al.*, 2000] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley Publishing, 2nd edition, November 2000.

[Hornik *et al.*, 1989] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[Howard, 1960] Ronald A. Howard. *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, MA, 1960.

[Isaacson and Madsen, 1976] Dean L. Isaacson and Richard W. Madsen. *Markov Chains: Theory and Applications*. John Wiley & Sons, 1976.

[Kaelbling, 1993] Leslie P. Kaelbling. Hierarchical reinforcement learning: Preliminary results. In *Proceedings of the Tenth Interational Conference on Machine Learning (ICML-93)*, pages 167–173, 1993.

[Kakade and Langford, 2002] Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *Proceedings of the Nineteenth International Conference on Machine Learning (ICML-02)*, 2002.

[Kakade, 2001] Sham Kakade. Optimizing average reward using discounted rewards. In *Proceedings of the Fourteenth Annual Conference on Computational Learning Theory (COLT-01)*, 2001.

[Kakade, 2002] Sham Kakade. A natural policy gradient. In *Advances in Neural Information Processing Systems (NIPS-01)*, volume 13, 2002.

[Kakade, 2003] Sham Kakade. *On the Sample Complexity of Reinforcement Learning*. PhD thesis, University College London, UK, 2003.

[Kearns *et al.*, 2000] Michael Kearns, Yishay Mansour, and Andrew Y. Ng. Approximate planning in large POMDPs via reusable trajectories. In *Advances in Neural Information Processing Systems (NIPS-99)*, volume 12, 2000.

[Kirkpatrick *et al.*, 1983] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, May 1983.

[Konda and Borkar, 1999] Vijay R. Konda and Vivek S. Borkar. Actor-critic–type learning algorithms for Markov decision processes. *SIAM Journal on Control and Optimization*, 38(1):94–123, 1999.

[Konda and Tsitsiklis, 2000] Vijay R. Konda and John N. Tsitsiklis. Actor-critic algorithms. In *Advances in Neural Information Processing Systems (NIPS-99)*, volume 12, 2000.

[Konda, 2002] Vijaymohan Konda. *Actor-Critic Algorithms*. PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, MA, June 2002.

[Lagoudakis and Parr, 2003a] Michail G. Lagoudakis and Ronald Parr. Least-squares policy iteration. *Journal of Machine Learning Research*, 4:1107–1149, 2003.

[Lagoudakis and Parr, 2003b] Michail G. Lagoudakis and Ronald Parr. Reinforcement learning as classification: Leveraging modern classifiers. In *Proceedings of the Twentieth International Conference on Machine Learning (ICML-03)*, Washington DC, 2003.

[Langford and Zadrozny, 2003] John Langford and Bianca Zadrozny. Reducing T-step reinforcement learning to classification. In *Proceedings of the Machine Learning Reductions Workshop*, Chicago, IL, 2003.

[Levner and Bulitko, 2004] Ilya Levner and Vadim Bulitko. Machine learning for adaptive image interpretation. In *Proceedings of the Sixteenth Innovative Applications of Artificial Intelligence Conference (IAAI-04)*, 2004.

[Levner *et al.*, 2003] Ilya Levner, Vadim Bulitko, Lihong Li, Greg Lee, and Russell Greiner. Towards automated creation of image interpretation systems. In *Proceedings of the Sixteenth Australian Joint Conference on Artificial Intelligence (AI-03)*, Perth, Australia, December 2003.

[Levner, 2003] Ilya Levner. Multi resolution adaptive object recognition system: A step towards autonomous vision systems. Master's thesis, Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada, September 2003.

[Li *et al.*, 2003] Lihong Li, Vadim Bulitko, Russell Greiner, and Ilya Levner. Improving an adaptive image interpretation system by leveraging. In *Proceedings of the Eighth Australian and New Zealand Intelligent Information System Conference (ANZIIS-03)*, Sydney, Australia, 2003.

[Li *et al.*, 2004a] Lihong Li, Vadim Bulitko, and Russ Greiner. Batch reinforcement learning with state importance. In *Proceedings of the European Conference on Machine Learning (ECML-04)*, Pisa, Italy, 2004.

[Li *et al.*, 2004b] Lihong Li, Vadim Bulitko, and Russ Greiner. Focusing attention in reinforcement learning. In *AAAI-04 Workshop on Learning and Planning in Markov Processes: Advances and Challenges*, 2004.

[Lin, 1992] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8:293–321, 1992.

[Littman, 1996] Michael L. Littman. *Algorithms for Sequential Decision Making.* PhD thesis, Brown University, Providence, Rhode Island, March 1996. CS-96-09.

[Maclin and Optiz, 1997] Richard Maclin and David Optiz. An empirical evaluation of bagging and boosting. In *the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, 1997.

[Meir and Räsch, 2003] Ron Meir and Gunnar Räsch. An introduction to boosting and leveraging. In *Advanced Lectures on Machine Learning*, LNCS, pages 119–184. Springer, 2003.

[Mitchell, 1997] Tom Mitchell. *Machine Learning.* McGraw-Hill, March 1997.

[Ng and Jordan, 2000] Andrew Y. Ng and Michael Jordan. PEGASUS: A policy search method for large MDPs and POMDPs. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI-00)*, 2000.

[Ng *et al.*, 1999] Andrew Y. Ng, Ronald Parr, and Daphne Koller. Policy search via density estimation. In *Proceedings of the Sixteenth International Conference on Machine Learning (ICML-99)*, Bled, Slovenia, 1999.

[Ng *et al.*, 2004] Andrew Y. Ng, H. Jin Kim, Michael I. Jordan, and Shankar Sastry. Autonomous helicopter flight via reinforcement learning. In *Advances in Neural Information Processing Systems (NIPS-03)*, volume 16, 2004.

[Precup, 2000] Doina Precup. *Temporal Abstraction in Reinforcement Learning.* PhD thesis, University of Massachusetts at Amherst, MA, May 2000.

[Puterman and Shin, 1978] Martin L. Puterman and Moon Chirl Shin. Modified policy iteration algorithms for discounted Markov decision problems. *Management Science*, 24(11):1127–1137, July 1978.

[Puterman, 1994] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* Wiley-Interscience, New York, 1994.

[Quinlan, 1993] J. Ross Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, 1993.

[Resenblatt, 1962] Frank Resenblatt. *Principles of Neurodinamics: Perceptron and Theory of Brain Mechanism.* Spartan Books, Washington D.C., 1962.

[Rumelhart *et al.*, 1994] David E. Rumelhart, Bernard Widrow, and Michael A. Lehr. The basic ideas in neural networks. *Communications of the ACM*, 37(3):87–92, March 1994.

[Russell and Zimdars, 2003] Stuart J. Russell and Andrew L. Zimdars. Q-decomposition for reinforcement learning agents. In *Proceedings of the Twentieth International Conference on Machine Learning (ICML-03)*, Washington DC, August 2003.

[Schapire, 1990] Robert E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, 1990.

[Schölkopf and Smola, 2001] Bernhard Schölkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond.* MIT Press, Cambridge, MA, Decemter 2001.

[Singh and Bertsekas, 1997] Satinder Singh and Dimitri P. Bertsekas. Reinforcement learning for dynamic channel allocation in cellular telephone systems. In *Advances in Neural Information Processing Systems (NIPS-96)*, volume 9, pages 974–980, 1997.

[Singh and Yee, 1994] Satinder Singh and Richard C. Yee. An upper bound on the loss from approximate optimal-value functions. *Machine Learning*, 16(3):227–233, 1994.

[Singh *et al.*, 2000] Satinder Singh, Tommi Jaakkola, Michael L. Littman, and Csaba Szepesvári. Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning*, 38(3):287–308, 2000.

[Singh *et al.*, 2002] Satinder Singh, Diane Litman, Michael Kearns, and Marilyn Walker. Optimizing dialogue management with reinforcement learning: Experiments with the NJFun system. *Journal of Artificial Intelligence Research*, 16:105–133, 2002.

[Singh, 1994] Satinder Singh. *Learning to Solve Markovian Decision Processes.* PhD thesis, University of Massachusetts, Amherst, MA, February 1994.

[Sutton and Barto, 1998] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, MA, March 1998.

[Sutton *et al.*, 1999] Richard S. Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112:181–121, 1999. An earlier version appeared as Technical Report 98-74, Department of Computer Science, University of Massachusetts, Amherst, MA 01003. April, 1998.

[Sutton *et al.*, 2000] Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems (NIPS-99)*, volume 12, 2000.

[Sutton, 1984] Richard S. Sutton. *Temporal Credit Assignment in Reinforcement Learning.* PhD thesis, University of Massachusetts, Amherst, MA, 1984.

[Sutton, 1988] Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

[Sutton, 1990] Richard S. Sutton. Integrated architectures for learning, planning and reacting based on approximating dynamic programming. In *Seventeenth International Conference on Machine Learning (ICML-90)*, pages 216–224, 1990.

[Sutton, 1996] Richard S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems (NIPS-96)*, volume 8, pages 1038–1044, 1996.

[Suykens *et al.*, 2002] Johan Suykens, Tony Van Gestel, Jos De Brabanter, Bart De Moor, and Joos Vandewalle. *Least Squares Support Vector Machines.* World Scientific Pub. Co., Singapore, 2002.

[Tesauro and Galperin, 1997] Gerald Tesauro and Gregory R. Galperin. On-line policy improvement using Monte-Carlo search. In *Advances in Neural Information Processing Systems (NIPS-96)*, volume 9, 1997.

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

[Tesauro, 1995] Gerald Tesauro. Temporal difference learning and TD-Gammon. *Communications of the ACM*, 38(3):58–68, March 1995.

[Tsitsiklis and Van Roy, 1997] John N. Tsitsiklis and Benjamin Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42:674–690, 1997.

[Turney, 2000] Peter Turney. Types of cost in inductive concept learning. In *The Seventeenth International Conference on Machine Learning (ICML-00) Workshop on Cost-Sensitive Learning*, pages 15–21, Stanford, CA, 2000.

[Valiant, 1984] Leslie G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134 – 1142, November 1984.

[Vapnik, 1999] Vladimir Vapnik. *The Nature of Statistical Learning Theory.* Sringer Verlag, NY, 2nd edition, 1999.

[Wang and Dietterich, 1999] Xin Wang and Thomas G. Dietterich. Efficient value function approximation using regression trees. In *Proceedings of the IJCAI-99 Workshop on Statistical Machine Learning for Large-Scale Optimization*, Stockholm, Sweden, 1999.

[Watkins, 1989] Christopher J.C.H. Watkins. *Learning from Delayed Rewards.* PhD thesis, King's College, University of Cambridge, UK, 1989.

[Weaver and Baxter, 1999] Lex Weaver and Jonathan Baxter. Reinforcement learning from state and temporal differences. Technical report, Department of Computer Science, Australian National University, September 1999.

[Williams and Baird, 1993] Ronald J. Williams and Leemon Baird. Tight performance bounds on greedy policies based on imperfect value functions. Technical Report NU-CCS-93-14, College of Computer Science, Northeastern University, 1993.

[Williams, 1992] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

[Yoon *et al.*, 2002] SungWook Yoon, Alan Fern, and Robert Givan. Inductive policy selection for first-order MDPs. In *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence (UAI-02)*, Edmonton, Canada, 2002. Morgan Kaufmann.

[Zadrozny and Langford, 2003] Bianca Zadrozny and John Langford. Cost-sensitive learning by cost-proportionate example weighting. In *Proceedings of the IEEE International Conference on Data Mining (ICDM-03)*, 2003.

[Zhang and Dietterich, 1995] Wei Zhang and Thomas G. Dietterich. A reinforcement learning approach to job-shop scheduling. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 1114–1120, 1995.

Appendix A

Supervised Learning

Supervised learning [Vapnik, 1999] is a subfield of machine learning that addresses the problem of how to learn a mapping from a set of input-output pairs. In contrast to reinforcement learning, the decisions made in supervised learning are single-step. This section briefly introduces the basic concepts, methodologies, and several examples that were used in the thesis.

A.1 Overview

The goal of supervised learning is to induce a function called *hypothesis* to approximate a target function from a set of training pairs, so that the prediction error of the hypothesis is minimized. Formally, let $f : \mathcal{X} \mapsto \mathcal{Y}$ be the target function with domain $\mathcal{X}$ and range $\mathcal{Y}$. A set of training data are given:

$$T = \{\langle x_i, y_i \rangle \mid x_i \in \mathcal{X},\, y_i = f(x_i) \in \mathcal{Y},\, i = 1, 2, \cdots, l\},$$

where the inputs are drawn randomly from an unknown distribution: $x_i \sim D_{\mathcal{X}}$. The learning agent is given a class of hypotheses, $\mathcal{H}$. Each element $h \in \mathcal{H}$ is function mapping $\mathcal{X}$ to $\mathcal{Y}$, and can be viewed as a candidate approximation to f. The prediction error of h on an instance, $\langle x, f(x) \rangle$, is denoted by $\mathrm{Err}(f(x), h(x))$. The *prediction error* of h on the whole input set $\mathcal{X}$ is then defined as:

$$\mathrm{Err}(h) = \sum_{x \in \mathcal{X}} \big(D_{\mathcal{X}}(x) \cdot \mathrm{Err}(f(x), h(x)) \big). \tag{A.1}$$

The goal of a supervised learning agent is to induce the optimal hypothesis h^* from $\mathcal{H}$:

$$h^* = \arg\min_{h \in \mathcal{H}} \mathrm{Err}(h).$$

Regression and Classification

The two most studied supervised learning problems are *regression* and *classification.* In regression, the goal is to learn a *real-value* function (i.e., $\mathcal{Y} \subseteq \mathbb{R}$); the error on an instance is usually defined as the squared error:

$$\mathrm{Err}_{\mathrm{regression}}(f(x), h(x)) = \big(f(x) - h(x)\big)^2,$$

and the prediction error of h on $\mathcal{X}$ is called the *mean squared error* (MSE):

$$\mathrm{Err}_{\mathrm{regression}}(h) = \sum_{x \in \mathcal{X}} \Big(D_{\mathcal{X}}(x) \cdot \big(f(x) - h(x)\big)^2\Big). \tag{A.2}$$

Sometimes we will also use the *root mean squared error* (RMSE) in place of MSE, which is defined as the squared root of MSE: $\mathrm{RMSE}(h) = \sqrt{\mathrm{MSE}(h)}$.

Classification is different from regression in that $\mathcal{Y}$ is a discrete set and the 0/1-loss is often used as the error function. For simplicity, we only consider the binary classification problem where $\mathcal{Y} = \{+1, -1\}$, and the error function is:

$$\mathrm{Err}_{\mathrm{classification}}(f(x), h(x)) = \mathcal{I}\big(f(x) \neq h(x)\big),$$

where

$$\mathcal{I}(A) = \begin{cases} 1, & \text{if } A \text{ is true} \\ 0, & \text{if } A \text{ if false} \end{cases} \tag{A.3}$$

is the *indicator* function. Correspondingly, the error of h on $\mathcal{X}$ is called the *classification error*:

$$\mathrm{Err}_{\mathrm{classification}}(h) = \sum_{x \in \mathcal{X}} \big(D_{\mathcal{X}} \cdot \mathcal{I}\big(f(x) \neq h(x)\big)\big). \tag{A.4}$$

Hypothesis Evaluation

The distribution $D_{\mathcal{X}}$ in the definitions above, however, is usually unknown. Therefore, any learning algorithm cannot minimize the prediction error $\mathrm{Err}(h)$ directly. Instead, they minimize the *regularized empirical error*:

$$\widehat{\mathrm{Err}}(h) = \sum_{i=1}^{l} \mathrm{Err}(f(x_i), h(x_i)), \quad h \in \mathcal{H}.$$

It should be emphasized that the training data are randomly drawn according to the *same* distribution, $D_{\mathcal{X}}$. An important tradeoff has to be made between the complexity of h and the empirical error $\widehat{\mathrm{Err}}(h)$. It has been known that if $\mathcal{H}$ is very complex, then the prediction error $\mathrm{Err}(h)$ may be large even if the empirical error is very small. This is an important issue, but is out of the scope of the thesis.

In practice, *k-fold cross-validation* is often used to evaluate a learning algorithm. A common approach would be to divided the original training data set into several subset: $T_1, T_2, \cdots, T_k$. Then cross-validation repeats the training/testing process k times. At the i-th iteration, $T - T_i$ is used as the training set while T_i as the test set. A learning algorithm is used to induce a hypothesis h_i, whose empirical errors on the training set $(T - T_i)$ and the test set (T_i) are denoted by $\hat{e}_i^{\text{train}}$ and $\hat{e}_i^{\text{test}}$, respectively. Finally, the empirical errors in different iterations are averaged and the mean values become the final training/test errors:

$$\begin{aligned} \widehat{Err}^{\text{train}} &= \frac{1}{k}\sum_{i=1}^{k} \hat{e}_i^{\text{train}} \\ \widehat{Err}^{\text{test}} &= \frac{1}{k}\sum_{i=1}^{k} \hat{e}_i^{\text{test}} \end{aligned}$$

In the extreme case where the data are limited, a special form of cross-validation called *leave-one-out* (LOO) is used. In LOO, each subset T_i contains exactly one training datum, and the number of folds $k = |T|$.

A.2 PAC Learning

Ensemble learning techniques have been used in our experiments. This is a very useful and interesting topic in machine learning. Before discussing ensemble learning in the following subsection, the *probably approximately correct* (PAC) learning model [Valiant, 1984] has to be introduced.

Definition 12 *Assuming the same notation used in the previous section, a function class $\mathcal{F}$ is* PAC-learnable *by a learner L using a hypothesis space $\mathcal{H}$ iff, $\forall f \in \mathcal{F}$, $\forall D_{\mathcal{X}}$, $\forall 0 < \epsilon, \delta < 1/2$, L will output a hypothesis $h \in \mathcal{H}$ such that* $\text{Err}(h) < \epsilon$, *with probability at least* $(1 - \delta)$, *in time polynomial in* $1/\epsilon$ *and* $1/\delta$. *[Mitchell, 1997]*

However, not all target function classes are PAC-learnable by all learners. For example, consider a linear hypothesis space: $\mathcal{H} = \{h \mid h(x) = w \cdot x\}$. We cannot expect any $h \in \mathcal{H}$ to represent a nonlinear target function with an arbitrary small error. In such cases, the learner L can only achieve reasonably well. To distinguish a good learner from a poor one, we introduce two definitions adopted from [Duffy and Helmbold, 2000]:

Definition 13 *A* strong *PAC learner L for a target function class $\mathcal{F}$ has the property that $\forall D_{\mathcal{X}}$, $\forall f \in \mathcal{F}$, $\forall 0 < \epsilon, \delta < 1/2$, with probability at least* $(1 - \delta)$, *L outputs a hypothesis h with* $\text{Err}(h) < \epsilon$, *in time polynomial in* $1/\epsilon$ *and* $1/\delta$.

Definition 14 *A* weak *PAC learner is similar to a strong PAC learner, except that it needs to satisfy the conditions only for a particular* (ϵ_0, δ_0) *pair such that* $0 < \epsilon_0, \delta_0 < 1/2$.

A.3 Ensemble Learning

Ensemble learning [Dietterich, 2000a] is a class of learning algorithms that construct a set of base hypotheses and then make predictions on new data by taking a vote of the their predictions. Successful ensemble methods developed in recent years include boosting, leveraging, and bagging.

Boosting and Leveraging.

Boosting was first proposed in [Schapire, 1990], and then has attracted a lot of research interests in the past decade [Meir and Räsch, 2003]. A number of boosting algorithms were developed, which provably *boost* weak PAC learners to strong PAC learners by iteratively calling the weak learner to produce base hypotheses and then combining them linearly. At iteration k, they modify the training data or their weights according to the performance of previous base hypotheses; then they call the weak learner to produce the k-th base hypothesis, h_k. When predicting an instance $x \in \mathcal{X}$, a linear combination is used to combine the base hypotheses:

$$H(x) = \sum_k \alpha_k h_k(x),$$

where α_k are computed during the training phase. A well-known boosting algorithm for classification with great success both in theory and in practice is AdaBoost [Freund and Schapire, 1996]. Another boosting algorithm for regression called SquareLev.R was proposed recently [Duffy and Helmbold, 2002].

Leveraging algorithms are boosting-like algorithms except that they do not enjoy the PAC-boosting property [Duffy and Helmbold, 2000], i.e., they cannot *provably* boost a weak PAC learner to a strong PAC learner. But they can work well in practice.

Bagging.

Bagging [Breiman, 1996] (Bootstrap AGGregatING) operates very similarly to boosting, except they do not modify the training data or their weights, but only build base hypotheses, h_k, by presenting the learner with *bootstrap replicates* consisting of training data drawn randomly with replacement from the original training set. When making predictions on a new instance $x \in \mathcal{X}$, a majority vote of h_k becomes the final, ensemble hypothesis:

$$H(x) = \sum_k \alpha_k h_k(x).$$

The parameters α_k above are usually set to be equal and sum up to 1, in which case the vote is an *unweighted* majority vote. In some cases, setting them differently may help [Draper and Baek, 1998]:

$$\alpha_k = \frac{1}{\widehat{\mathrm{Err}}(h_k)}.$$

Bagging was shown effective to reduce the variance of the classifiers or regressors, especially when they are unstable (e.g., decision trees and neural networks). Although it is simple to use and easy to parallelize, most empirical studies show that it is often outperformed by boosting [Dietterich, 2000b; Maclin and Optiz, 1997].

A.4 Examples of Supervised Learning

In this section, we give a short introduction to several supervised learning algorithms that were used in our empirical studies. Each of them has its own architecture and parameters to represent a function, and the corresponding learning algorithms differ.

A.4.1 Gradient Descent

Gradient descent (GD) is a class of general methods that are guaranteed to converge to a local optimum by performing gradient descent to minimize a target error function. The gradient of the error function at any point is a vector pointing in the direction of *steepest descent*.

Simple Gradient Descent

Specifically, let the hypothesis be $h(x, \theta)$, where x is the input vector and $\theta \in \mathbb{R}^k$ is the k-dimensional parameter. The error function, denoted by $e(\theta)$, is a smooth, nonnegative, scalar function. Note that $e(\theta)$ can be several prediction error functions mentioned in the previous sections including Equation A.2. For the error function in Equation A.4 which is not smooth, it can be approximated by other smooth error function. For example, $h(x)$ can be a "soft" classifier which can smooth: $h : \mathcal{X} \mapsto [-1, 1]$. When making a prediction on a date point x, the class label is $+1$ if $h(x) > 0$ and -1 if $h(x) < 0$.

Starting from an initial value θ_0 which is usually a random value or zero, the learning agent iteratively updates the parameter to minimize $e(\theta)$ gradually. At the t-th training step, the gradient is computed by:

$$\nabla e(\theta) = \left(\frac{\partial e}{\partial \theta^1}, \frac{\partial e}{\partial \theta^2}, \cdots, \frac{\partial e}{\partial \theta^k} \right)^T,$$

where θ^i denotes the i-th component of θ. Then the value θ_t is updated by being moved towards a new value that minimizes $e(\theta)$ with a small, positive step size α_t:

$$\theta_{t+1} \leftarrow \theta_t - \alpha_t \nabla e(\theta_t), \tag{A.5}$$

The step-size parameter α_t is important to the convergence of the gradient descent method. If it is too small, θ_t may not converge to a local minimum; if it is too large, θ_t may fail to converge by oscillating around the local minimum. It is well-known that if α_t satisfies Assumption 4 below, then gradient descent with Equation A.5 is convergent to a local minimum. Intuitively, Equation A.6 guarantees that α_t is large enough to reach any point in $\mathbb{R}^k$ from any initial value θ_0, and Equation A.7 guarantees that the oscillation of θ_t decreases over time and therefore θ_t converges.

Assumption 4 *The step-size parameters, α_t, satisfy the two conditions:*

$$\sum_{t=0}^{\infty} \alpha_t = \infty, \tag{A.6}$$

$$\sum_{t=0}^{\infty} \alpha_t^2 < \infty. \tag{A.7}$$

Incremental and Stochastic Gradient Descent

In many learning problems, as mentioned before, the distribution $D_{\mathcal{X}}$ of input vectors is unknown and it is infeasible to compute the error function $e(\theta)$. Furthermore, the difficulty of computing $e(\theta)$ exactly is increased if the input space $\mathcal{X}$ is large, or if the training data come in one by one. In such a case, incremental gradient descent updates θ incrementally to minimize the errors at data points drawn randomly according to $D_{\mathcal{X}}$.

Specifically, at the t-th training step of incremental gradient descent, the training datum $x_t \sim D_{\mathcal{X}}$. (If the training data set is fixed, then x_t is drawn randomly from the training data set.) The error at point x_t is defined as $e(x_t, \theta)$ which, for example, can be the squared error in the regression problem. Then θ_t is updated by

$$\theta_{t+1} \leftarrow \theta_t - \alpha_t \cdot \nabla e(x_t, \theta_t). \tag{A.8}$$

LMS and Linear Function Approximation

As an example, we briefly introduce the *least-mean-square* (LMS) algorithm for linear function approximation. Using the same notation as before, a linear function approximator is a mapping:

$$h(x, \theta) = x \cdot \theta = \sum_{i=1}^{k} x^i \theta^i.$$

The LMS algorithm aims at minimizing the squared error:[1]

$$e(\theta) = \frac{1}{2}\sum_x D_{\mathcal{X}}\left(f(x) - h(x,\theta)\right)^2 .$$

Differentiating $e(\theta)$ with respect to the parameter θ yields

$$\nabla e(\theta) = -\sum_x D_{\mathcal{X}}\big(f(x) - h(x,\theta)\big) \cdot x,$$

and the corresponding update rule is:

$$\theta_{t+1} \leftarrow \theta_t + \alpha_t \cdot \big(f(x) - h(x,\theta)\big) \cdot x,$$

which is easy to compute.

A.4.2 Artificial Neural Networks

Artificial neural networks [Haykin, 1999] (ANNs), inspired by biologic structures and processes of human brains, provide a general and practical way of supervised learning. They have been studied for long time and have been widely used in practice [Rumelhart *et al.*, 1994].

Neuron and Neural Networks

In ANN, a learning system (agent) is represented by a collection of interacting *neurons*. As a fundamental element to the operation of a neural network, a neuron maps the input signal to an output signal which becomes the input of other neurons or the system output (Figure A.1). Formally, let $x \in \mathbb{R}^k$ be the input signal and the neuron outputs $y \in \mathbb{R}$. The mapping implemented by the neuron is usually modelled as:

$$y = \phi(w \cdot x + b) = \phi(\sum_{i=1}^{k} w_i x_i + b),$$

where w is the weight vector of the neuron weighing the each input component x_i, $\phi(v)$ is a nonlinear *activation function*. There exist many choices for the activation function, including the *sigmoid* function used in our experiment (Figure A.2):

$$\phi_{\text{sigmoid}}(v) = \frac{1}{1 + e^{-v}}. \tag{A.9}$$

A single neuron already has some level of learning ability [Resenblatt, 1962]. When a collection of neurons are linked together and interacting with each other, the whole system (neural network) is able to demonstrate a more complex behavior and learning ability [Hornik

[1]the factor 1/2 in the definition does not affect the gradient direction of $e(\theta)$, but will simplify the equations later.

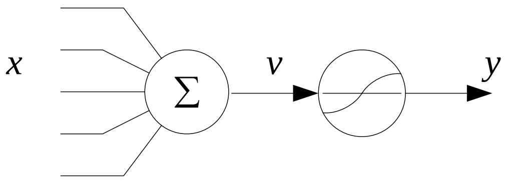


Figure A.1: The mathematical model of a neuron.

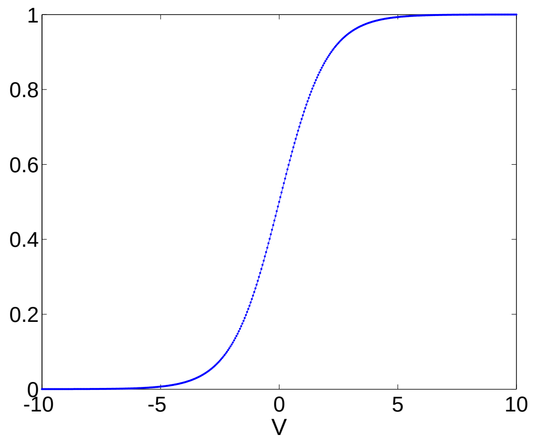


Figure A.2: The Sigmoid function (Equation A.9).

et al., 1989]. There have been a number of neural network architectures/typologies, among which is the *multilayer feedforward neural networks*. Figure A.3 illustrates such an architecture with one hidden layer. Each circle in the figure is a neuron, and the directed edges show how signals are transmitted within the network.

The Back-Propagation Algorithm

A well-known algorithm for training multilayer feedforward neural networks is the *back-propagation* (BP) algorithm. The basic idea is to apply the chain rule in computing the gradient (Equation A.5). Derivation and detailed update equations can be found in a number of textbooks for neural networks and supervised learning [Hagan *et al.*, 1996; Haykin, 1999; Mitchell, 1997].

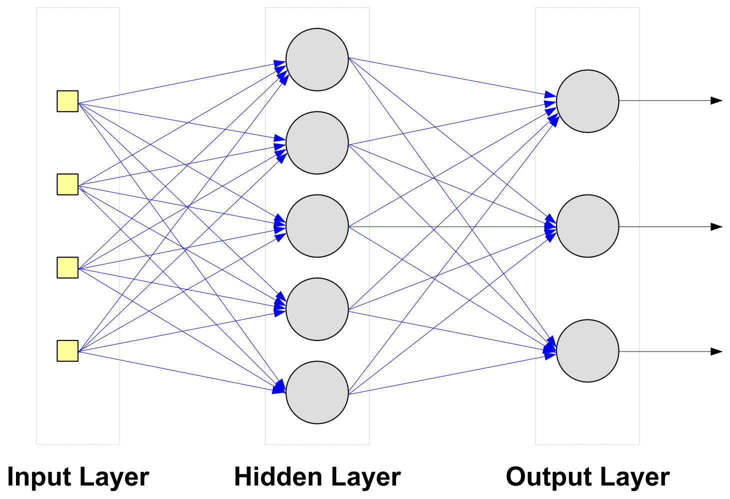


Figure A.3: A multi-layer feed-forward neural network with one hidden layer.