

# Feature Selection for Value Function Approximation

by

Gavin Taylor

Department of Computer Science  
Duke University

Date: \_\_\_\_\_

Approved:

---

Ronald Parr, Supervisor

---

Vincent Conitzer

---

Mauro Maggioni

---

Peng Sun

Dissertation submitted in partial fulfillment of the requirements for the degree of  
Doctor of Philosophy in the Department of Computer Science  
in the Graduate School of Duke University  
2011

ABSTRACT  
(Computer Science)

Feature Selection for Value Function Approximation

by

Gavin Taylor

Department of Computer Science  
Duke University

Date: \_\_\_\_\_

Approved:

---

Ronald Parr, Supervisor

---

Vincent Conitzer

---

Mauro Maggioni

---

Peng Sun

An abstract of a dissertation submitted in partial fulfillment of the requirements for  
the degree of Doctor of Philosophy in the Department of Computer Science  
in the Graduate School of Duke University  
2011

Copyright © 2011 by Gavin Taylor  
All rights reserved

# Abstract

The field of reinforcement learning concerns the question of automated action selection given past experiences. As an agent moves through the state space, it must recognize which state choices are best in terms of allowing it to reach its goal. This is quantified with value functions, which evaluate a state and return the sum of rewards the agent can expect to receive from that state. Given a good value function, the agent can choose the actions which maximize this sum of rewards. Value functions are often chosen from a linear space defined by a set of features; this method offers a concise structure, low computational effort, and resistance to overfitting. However, because the number of features is small, this method depends heavily on these few features being expressive and useful, making the selection of these features a core problem. This document discusses this selection.

Aside from a review of the field, contributions include a new understanding of the role approximate models play in value function approximation, leading to new methods for analyzing feature sets in an intuitive way, both using the linear and the related kernelized approximation architectures. Additionally, we present a new method for automatically choosing features during value function approximation which has a bounded approximation error and produces superior policies, even in extremely noisy domains.

To Allison, who supported me more than I knew was possible.

# Contents

<b>Abstract</b>	<b>iv</b>
<b>List of Tables</b>	<b>ix</b>
<b>List of Figures</b>	<b>x</b>
<b>Acknowledgements</b>	<b>xi</b>
<b>1 Introduction</b>	<b>1</b>
1.1 MDPs and Value Functions . . . . .	3
1.2 Topics . . . . .	4
1.2.1 Regression . . . . .	5
1.2.2 Models . . . . .	5
1.2.3 Regularization . . . . .	6
1.3 Document Organization and Contributions . . . . .	6
<b>2 Notation and Past Work</b>	<b>9</b>
2.1 Formal Problem Statement and Notation . . . . .	9
2.1.1 MDPs, Value Functions, and Policies . . . . .	9
2.1.2 Sampling and the Bellman Operator . . . . .	10
2.1.3 Value Function Approximation Architectures . . . . .	11
2.2 Value Function Calculation Algorithms . . . . .	13
2.2.1 Value Iteration . . . . .	13
2.2.2 Fitted Value Iteration . . . . .	14

2.2.3	Policy Iteration . . . . .	15
2.2.4	Least-Squares Policy Iteration . . . . .	15
2.2.5	Linear Programming . . . . .	16
<b>3</b>	<b>Linear Value Function Approximation</b>	<b>17</b>
3.1	Previous Work . . . . .	18
3.1.1	Linear Fixed-Point Methods . . . . .	18
3.1.2	Linear Feature Generation . . . . .	21
3.2	Linear Models . . . . .	22
3.3	Linear Fixed-Point Solution and Linear Model Solution Equivalence .	24
<b>4</b>	<b>Kernel-Based Value Function Approximators</b>	<b>26</b>
4.1	Previous Work . . . . .	26
4.1.1	Kernelized Regression . . . . .	27
4.1.2	Kernelized Value Function Approximation . . . . .	28
4.2	A General Kernelized Model-Based Solution . . . . .	30
4.3	Equivalence . . . . .	32
<b>5</b>	<b>Analysis of Error</b>	<b>37</b>
5.1	Error in Linear Value Function Approximations . . . . .	37
5.1.1	Experimental Results . . . . .	39
5.2	Error in Kernel-Based Value Function Approximation . . . . .	44
5.2.1	Experimental Results . . . . .	47
5.3	Generalized Analysis . . . . .	49
<b>6</b>	<b><math>L_1</math>-Regularization For Feature Selection</b>	<b>52</b>
6.1	Previous Work . . . . .	53
6.1.1	Approximate Linear Programming . . . . .	53
6.1.2	$L_1$ Regularization for Regression . . . . .	55

6.1.3	$L_1$ Regularization for Value Function Approximation . . . . .	57
6.2	$L_1$ -Regularized Approximate Linear Programming . . . . .	58
6.3	Theoretical Bounds . . . . .	60
6.3.1	Noiseless RALP . . . . .	61
6.3.2	Noisy RALP . . . . .	71
6.4	Experimental Results . . . . .	74
6.4.1	Benefits of Regularization . . . . .	75
6.4.2	Benchmark Problems . . . . .	76
<b>7</b>	<b><math>L_1</math>-Regularized Approximate Linear Programming in Noisy Domains</b>	<b>79</b>
7.1	Smoothers and Averagers for Value Function Approximation . . . . .	80
7.2	Locally Smoothed $L_1$ -Regularized Approximate Linear Programming	81
7.3	Theoretical Results . . . . .	82
7.4	Experimental Results . . . . .	84
<b>8</b>	<b>Future Work</b>	<b>89</b>
<b>9</b>	<b>Summary and Conclusions</b>	<b>92</b>
9.1	Linear Analysis . . . . .	92
9.2	Kernel Analysis . . . . .	93
9.3	Automatic Feature Selection . . . . .	94
	<b>Bibliography</b>	<b>96</b>
	<b>Biography</b>	<b>101</b>

# List of Tables

4.1	Previously introduced methods of kernelized value-function approximation are equivalent to the novel model-based approximation . . . .	36
7.1	Performance of LS-RALP and RALP for the noisy mountain car . . .	87

# List of Figures

3.1	Illustration of the linear fixed point . . . . .	19
5.1	Illustration of the two-room problem from Mahadevan and Maggioni (2006) . . . . .	42
5.2	Decomposition of the Bellman error for three different problems . . .	45
5.3	Two-room domain . . . . .	48
5.4	Decomposition of the Bellman error for the continuous two-room problem . . . . .	50
6.1	A simple four-state chain with no action choices. . . . .	54
6.2	Illustration of the geometry of $L_1$ regularization . . . . .	56
6.3	Illustration of the effect of $L_1$ regularization on regression . . . . .	57
6.4	Illustration of Lemma 6.3.5 . . . . .	69
6.5	Illustration of the effect of noise in RALP . . . . .	72
6.6	Comparison of the objective value of RALP with the true error. . . .	76
6.7	Comparison of the performance of RALP for multiple values of $\psi$ . .	76
6.8	Comparison of the performance of RALP and ALP for an increasing number of features . . . . .	76
6.9	RALP performance on pendulum as a function on the number of episodes. . . . .	76
6.10	RALP performance on bicycle as a function on the number of episodes.	76
7.1	Performance of LS-RALP, RALP, and LSPI for the noisy pendulum .	86

# Acknowledgements

High school flowed smoothly into college, and college into graduate school. The completion of this document is the first time that I feel I have a discontinuity in my life path. The opportunity to reflect and thank those who have helped me get to this point is a welcome one.

This document is the most significant thing I have done; I say this in hopes that those I thank on this page understand what it means to me to thank them here in particular. Including their names in this document is necessary, as it would not exist without them.

First, my fiancée Allison, to whom this document is dedicated. Her constant and unflagging belief in me was motivation and support, as I tried to be what she believes me to be.

Next, my mother, who has always kept me going with both sticks and carrots. Any work ethic and high standards I may have are gifts from her, given through both nature and long, hard, thankless nurture.

Next, my advisor Ron, who was exactly who I needed him to be. An advisor must know which buttons to push; he must choose when to remediate, and when to expect more, when to guide and when to disappear to force independent work. Ron played those buttons like a virtuoso.



Next, Ron's research group and my graduate school friends. In particular, this includes my extraordinary friends Mac Mason, Susanna Ricco, and Neeti Wagle, whom I admire greatly and with whom I've shared



too many experiences and emotions to trivialize by listing; Christopher Painter-Wakefield, who taught me often during my first few years and set a consistent example throughout my graduate career; and Erik Halvorson, for one very important conversation during a difficult time.

Finally, those collaborators not yet listed whose work appears in this document: Lihong Li, Michael Littman, Marek Petrik, and Shlomo Zilberstein. I thank them for sharing their effort, knowledge, and high standards.

On a less emotional but no less vital note, I acknowledge monetary support from: Duke University, the Duke University Center for Theoretical and Mathematical Science, DARPA CSSG HR0011-06-1-0027, and NSF IIS-0713435,

I am truly thankful to the people on this page, most of whom I will never be able to repay in kind. I am thankful for the friendship and example they have set for me, and am humbled by the fortune I have had in being surrounded by such remarkable people.

# 1

## Introduction

Imagine you are a graduate student and a fan of a college basketball team. At the last minute, you have been offered tickets to attend an important game. Unfortunately, the game overlaps a group meeting with your advisor. When you make your decision of whether to attend the game, there are some things to consider. The first is long-term versus short-term benefit; in the long term, attending the game may damage your relationship with your boss, while in the short term you may experience tremendous excitement if your team wins. The second is the likelihood of receiving these benefits; your advisor may not notice your absence, or your team may not win. Therefore, you must weigh the expected short- and long-term benefit of attending the game versus the expected short- and long-term benefit of not attending the game. The only information you have to base your decision on is past experience watching the team and in past experiences with your advisor.<sup>1</sup>

Now imagine you are a shopkeeper purchasing wares to sell for a profit. Just as the graduate student was risking his advisor's wrath for a win which may not

---

<sup>1</sup> The author would like to note that he attended the game, that Davidson won its first NCAA Tournament game in 39 years, and that his advisor absolutely did notice his absence. He will leave the question "was it worth it" unanswered until such time that this document has been approved.

happen, you are now risking capital and overhead. Perhaps your product will fall out of fashion, or perhaps the overhead of storage will increase. Again, the only information you can base your decision on is past experience.

Finally, imagine you are a child, new to bicycling, who has found himself in a race. At a turn in the race, do you bank sharply, risking injury in the short-term for a possible long-term victory? Or, has your past experience led you to channel the tortoise, causing you to choose to ride slower and steadier?

These questions, for which we must use past experience to inform decision-making in an uncertain environment, are at the heart of intelligence. Smart people notice features in the world around them which tend to be correlated with particular outcomes; for example, experienced bicyclists know they cannot turn as sharply on gravel as they can on asphalt, and shopkeepers know not to buy short-sleeved shirts in November. In other words, there are characteristics of the current state (the road material or the month) which hint at the desirability of our current situation and inform our decision-making.

The goal of the field of reinforcement learning is to automate this process, so autonomous agents can use past experiences to make good decisions, even when the outcomes of those decisions are uncertain. We refer to our title, “Feature Selection for Value Function Approximation.” The purpose of a value function is to quantify the desirability of a state (a state in which your wheels have slipped is undesirable, and so has low value), and the purpose of feature selection is to choose those characteristics which imply a low or high value (“gravel road” or “November”). If we know what from our past experience to pay attention to, we can make good decisions. Choosing those characteristics to help us accurately judge the value of a state is the subject of this document.

## 1.1 MDPs and Value Functions

In reinforcement learning, we model problems as Markov Decision Processes (MDPs). Because of their applicability, MDPs are studied not only in reinforcement learning's superfields of Artificial Intelligence and Machine Learning, but also in Economics, Statistics, and Operations Research. The MDP framework includes several assumptions. First, we assume the world is perfectly-observable; that is, any question we choose to ask about the current state can be answered with certainty. "The road is probably gravel" is not a feature that fits into our framework; "the road is gravel" is. The second assumption we make is the Markovian assumption, which is that history is irrelevant in determining the outcome of taking an action in a certain state. The fact that our graduate student is holding a ticket is relevant; how he came to acquire that ticket is not. Finally, we assume there is some preference towards receiving short-term benefit over long-term benefit; we would choose a path which would deliver some benefit now over a path which will deliver the same benefit in ten days.

An MDP consists of a set of states, a set of actions, a reward function, a transition function, and a discount factor, which we explain in turn.

- The state space defines the set of all possible situations an agent can be in. A state is a complete description of the agent; a bicycle's state could consist of the velocity of the bicycle, the angle of the bicycle from the road, the angle of the bicycle's handlebars, the coefficient of friction of the road material, and the distance to the finish line.
- The actions are the options available to the agent; the bicyclist could shift his weight, turn the handlebars, pedal harder, or apply the brakes.
- The reward function maps a state to the immediate benefit of being at that state. Reward is a numerical measurement of how much benefit we receive at

a particular state. A state in which you have just won a race delivers a high reward; a state in which you have fallen and have hurt your knee delivers a low reward.

- The transition function determines the rules of moving through the space. It is impossible to go from a mile away from the finish line to being at the finish line without traversing several other states; this is captured in the transition function.
- Finally, the discount factor determines how much we prefer current reward to one received in the next state.

When choosing between actions, the agent will choose the action which it expects will provide the highest sum of discounted rewards; this expected sum of discounted rewards is the definition of value. In other words, a state in which the bicyclist is about to fall does not have an associated negative reward, as nothing immediately bad is occurring. However, it will have a low value, as we can expect to receive negative reward soon.

This expectation, of course, depends on a knowledge of the reward and transition functions; we have learned that being at too steep of an angle tends to lead to falling, and that falling tends to hurt, giving us negative reward.

The problem of value function approximation is the problem of using past data to calculate this expectation. The problem of feature selection is the discovery of which calculations on the state space correlate with high or low value and thus, allow us to predict value.

## 1.2 Topics

The contributions in this document revolve around the application of several simple ideas to value function approximation. We review these topics from a high level here.

### 1.2.1 Regression

Regression and value function approximation are similar in several ways, and different in one major one. Regression is the process of reproducing a function based on data drawn from that function; that is, given a set of data  $((x_1, y_1), (x_2, y_2), \dots, (x_n, y_n))$ , where a single datum  $(x_i, y_i)$  means  $y_i = f(x_i) + \epsilon_i$  for some function  $f$  and noise term  $\epsilon_i$ . We create a predictive function  $\hat{f}(x)$  such that for all  $(x_i, y_i)$ ,  $\hat{f}(x_i) \approx y_i$ . Value function approximation is similar; we have a data set, and from that data set, we are constructing a function.

The difference is that in value function approximation, we cannot sample values; we can only sample transitions and rewards. Nevertheless, regression can be a helpful tool for value function approximation, and appears in Chapters 3, 4, and 5.

### 1.2.2 Models

The model of an MDP consists of the transition and reward functions. As noted above in Subsection 1.2.1, we cannot perform regression on value functions, but we can perform regression on our transition and reward functions. One of the main contributions of this document is that accurate approximations of the transition and reward functions imply an accurate approximation of the value function. In fact, we show that even value function approximation methods which make no explicit attempt to perform model approximations are equivalent to easy-to-understand methods which first perform regression on the transition and reward functions, and then use those approximations to calculate a value function. This enables us to analyze approximation schemes and feature sets in an intuitive way. Model approximations appear in Chapters 3, 4, and 5.

### 1.2.3 Regularization

MDPs are not deterministic; taking a certain action at a specific state will not always have the same effect. For example, if our graduate student attends the game, we do not know if he will enjoy a win or suffer a loss; there is chance. The effects of this chance in the data is samples which deviate from expectation; this deviation is referred to as “noise.” The danger of noise is that an approximator may fit each individual data point too precisely, losing the overall shape of the function; this is known as “overfitting.”

This problem is common to regression as well, and has resulted in the application of regularization. Regularization limits the expressiveness of the approximation function  $\hat{f}(x)$ , limiting it to approximating the general curve of the function, rather than reach each datum. Previous work by Farahmand et al. (2008) and Kolter and Ng (2009) has demonstrated the benefits of regularization in value function approximation. Chapters 4, 6, and 7 not only expand on this use of the smoothing effect of regularization, but also apply a specific form of regularization to perform automatic feature selection.

## 1.3 Document Organization and Contributions

This document explores different approaches to understanding and performing feature selection. Chapter 2 introduces notation and fundamental value function approximation algorithms. Chapter 3 encapsulates a discussion on methods for approximation in which the feature set has been determined *a priori*. This chapter introduces linear model approximations in Section 3.2. It also includes our first contribution, a demonstration of equivalence between model-free and model-based linear value function approximators, which is useful for analyzing feature sets. We presented this work in Parr et al. (2008); it was also discovered independently by

Schoknecht (2002) and Sutton et al. (2008), though each paper uses this observation to make different contributions.

Chapter 4 contains results regarding kernelized value function approximation methods, which were first presented by Taylor and Parr (2009). Kernelized methods are an approach to avoiding the feature selection problem by replacing linear features with kernel functions between states. We contribute a general kernelized, model-based approximation method which is presented in Section 4.2, and demonstrate its equivalence to previously presented methods in Section 4.3. This equivalence not only allows us to analyze kernelized methods and regularization through a model-based lens, but also unifies the field by showing equivalence between methods previously thought to be different.

Chapter 5 contains results from both Parr et al. (2008) and Taylor and Parr (2009) regarding approximation error analysis given a previously-selected set of features or kernel and regularization parameters. Section 5.1 demonstrates a model-based Bellman error decomposition convenient for analyzing feature selection, as demonstrated by experiments in Subsection 5.1.1. Section 5.2 contains a similar analysis of Bellman error in the kernelized approximation context, and Subsection 5.2.1 contains demonstrative experiments. Finally, Section 5.3 expands this decomposition to any fixed-point approximation architecture. This chapter uses the equivalencies of Chapters 3 and 4 to analyze feature selection in an intuitive way.

Chapter 6 then shifts from feature sets chosen *a priori* to automatic feature selection using  $L_1$  regularization. Section 6.2 discusses our method,  $L_1$ -regularized Approximation Linear Programming (RALP), which was first introduced by Petrik et al. (2010). This chapter includes both bounds on approximation error (Section 6.3) and experimental results which demonstrate the utility of RALP (Section 6.4).

Section 7.2 then addresses RALP's frailty in noisy environments, by introducing Locally Smoothed  $L_1$ -Regularized Approximate Linear Programming (LS-RALP). It

then presents proof of LS-RALP's benefits in Section 7.3 and experimental results in Section 7.4.

Finally, we present some ideas for future work in Chapter 8, and make some concluding remarks in Chapter 9.

# 2

## Notation and Past Work

This chapter fulfills two purposes. The first is to introduce the notation we will use in this document. The second is to introduce important past work in value function calculation which is not directly related to our contributions. Work that is directly related will be introduced in the relevant chapters.

### 2.1 Formal Problem Statement and Notation

In this section, we introduce notation and the mathematical underpinnings of the problem.

#### *2.1.1 MDPs, Value Functions, and Policies*

This work concerns controlled Markov processes, which are referred to as Markov decision processes (MDP):  $M = (\mathcal{S}, \mathcal{A}, P, R, \gamma)$ . Given a state  $s_i \in \mathcal{S}$ , the probability of a transition to a state  $s_j$  as a result of action  $a \in \mathcal{A}$  is given by  $P(s_j|s_i, a)$  and results in an expected reward of  $R(s_i)$ .

We are concerned with finding value functions  $V$  that map each state  $s_i \in \mathcal{S}$  to the expected total  $\gamma$ -discounted reward for the process. In particular, we would like

to find the solution to the Bellman equation.

$$V^*(s_i) = R(s_i) + \max_{a \in \mathcal{A}} \left( \gamma \sum_{s_j \in \mathcal{S}} P(s_j | s_i, a) V^*(s_j) \right)$$

Because the max operator results in the best action being taken at each state,  $V^*$  is also known as the *optimal* value function.

Value functions can be useful in creating or analyzing a policy  $\pi$ , which maps each state  $s_i \in \mathcal{S}$  to an action  $a \in \mathcal{A}$ . The value function of a policy  $\pi$  is

$$V_\pi(s_i) = R(s_i) + \gamma \sum_{s_j \in \mathcal{S}} P(s_j | s_i, \pi(s_i)) V_\pi(s_j).$$

Ultimately, we seek an optimal policy

$$\pi^*(s_i) = \operatorname{argmax}_{a \in \mathcal{A}} \left( R(s_i) + \gamma \sum_{s_j \in \mathcal{S}} P(s_j | s_i, a) V^*(s_j) \right).$$

If the policy for an MDP is treated as a constant, then the MDP induces a Markov reward process (MRP). The task of computing the value function for the resulting MRP is sometimes referred to as policy evaluation.

For a given policy  $\pi$  and a finite state space, we can denote the transition function  $P$  as a matrix, where  $P_\pi(i, j) = P(s_j | s_i, \pi(s_i))$ .

### 2.1.2 Sampling and the Bellman Operator

The goal of reinforcement learning is to gain intelligent behavior by learning from samples, or experiences. The question of sampling is therefore an important one. In this document, we will discuss two types of samples: *samples with expectation*, and *simple samples*. Samples with expectation consist of  $\sigma = (s, a, R(s), P(\cdot | s, a))$  tuples. We will denote a set of samples with expectation as  $\Sigma$ . Simple samples consist of  $\dot{\sigma} = (s, a, r, s')$  tuples, where  $r$  and  $s'$  are not the expected reward and

next state, but are instead the reward and next state that were actually experienced. A set of simple samples will be denoted  $\dot{\Sigma}$ . Clearly, samples with expectation are more useful. However, these types of samples require either *a priori* knowledge of  $P$  and  $R$  or a great deal of control over the sampling agent; practically speaking, this may be unrealistic. Therefore, simple samples are often more attainable. An element of a sample  $\sigma$  will be denoted with superscripts; that is, the  $s$  component of a  $\sigma = (s, a, R(s), P(\cdot|s, a))$  sample will be denoted  $\sigma^s$ .

We will refer to the Bellman operator  $T$ , where

$$TV(s) = R(s) + \gamma \max_{a \in \mathcal{A}} \left[ \sum_{s_i \in \mathcal{S}} P(s_i|s, a) V(s_i) \right],$$

and  $T_a$  refers specifically to

$$T_a V(s) = R(s) + \gamma \sum_{s_i \in \mathcal{S}} P(s_i|s, a) V(s_i).$$

Because the summation captures the expected value at the next state, we also refer to  $T$  as the Bellman operator with expectation. Simple sampling also has an associated sampled Bellman operator  $\dot{T}$ , where

$$\dot{T}_{\sigma} V(\dot{\sigma}^s) = \dot{\sigma}^r + \gamma V(\dot{\sigma}^{s'}).$$

### 2.1.3 Value Function Approximation Architectures

As the state space grows larger, it becomes more and more difficult to calculate the Bellman equation for every state, making it necessary to approximate the value function. In this document, we will discuss two related approximation architectures, linear approximation and kernelized approximation.

Linear approximation consists of the linear weighting of possibly non-linear features. The problem of *feature selection* in value function approximation refers to the

construction of a basis matrix  $\Phi$  of size  $|\mathcal{S}| \times k$ , preferably where  $k \ll |\mathcal{S}|$ . The basis matrix consists of  $k$  *basis functions* or *features* defined over the states of the MDP. For continuous or very large MDPs, where  $|\mathcal{S}|$  is too large, features are calculated on a set of sampled states. A basis matrix is a successful one when a linear combination of feature values produces an approximate value function  $\hat{V} = \Phi \mathbf{w}$ , which is close to  $V^*$ . We use  $\mathbf{s}$  and  $\mathbf{r}$  to represent vectors of sampled states and rewards, respectively. The row vector of feature values at a state  $s$  will be denoted  $\Phi(s)$ .

This form of linear representation allows for the calculation of an approximate value function in a lower-dimensional space, which provides significant computational benefits over using a complete basis, as well as guarding against fitting any noise in the samples.

Another approach to feature selection is to define the problem differently by using the *kernel trick* to replace dot products between features with a kernel function. A kernel is a symmetric function between two points, denoted  $k(s_i, s_j) = k(s_j, s_i)$ , often collected into a kernel matrix  $\mathbf{K}$ , where  $K_{ij} = k(s_i, s_j)$ , and  $\mathbf{k}(x)$  is a column vector with elements  $k_n(x) = k(s_n, \mathbf{s})$ . Note that because the kernel function is symmetric,  $\mathbf{K}$  is also symmetric.

The kernel trick is allowed by Mercer’s theorem, which states that if the kernel is continuous and positive semi-definite, the function can be interpreted as a dot product between the two points in a higher-dimensional space; that is,  $k(s_i, s_j) = \Phi(s_i)^T \Phi(s_j)$ , where the number of features is large (Mercer, 1909). This creates an implicit, highly-expressive approximation space, without having to actually calculate feature values.

In feature selection, we try to define an expressive space by defining useful basis functions; the more we use, the more expressive our approximation can be. In kernelized approaches, we use kernels to gain the expressiveness of a high number of basis functions without actually calculating the functions themselves. Kernel-based

methods are well suited for dealing with large or infinite state spaces due to this high expressiveness. However, the expressiveness of kernels come with a risk of overfitting training data and a risk of heavy computation despite the efficiency of the kernel trick.

## 2.2 Value Function Calculation Algorithms

The purpose of this section is to introduce value function calculation and approximation methods which are not directly related to our own contributions, but are too fundamental to omit. We introduce three methods for calculating a value function, Value Iteration, Policy Iteration, and Linear Programming, as well as their approximate analogues.

### 2.2.1 Value Iteration

Assume  $|\mathcal{S}|$  is small, and that  $R$  and  $P$  are known. In this case, we can perform value iteration to calculate a close approximation to the optimal value function. We start with an arbitrary value function vector  $\hat{V}_0$  and perform the following update:

$$\hat{V}_{k+1} = T\hat{V}_k$$

until  $\|\hat{V}_{k+1} - \hat{V}_k\|_\infty \leq \epsilon$ .

Value iteration makes use of the fact that the Bellman operator is a contraction in max-norm; that is, for any two vectors  $\hat{V}$  and  $\hat{V}'$ ,  $\|T\hat{V} - T\hat{V}'\|_\infty \leq \gamma\|\hat{V} - \hat{V}'\|_\infty$ . If we consider the case where  $\hat{V}' = V^*$ , we see  $\|T\hat{V} - V^*\|_\infty \leq \gamma\|\hat{V} - V^*\|_\infty$ . Note, this is because  $TV^* = V^*$ .

This means value iteration converges to the optimal value function  $V^*$  by a factor of  $\gamma$  each iteration. Value iteration ends when  $\|\hat{V} - T\hat{V}\|_\infty \leq \epsilon$ , where  $\epsilon$  is a stopping parameter. This distance between  $\hat{V}$  and  $T\hat{V}$  is known as the *Bellman error*; this term can also apply to the norm of this vector. The Bellman error of an approxi-

mation  $\hat{V}$ , denoted  $BE(\hat{V})$ , bounds the actual error of the approximation (Williams and Baird, 1993):

$$\|V^* - \hat{V}\|_\infty \leq \frac{\|BE(\hat{V})\|_\infty}{1 - \gamma}.$$

Our reason for assuming  $|\mathcal{S}|$  is small is it quickly becomes unwieldy to apply the Bellman operator for each state. For continuous or infinite state spaces it is impossible to even store the vector  $\hat{V}$ .

If the update above is replaced with

$$\hat{V}_{k+1} = T_\pi \hat{V}_k,$$

for some policy  $\pi$ , value iteration will converge to  $V_\pi$ .

### 2.2.2 Fitted Value Iteration

Fitted Value Iteration (FVI) is a general approach to adapt value iteration to large or infinite state spaces by using approximation architectures. If  $\hat{V}$  is an approximation of  $V^*$  in some approximation architecture (such as linear approximation), and  $\Pi_{\mathcal{F}}$  projects a vector into the approximation's function space, then fitted value iteration performs the following:

$$\hat{V}_{k+1} = \Pi_{\mathcal{F}} T \hat{V}_k.$$

FVI was extended to the approximation of *Q-functions*, in which values are assigned not just to states, but to state-action pairs (Ernst et al., 2006). Approximations on Q-functions can make greedy policy generation simpler, but the addition of action dimensions to the approximation space make approximation more difficult. Fitted Q-Iteration works by performing the following steps:

- For each sample  $\sigma \in \Sigma$ , calculate  $T_{\sigma^a} Q_k(\sigma^s, \sigma^a)$ .
- For each action  $a \in \mathcal{A}$ , regress on the set of samples  $T_{\sigma^a} Q_k(\sigma^s, \sigma^a)$  for which  $\sigma^a = a$  to calculate an approximate function on  $s$ :  $\Pi_{\mathcal{F}} T_a Q_k(s, a)$ .

- $Q_{k+1}(\sigma^s, \sigma^a) = \Pi_{\mathcal{F}} T_a Q_k(\sigma^s, \sigma^a)$
- Repeat until stopping conditions are met.

While FVI does extend value iteration to approximation architectures capable of handling large state spaces, it loses value iteration’s convergence guarantees. This was demonstrated by Boyan and Moore (1995) and Tsitsiklis and Van Roy (1996), who showed that divergence can occur even when  $V^*$  lies in the space defined by  $\Phi$ .

### 2.2.3 Policy Iteration

We note that any value function approximation algorithm which can approximate  $V_\pi$  for a given policy  $\pi$  can be the policy evaluation step of policy iteration. Policy iteration starts by defining an arbitrary policy  $\pi_0$ . It then alternates between two steps, *policy evaluation*, in which  $V_{\pi_i}$  is calculated, and *policy improvement*, in which a new greedy policy  $\pi_{i+1}$  is calculated, where

$$\pi_{i+1}(s) = \operatorname{argmax}_{a \in \mathcal{A}} T_a V_{\pi_i}(s).$$

Policy iteration stops when the policy no longer changes.

If  $V$  is calculated exactly, every iteration of policy iteration can be shown to result in a better policy; with a finite state space, there is a finite number of policies. Therefore, policy iteration must arrive at the optimal policy.

### 2.2.4 Least-Squares Policy Iteration

Least-Squares Policy Iteration (LSPI) was introduced by Lagoudakis and Parr (2003) as a policy iteration algorithm using linear approximation for the value function. As with Fitted Q-Iteration, LSPI approximates Q-functions to ease policy construction, using a different feature set for each possible action. This has the additional benefit of allowing off-policy samples. The weighting of the features is done by LSTDQ,

a Q-function version of Least-Squares Temporal Difference Learning, which will be discussed further in Subsection 3.1.1.

LSPI guarantees convergence to a policy, and guarantees this policy’s performance is close to the performance of an optimal policy.

### 2.2.5 Linear Programming

Finally, we present the linear programming formulation initially presented by d’Epenoux (1963):

$$\begin{aligned} \min_V \quad & \sum_{s \in \mathcal{S}} \rho(s) V(s) \\ \text{s.t.} \quad & T_a V(s) \leq V(s) \quad \forall s \in \mathcal{S}, a \in \mathcal{A}, \end{aligned}$$

where  $\rho$  is a distribution over the initial states; that is  $\sum_{s \in \mathcal{S}} \rho(s) = 1$ . The intuition behind this formulation depends on the fact that the Bellman operator is a contraction and  $TV$  is therefore closer to  $V^*$ . Additionally, because the error is one-sided, that is,  $V(s) \geq TV(s) \forall s \in \mathcal{S}$ , the Bellman operator is also monotonic. Therefore, for some feasible point  $V$ ,  $V \geq TV \geq V^*$ . Solving this LP, therefore, results in the vector  $V$  approaching  $V^*$  from above.

Note that it is not necessary to have the max operator be explicit in the LP formulation; because a constraint exists for every possible action, only the best action will have a tight constraint.

As with value iteration, the LP formulation is only useful if  $|\mathcal{S}|$  is small; a large number of states means a uselessly large number of variables for the solver to handle.

The approximate version of the LP formulation, called Approximate Linear Programming (ALP), was introduced by Schweitzer and Seidmann (1985) and will be introduced in Subsection 6.1.1.

## Linear Value Function Approximation

It has been traditionally believed that there was a tradeoff between model-free and model-based linear value function approximation schemes. The main contribution of this chapter is to demonstrate this line of thinking is misleading; the two approaches actually result in identical approximations. We believe this to be important, as model-based approximation schemes are extremely intuitive and easy to understand. This approachability has value while analyzing the effects of different feature choices. The results in this chapter were independently presented by Schoknecht (2002), Parr et al. (2008) and Sutton et al. (2008), though each paper used this insight to contribute in different ways.

We introduce previous work in two areas in Section 3.1. First, we discuss model-free linear value function approximation schemes which arrive at the linear fixed-point. Next, we discuss feature generation methods and feature selection.

Then, Section 3.2 introduces linear models that approximate the underlying dynamics of the system, and uses those models to produce a novel model-based linear value function approximation scheme. Finally, Section 3.3 demonstrates the value function resulting from the approximate linear model is equal to the solution pro-

duced by the linear fixed-point value function approximators. This result allows us to conclude that in the linear approximation setting, successfully approximating the model explicitly results in successfully approximating the value function.

In this chapter, we are more interested in analyzing the approximation schemes than in the effects of sampling. Therefore, we assume that we know the optimal policy  $\pi^*$  and that  $P_{\pi^*}$  and  $R$  are known and can be represented by a matrix and a vector, respectively.

### 3.1 Previous Work

This section introduces previous work in two areas crucial to the contributions of this chapter. First, Subsection 3.1.1 introduces linear fixed-point methods. Subsection 3.1.2 then introduces methods of generating sets of useful features, which will later be used in experiments and analyzed.

#### 3.1.1 Linear Fixed-Point Methods

The methods presented in this subsection are methods to weight a set of provided features such that the approximation lies at the linear fixed-point. The linear fixed point for a feature set  $\Phi$  is the value of  $\mathbf{w}$  such that

$$\hat{V} = \Phi \mathbf{w}_\Phi = \Pi_\rho(R + \gamma P \Phi \mathbf{w}_\Phi). \quad (3.1)$$

Here,  $\Pi_\rho$  is an operator that is the  $\rho$ -weighted  $L_2$  projection into  $\text{span}(\Phi)$ , and where  $\rho$  is a state weighting distribution, typically the stationary distribution of  $P$ . If  $\varrho = \text{diag}(\rho)$ ,  $\Pi_\rho = \Phi(\Phi^T \varrho \Phi)^{-1} \Phi^T \varrho$ . For convenience, an unweighted projection (uniform  $\rho$ ) or some other  $\rho$  is often used. Our results do not depend upon the projection weights, so we shall assume uniform  $\rho$ . An illustration of the linear fixed point appears in Figure 3.1.

We can use Equation 3.1 and the definition of the projection operator  $\Pi$  to solve

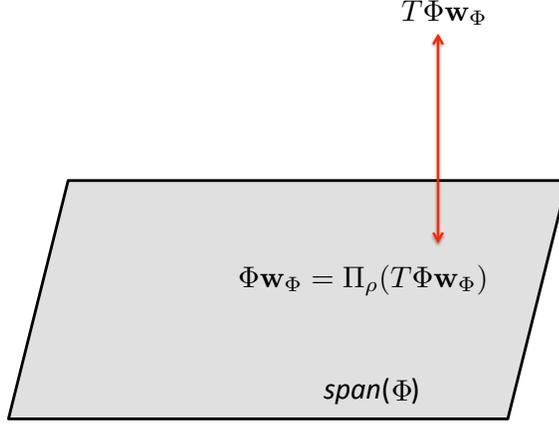


FIGURE 3.1: Illustration of the linear fixed point. Applying the Bellman operator almost certainly removes the approximation from the column space of  $\Phi$ . The point at which the projection back into the span of  $\Phi$  is equal to the point prior to the application of the Bellman operator is the linear fixed-point.

for  $\mathbf{w}_\Phi$  Bradtke and Barto (1996):

$$\begin{aligned} \mathbf{w}_\Phi &= (I - \gamma(\Phi^T \Phi)^{-1} \Phi^T P \Phi)^{-1} (\Phi^T \Phi)^{-1} \Phi^T R \\ &= (\Phi^T \Phi - \gamma \Phi^T P \Phi)^{-1} \Phi^T R. \end{aligned} \quad (3.2)$$

Methods for finding this fixed point given  $\Phi$  defined on a set of samples include linear TD( $\lambda$ ) (Sutton, 1988), LSTD (Bradtke and Barto, 1996) and LSPE (Bertsekas and Ioffe, 1996).

In linear TD( $\lambda$ ), after each sample  $\dot{\sigma}$  is drawn, the weight vector is updated such that

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \left( \dot{\sigma}^r + \gamma \Phi(\dot{\sigma}^{s'}) \mathbf{w} - \Phi(\dot{\sigma}^s) \mathbf{w} \right) \sum_{i=1}^t \lambda^{t-i} \nabla_{\mathbf{w}} \Phi(\dot{\sigma}_i^s) \mathbf{w}. \quad (3.3)$$

Here,  $\alpha$  is a learning rate between 0 and 1,  $\nabla_{\mathbf{w}} \Phi(\dot{\sigma}^s) \mathbf{w}$  is the vector of partial derivatives of  $\Phi(\dot{\sigma}_k^s) \mathbf{w}$  with respect to each element of  $\mathbf{w}$ , and  $\lambda$  is a number between 0 and 1 which weights recent experiences heavier than past experiences. It is common to set  $\lambda = 0$ , so as to use only the most recent experience. Sutton (1988) showed that for absorbing MDPs with linearly independent features, with samples drawn

from a policy  $\pi$ , and for a properly chosen  $\alpha$  which shrinks as data are added, TD(0) converges. Convergence of TD( $\lambda$ ) for general  $\lambda$  and linearly dependent features was proven by Dayan (1992), while Tsitsiklis and Van Roy (1997) showed this convergence was to the linear fixed-point for the Bellman operator  $T_\pi$ .

Unless  $\alpha$  is chosen carefully, TD( $\lambda$ ) is susceptible to oscillation. However, values of  $\alpha$  which guarantee convergence are often extremely small, forcing users of TD( $\lambda$ ) to choose between risking nonconvergent behavior or extremely slow convergence. To address this problem, Bradtke and Barto (1996) introduced Least-Squares Temporal Difference Learning (LSTD). In LSTD, least square function approximation is used to arrive at the following calculation for  $\mathbf{w}$ , which is equivalent to Equation 3.2:

$$\mathbf{w}_t = \left[ \frac{1}{t} \sum_{i=1}^t \Phi(\dot{\sigma}_i^s) \left( \Phi(\dot{\sigma}_i^s) - \gamma \Phi(\dot{\sigma}_i^{s'}) \right)^T \right]^{-1} \left[ \frac{1}{t} \sum_{i=1}^t \Phi(\dot{\sigma}_i^s) \dot{\sigma}_i^r \right]. \quad (3.4)$$

Bradtke and Barto showed that as data are added, the LSTD solution converges to the linear fixed point faster than TD(0), and without the selection of a sensitive design parameter  $\alpha$ . This algorithm and convergence proof was later generalized to LSTD( $\lambda$ ) by Boyan (1999).

LSTD is the policy evaluation step of a notable policy iteration algorithm, Least Squares Policy Iteration (LSPI) Lagoudakis and Parr (2003). The policies generated by LSPI will be a point of comparison for later experiments.

The third linear fixed point algorithm of note is Least-Square Policy Evaluation or LSPE( $\lambda$ ) (Bertsekas and Ioffe, 1996). LSPE assumes a series of  $I$  sample trajectories, where each trajectory is of some length  $J_i$ . The update is then

$$\mathbf{w}_{t+1} = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^I \sum_{j=0}^{J_i} \left[ \Phi(\dot{\sigma}_{i,j}^s) \mathbf{w} - \Phi(\dot{\sigma}_{i,j}^s) \mathbf{w}_t - \sum_{k=j}^{J_i} \lambda^{k-j} d_t(\dot{\sigma}_{i,k}) \right],$$

where

$$d_t(\dot{\sigma}) = \dot{\sigma}^r + \Phi(\dot{\sigma}^{s'}) \mathbf{w}_t - \Phi(\dot{\sigma}^s) \mathbf{w}_t.$$

LSPE( $\lambda$ ) not only converges to the linear fixed-point, but does so at the same rate as LSTD( $\lambda$ ) (Yu and Bertsekas, 2006).

### 3.1.2 Linear Feature Generation

The previous subsection discussed ways of calculating appropriate weights for a given feature set. This subsection discusses methods of constructing that feature set in the first place. One approach to feature generation, presented by Mahadevan and Maggioni (2006), is proto-value functions (PVFs). This method involves first constructing a graph between sampled states in which an edge indicates a transition between states which has either been observed or has been inferred to be possible given the right action. This graph approximates a manifold which attempts to represent the true distances between states more accurately than the ambient space does. Proto-value functions, then, are the eigenvectors of the graph Laplacian, enumerated in increasing order of eigenvalues. These features are added to the feature set in order of smoothness on the manifold approximated by the graph. Proto-value functions are described as “task-independent” basis functions, as they do not consider the reward structure of the problem; the goal of the task can be changed without altering the feature set. Proto-value functions will be explored further in Section 5.1.

Another approach is to construct basis functions using the Bellman error (Wu and Givan, 2004; Sanner and Boutilier, 2005; Parr et al., 2007; Keller et al., 2006). In this framework, features are added iteratively, as feature  $\phi_{k+1}$  is the Bellman error resulting from the approximation using features  $\phi_1 \dots \phi_k$ . Parr et al. (2007) called these Bellman Error Basis Functions (BEBFs) and showed the addition of one more BEBF causes the distance between the approximation and  $V^*$  to improve at least as quickly as the Bellman operator does; in other words, iteratively adding BEBFs causes the bound on the approximation to converge at least as quickly as it does in value iteration. BEBFs were shown to be equivalent to the Krylov basis, which

consists of powers of  $P$  multiplied by  $R$  (Parr et al., 2008). The Krylov basis was first applied to feature selection for value function approximation by Petrik (2007). BEBFs will also be discussed further in section 5.1.

### 3.2 Linear Models

This section introduces linear models and a novel, model-based approach to calculating  $\mathbf{w}$  for a feature set  $\Phi$  based on linear regression. The *model* of an MDP consists of the transition and reward dynamics of the MDP, that is, it consists of  $P$  and  $R$ .

As in the case of linear value functions, we assume the existence of a set of linearly independent features  $\phi_1 \dots \phi_k$  for representing transition and reward models, with  $\phi_i(s)$  defined as the value of feature  $i$  in state  $s$ . While value-function approximation typically uses features to predict values, we will consider the use of these same features to predict rewards and *next* feature values. We begin by discussing feature value prediction.

For feature vector  $\Phi(s) = [\phi_1(s) \dots \phi_k(s)]$ , we define  $\Phi(s'|s)$  as the random vector of next feature values:

$$\Phi(s'|s) \stackrel{s' \sim P(s'|s)}{=} [\phi_1(s'), \dots, \phi_k(s')].$$

Our objective will be to produce a  $k \times k$  matrix  $P_\Phi$  that predicts expected next feature vectors,

$$\Phi(s)P_\Phi \approx E_{s' \sim P(s'|s)}\{\Phi(s'|s)\},$$

and minimizes the expected feature-prediction error:

$$P_\Phi = \operatorname{argmin}_{P_k} \sum_s \|\Phi(s)P_k - E\{\Phi(s'|s)\}\|_2^2. \quad (3.5)$$

(We shall henceforth leave  $s' \sim P(s'|s)$  implicit). One way to solve the minimization problem in Equation 3.5 is to compute the expected next feature values explicitly as

the  $n \times k$  matrix  $P\Phi$  and then find the least-squares solution to the over-constrained system  $\Phi P_\Phi \approx P\Phi$ , since the  $i^{\text{th}}$  row of  $\Phi P_\Phi$  is  $P_\Phi$ 's prediction of the next feature values for state  $i$  and the  $i^{\text{th}}$  row of  $P\Phi$  is the expected value of these features. Using standard linear regression, the least-squares solution is

$$P_\Phi = (\Phi^T \Phi)^{-1} \Phi^T P \Phi, \quad (3.6)$$

with approximate next feature values  $\widehat{P\Phi} = \Phi P_\Phi$ . To predict the reward model using the same features, we could perform a standard least-squares projection into  $\text{span}(\Phi)$  to compute an approximate reward predictor:

$$r_\Phi = (\Phi^T \Phi)^{-1} \Phi^T R, \quad (3.7)$$

with corresponding approximate reward:  $\hat{R} = \Phi r_\Phi$ .

Because the approximate model transforms feature vectors to feature vectors, any  $k$ -vector is a state in the approximate model. If  $x$  is such a state, then in the approximate model,  $r_\Phi^T x$  is the reward for this state and  $P_\Phi^T x$  is the next state vector. The Bellman equation for state  $x$  is:

$$\begin{aligned} V(x) &= r_\Phi^T x + \gamma V(P_\Phi^T x) \\ &= \sum_{i=0}^{\infty} \gamma^i r_\Phi^T (P_\Phi^i)^T x. \end{aligned}$$

Expressed with respect to the original state space, the value function becomes

$$V = \Phi \sum_{i=0}^{\infty} \gamma^i P_\Phi^i r_\Phi,$$

which is obviously a linear combination of the columns of  $\Phi$ . Since  $V = \Phi \mathbf{w}$  for some

$\mathbf{w}$ , the fixed-point equation becomes:

$$\begin{aligned}
 V &= \hat{R} + \gamma \widehat{P\Phi} \mathbf{w} \\
 V &= \Phi r_\Phi + \gamma \Phi P_\Phi \mathbf{w} \\
 \Phi \mathbf{w} &= \Phi r_\Phi + \gamma \Phi P_\Phi \mathbf{w} \\
 \mathbf{w} &= (I - \gamma P_\Phi)^{-1} r_\Phi.
 \end{aligned} \tag{3.8}$$

We call the solution to the system above the *linear model solution*. A solution will exist when  $P_\Phi$  has a spectral radius less than  $1/\gamma$ . This condition is not guaranteed because  $P_\Phi$  is not necessarily a stochastic matrix; it is simply a matrix that predicts expected next feature values. The cases where the spectral radius of  $P_\Phi$  exceeds  $1/\gamma$  correspond to the cases where the value function defined by  $P_\Phi$  and  $r_\Phi$  assigns unbounded value to some states.

### 3.3 Linear Fixed-Point Solution and Linear Model Solution Equivalence

The notion that linear fixed-point methods are implicitly computing some sort of model has been recognized in varying degrees for several years. For example, Boyan (1999) considered the intermediate calculations performed by LSTD in some special cases, and interpreted parts of the LSTD algorithm as computing a compressed model. In this section, we show that the linear fixed-point solution for features  $\Phi$  is *exactly* the solution to the linear model described by  $P_\Phi$  and  $r_\Phi$ . Aside from our paper, this result was shown independently by Schoknecht (2002) and Sutton et al. (2008). These results hold for  $\lambda = 0$ ; they were later extended to  $\lambda > 0$  by Szepesvári (2010). Our results concern unweighted projections, but generalize readily to weighted projections.

We seek to show that for any MRP  $M$  and set of features  $\Phi$ , the linear-model solution and the linear fixed-point solution are identical. We begin with the expression

for the linear-model solution from Equation 3.8 and then proceed by substituting the definitions of  $P_\Phi$  and  $r_\Phi$  from Equation 3.6 and Equation 3.7, yielding:

$$\begin{aligned}
\mathbf{w} &= (I - \gamma P_\Phi)^{-1} r_\Phi \\
&= (I - \gamma(\Phi^T \Phi)^{-1} \Phi^T P \Phi)^{-1} (\Phi^T \Phi)^{-1} \Phi^T R \\
&= (\Phi^T \Phi - \gamma \Phi^T P \Phi)^{-1} \Phi^T R \\
&= \mathbf{w}_\Phi.
\end{aligned} \tag{3.9}$$

Note the final step occurs because step 3.9 is identical to the linear fixed-point solution of Equation 3.2.

This result demonstrates that for a given set of features  $\Phi$ , there is no difference between using the exact model to find an *approximate* linear fixed-point value function in terms of  $\Phi$  and first constructing an approximate linear model in terms of  $\Phi$  and then solving for the *exact* value function of the approximate model using Equation 3.8. Although the model-based view produces exactly the same value function as the model-free view, the model-based view can give a new perspective on error analysis and feature selection, as shown in Chapter 5.

## Kernel-Based Value Function Approximators

In scenarios where more expressiveness is needed from features, many have tried kernel function approximators. As stated in Section 2.1, a *kernel* is a symmetric function between two points, and through the kernel trick, can be used to place the problem in a high-dimensional space without explicitly calculating the features that define that high-dimensional space (Aizerman et al., 1964). This chapter presents techniques for performing kernel-based value function approximation, before demonstrating all presented approaches are equivalent to a novel and intuitive model-based approach. This offers both a unification of the disparate techniques in the field and an intuitive approach to analyzing regularization and kernel selection. This work was first presented by Taylor and Parr (2009).

### 4.1 Previous Work

In this section, we offer an overview of kernelized methods. As an introduction to kernelized methods, we first introduce kernelized regression; regression is intuitive, and will be used in the construction of our model-based approximator. We then introduce previously-presented techniques for using kernels to approximate value func-

tions, namely Kernelized Least-Squares Temporal Difference Learning (KLSTD) (Xu et al., 2005), Gaussian Process Temporal Difference Learning (GPTD) (Engel et al., 2005), and Gaussian Processes in Reinforcement Learning (GPRL) (Rasmussen and Kuss, 2004).

#### 4.1.1 Kernelized Regression

If linear least-squares regression is rederived using the kernel trick to replace dot products between features as described in Subsection 2.1.3, we arrive at the dual form of linear least-squares regression,

$$y(\mathbf{x}) = \mathbf{k}(\mathbf{x})^T \mathbf{K}^{-1} \mathbf{t}, \quad (4.1)$$

where  $\mathbf{t}$  represents the target values of the sampled points,  $\mathbf{K}$  represents the kernel matrix, where  $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$ , and  $\mathbf{k}(\mathbf{x})$  is a column vector with elements  $k_n(\mathbf{x}) = k(\mathbf{x}_n, \mathbf{x})$ .

Because  $\mathbf{K}$  is necessarily  $n \times n$ , where  $n$  is the number of samples in  $\mathbf{t}$ , overfitting is a concern. Therefore, we present the standard regularized form of kernel regression, which again follows easily from applying the kernel trick to regularized linear least-squares regression (Bishop, 2006):

$$y(\mathbf{x}) = \mathbf{k}(\mathbf{x})^T (\mathbf{K} + \lambda \mathbf{I})^{-1} \mathbf{t}, \quad (4.2)$$

where  $\lambda$  is the regularization parameter.

It is worth noting that due to the symmetry of  $\mathbf{K}$ , if  $\mathbf{K}$  is invertible, performing kernel regression is identical to performing linear regression with a feature set  $\Phi$  defined by  $\mathbf{K}$  and weights  $\mathbf{w} = \mathbf{K}^{-1} \mathbf{t}$ , as shown here:

$$\begin{aligned} y(\mathbf{x}) &= \mathbf{k}(\mathbf{x})^T (\mathbf{K}^T \mathbf{K})^{-1} \mathbf{K}^T \mathbf{t} \\ &= \mathbf{k}(\mathbf{x})^T (\mathbf{K} \mathbf{K})^{-1} \mathbf{K} \mathbf{t} \\ &= \mathbf{k}(\mathbf{x})^T \mathbf{K}^{-1} \mathbf{t}. \end{aligned}$$

Another approach to kernelized regression is by using Gaussian Processes (GPs). GPs return not a prediction  $y(\mathbf{x})$ , but a full Gaussian distribution from which to draw a prediction, with mean equal to the kernelized regression solution in Equation 4.2, where  $\lambda$  is the standard deviation of noise in the function. The covariance of this Gaussian process depends upon a prior on the noise in the data, and the variance and density of the data set itself. If training data around the testing point are dense and of low variance, or if the prior indicates a small amount of noise, the variance of the returned Gaussian will be smaller; similarly, if relevant training data are sparse or of high variance, or the prior predicts a large amount of noise, the variance of the returned Gaussian will be larger. As a result, the width of this Gaussian can be interpreted as a measure of uncertainty in the prediction.

#### 4.1.2 Kernelized Value Function Approximation

There have been several kernelized reinforcement learning approaches proposed in recent years, with few obvious similarities in the algorithms. One example is Kernelized Least-Squares Temporal Difference Learning (KLSTD) (Xu et al., 2005). KLSTD begins with the general LSTD( $\lambda$ ) algorithm (Boyan, 1999) and uses the kernel trick discussed in Subsection 2.1.3 to derive a kernelized version of LSTD; every place a dot product between features appears, that dot product is replaced with a kernel matrix. This is equivalent to performing LSTD with a large number of features. We focus on the  $\lambda = 0$  case, for which the approximate value function solution is

$$\hat{V}(s) = \mathbf{k}(s)^T (\mathbf{KHK})^{-1} \mathbf{K}\mathbf{r}, \quad (4.3)$$

where

$$\mathbf{H} = \begin{bmatrix} 1 & -\gamma & 0 & \dots & 0 \\ 0 & 1 & -\gamma & \dots & 0 \\ \vdots & & & & \vdots \\ 0 & 0 & \dots & 1 & -\gamma \\ 0 & 0 & \dots & 0 & 1 \end{bmatrix}.$$

KLSTD was presented without any form of regularization, but the authors did use sparsification, in which data points that were similar to previous data points were removed from the sampled set. This may have helped regularize their solution somewhat and may have also helped improve the conditioning of  $\mathbf{K}$  in their experimental results.

Another example is Gaussian Process Temporal Difference Learning (GPTD) (Engel et al., 2005), which takes a GP approach to direct value function approximation. The GP approach builds a jointly Gaussian model over the training data and implies a predictive distribution over novel states. As with GP regression, the variance of this distribution can be interpreted as a measure of certainty. GPTD begins by modeling the residual:

$$R(\dot{\sigma}^s) = V(\dot{\sigma}^s) - \gamma V(\dot{\sigma}^{s'}) + N(\dot{\sigma}^s, \dot{\sigma}^{s'}).$$

for all samples  $\dot{\sigma} \in \dot{\Sigma}$ .  $N$  is modeled with a Gaussian process,  $N(s, s') \sim \mathcal{N}(0, \Sigma)$ . This formulation results in the approximate value function represented as a Gaussian distribution. The mean of this distribution is

$$\hat{V}(s) = \mathbf{k}(s)^T \mathbf{H}^T (\mathbf{H} \mathbf{K} \mathbf{H}^T + \Sigma)^{-1} \mathbf{r}, \quad (4.4)$$

where  $\mathbf{H}$  is defined as in KLSTD.<sup>1</sup> As with KLSTD, they propose a sparsification technique that approximates the above solution with less computation.

A third approach, Gaussian processes in reinforcement learning (GPRL) (Rasmussen and Kuss, 2004), is a *model-based* approach which first uses Gaussian Processes to approximate the transition and reward models, and then solve for the value function of the approximate transition and reward models. Additionally, kernels are assumed to be the sum of a Gaussian kernel and a weighted delta function; the re-

---

<sup>1</sup> GPTD actually defined  $\mathbf{H}$  without the last row of KLSTD's  $\mathbf{H}$ . The last row corresponds to the assumption that the trajectory ends with a transition to an absorbing state, an assumption we preserve for our version of GPTD for the convenience of having a square  $\mathbf{H}$ .

sulting kernel matrix is denoted  $\mathbf{K}_v$ . Rewards are assumed to be noiseless. This construction makes the value function

$$\hat{V}(s) = \hat{R}(s) + \gamma \int \hat{P}_{s_i, s'} \hat{V}(s') ds'$$

difficult to calculate in closed form, as  $\hat{P}_{s_i, s'}$  and  $\hat{V}(s')$  are both Gaussian. Using a result from Girard et al. (2003), GPRL approximates  $\int \hat{P}_{s_i, s'} V(s') ds'$  with  $\mathbf{W}_i \mathbf{K}_v^{-1} V$ , replacing the integral over the entire state space with a product of  $\mathbf{W}_i$ , which is the expected next kernel values given state  $s_i$ , and the kernelized representation of the value function,  $\mathbf{K}_v^{-1} V$ . The final result is a Gaussian distribution for the value of all sampled points with mean

$$\hat{V} = (\mathbf{I} - \gamma \mathbf{W} \mathbf{K}_v^{-1})^{-1} \mathbf{r},$$

where  $\mathbf{W}$  is a matrix of expected next kernel values with  $\mathbf{W}_{ij} = \mathbb{E}[k(s'_i, s_j)]$ . Because the kernel function consists of a sum,  $\mathbf{K}_v$  can be decomposed such that  $\mathbf{K}_v = \mathbf{K} + \sigma^2 \mathbf{\Delta}$ , where  $\mathbf{K}$  is the kernel matrix resulting from the Gaussian kernel, and  $\mathbf{\Delta}_{ij} = \delta(i, j)$ , producing the value function

$$\hat{V} = (\mathbf{I} - \gamma \mathbf{W} (\mathbf{K} + \sigma^2 \mathbf{\Delta})^{-1})^{-1} \mathbf{r}. \quad (4.5)$$

Each of these approaches are differently motivated and produce seemingly different approximations; two approximate the value function directly, while one looks to first approximate the model, then solve for a value function. However, Section 4.3 will show that they are in fact very similar.

## 4.2 A General Kernelized Model-Based Solution

We now present a general, kernelized approach to model-based RL built upon kernelized regression. Using Equation 4.2 with  $\mathbf{\Sigma} = \mathbf{0}$ , we can formulate our unregularized

*approximate reward model* for state  $s \in \mathcal{S}$  as

$$\hat{R}(s) = \mathbf{k}(s)^T \mathbf{K}^{-1} R, \quad (4.6)$$

and the regularized version:

$$\hat{R}(s) = \mathbf{k}(s)^T (\mathbf{K} + \Sigma_R)^{-1} R. \quad (4.7)$$

The *approximate transition model* is similar. Our approach differs from GPRL in one important way: GPRL learns an approximate model that predicts next state variables given current state variables. Our approximate model does not seek to predict the state itself, but seeks to predict kernel values, i.e.  $\mathbf{k}(s')$  given  $\mathbf{k}(s)$ . This defines the relationship of our predicted next state to our sampled points in the space implicitly defined by the kernel function. We first define the matrix  $\mathbf{K}' = P\mathbf{K}$ , in which  $K'_{ij} = \mathbb{E}[k(x'_i, x_j)]$ . Here,  $\mathbf{K}'$  can be thought of as target data consisting of the vectors  $\mathbf{k}(s')$ . To approximate the transition model, we again use kernelized regression:

$$\hat{\mathbf{k}}(s') = \mathbf{k}(s)^T \mathbf{K}^{-1} \mathbf{K}'. \quad (4.8)$$

As with the reward model, we can construct a regularized version of the approximate transition model,

$$\hat{\mathbf{k}}(s') = \mathbf{k}(s)^T (\mathbf{K} + \Sigma_P)^{-1} \mathbf{K}' \quad (4.9)$$

We can use the models expressed in Equations 4.6 and 4.8 to construct an un-

regularized approximate value function.

$$\begin{aligned}
\hat{V}(s) &= \mathbf{k}(s)^T \mathbf{K}^{-1} R + \gamma \underbrace{\mathbf{k}(s)^T \mathbf{K}^{-1} \mathbf{K}' \mathbf{K}^{-1} R}_{\hat{R}(s')} + \gamma^2 \mathbf{k}(s)^T (\mathbf{K}^{-1} \mathbf{K}')^2 \mathbf{K}^{-1} R + \dots \\
&= \mathbf{k}(s)^T \sum_{i=0}^{\infty} \left[ \gamma^i (\mathbf{K}^{-1} \mathbf{K}')^i \right] \mathbf{K}^{-1} R \\
&= \mathbf{k}(s)^T (\mathbf{I} - \gamma \mathbf{K}^{-1} \mathbf{K}')^{-1} \mathbf{K}^{-1} R
\end{aligned}$$

Distributing  $\mathbf{K}^{-1}$ , we arrive at our final value function,

$$V(s) = \mathbf{k}(s)^T (\mathbf{K} - \gamma \mathbf{K}')^{-1} R. \quad (4.10)$$

It is also possible to perform this derivation using the regularized reward and model approximations from Equations 4.7 and 4.9, resulting in the following value function:

$$\hat{V}(s) = \mathbf{k}(s)^T [(\mathbf{K} + \Sigma_R) - \gamma (\mathbf{K} + \Sigma_R) (\mathbf{K} + \Sigma_P)^{-1} \mathbf{K}']^{-1} R \quad (4.11)$$

Regularization is often necessary, as real RL problems exhibit noise and the high expressiveness of the kernel matrix can result in overfitting. Also, there is no guarantee the kernel matrix  $\mathbf{K}$  will be invertible and regularization tends to improve the conditioning of the matrix inversion problem. A benefit of the model-based solution presented here is that it offers the ability to regularize reward and transition approximations separately.

So far, our derivation has assumed that  $\mathbf{K}'$  could be computed and represented explicitly for the entire state space. In practice one would typically apply kernel-based approximation algorithms to large or continuous state spaces. In these cases, it is impossible to create a kernel matrix  $\mathbf{K}$  representing the kernel functions between

every state. Instead, we use samples  $\dot{\Sigma}$ , and construct a *sampled*  $\mathbf{K}$ ,  $\mathbf{K}'$ , and  $\mathbf{r}$ . In this case,  $k(s_i, s_j) = k(\dot{\sigma}_i^s, \dot{\sigma}_j^s)$ , where  $\dot{\sigma}_i, \dot{\sigma}_j \in \dot{\Sigma}$ , and  $\mathbf{K}'_{ij} = k(\dot{\sigma}_i^{s'}, \dot{\sigma}_j^s)$ . In the special case where the samples are drawn from trajectories,  $\dot{\sigma}_i^{s'} = \dot{\sigma}_{i+1}^s$ , and  $k(\dot{\sigma}_i^{s'}, \dot{\sigma}_j^s) = k(\dot{\sigma}_{i+1}^s, \dot{\sigma}_j^s)$ . Therefore,  $\mathbf{K}' = \mathbf{G}\mathbf{K}$ , where

$$\mathbf{G} = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & & & & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ 0 & 0 & 0 & \dots & 0 \end{bmatrix}.$$

### 4.3 Equivalence

With the model-based value functions defined, we are ready to state our equivalence theorems.

**Theorem 4.3.1.** *Given the same trajectories and same kernels, the KLSTD value function is equivalent to the unregularized model-based value function.*

*Proof.* Note  $\mathbf{H} = \mathbf{I} - \gamma\mathbf{G}$ . Our first step is to show that  $\mathbf{K}\mathbf{H}\mathbf{K} = \mathbf{K}\mathbf{K} - \gamma\mathbf{K}\mathbf{K}'$ , using  $\mathbf{H} = \mathbf{I} - \gamma\mathbf{G}$  and  $\mathbf{K}' = \mathbf{G}\mathbf{K}$ :

$$\begin{aligned} \mathbf{K}\mathbf{H}\mathbf{K} &= \mathbf{K}(\mathbf{I} - \gamma\mathbf{G})\mathbf{K} \\ &= \mathbf{K}\mathbf{K} - \gamma\mathbf{K}\mathbf{G}\mathbf{K} \\ &= \mathbf{K}\mathbf{K} - \gamma\mathbf{K}\mathbf{K}'. \end{aligned}$$

Starting from the solution to KLSTD in Equation 4.3,

$$\begin{aligned} V(s) &= \mathbf{k}(s)^T (\mathbf{K}\mathbf{H}\mathbf{K})^{-1} \mathbf{K}\mathbf{r} \\ &= \mathbf{k}(s)^T (\mathbf{K}\mathbf{K} - \gamma\mathbf{K}\mathbf{K}')^{-1} \mathbf{K}\mathbf{r} \\ &= \mathbf{k}(s)^T (\mathbf{K}^{-1}\mathbf{K}\mathbf{K} - \gamma\mathbf{K}^{-1}\mathbf{K}\mathbf{K}')^{-1} \mathbf{r} \\ &= \mathbf{k}(s)^T (\mathbf{K} - \gamma\mathbf{K}')^{-1} \mathbf{r}, \end{aligned}$$

which is equal to our unregularized approximate model-based value function in Equation 4.10.  $\square$

The implication of this theorem is that we can view KLSTD as implicitly approximating the underlying transition and reward functions of the system. We can prove a similar theorem about the relationship of GPTD to the model-based approximation.

**Theorem 4.3.2.** *Given the same trajectories and same kernels, the mean value function returned by GPTD is equivalent to the regularized model-based value function with  $\Sigma_R = \Sigma_P = \Sigma(\mathbf{H}^T)^{-1}$ .*

*Proof.* We begin with the GPTD mean approximate value function introduced in Equation 4.4, and show it is equivalent to Equation 4.11:

$$\begin{aligned} \hat{V}(s) &= \mathbf{k}(s)^T \mathbf{H}^T (\mathbf{H} \mathbf{K} \mathbf{H}^T + \Sigma)^{-1} \mathbf{r} \\ &= \mathbf{k}(s)^T (\mathbf{H} \mathbf{K} + \Sigma (\mathbf{H}^T)^{-1})^{-1} \mathbf{r} \\ &= \mathbf{k}(s)^T (\mathbf{K} - \gamma \mathbf{K}' + \Sigma (\mathbf{H}^T)^{-1})^{-1} \mathbf{r} \\ &= \mathbf{k}(s)^T [(\mathbf{K} + \Sigma_R) - \gamma (\mathbf{K} + \Sigma_R) (\mathbf{K} + \Sigma_P)^{-1} \mathbf{K}']^{-1} \mathbf{r}, \end{aligned}$$

when  $\Sigma_R = \Sigma_P = \Sigma(\mathbf{H}^T)^{-1}$ .  $\square$

In the noiseless case when  $\Sigma = \mathbf{0}$ , GPTD is equivalent to the unregularized model-based value function in Equation 4.10.

This theorem assumes that  $(\mathbf{H}^T)^{-1}$  exists, but this is ensured by the structure of  $\mathbf{H}$ . As with KLSTD, we see that GPTD can be viewed as implicitly approximating the underlying transition and reward models of the system. It is not surprising that GPTD's  $\Sigma$  appears in both the transition model and reward regularization since GPTD does not have separate noise terms for the reward and transition.

The appearance of  $(\mathbf{H}^T)^{-1}$  may be somewhat surprising. Loosely speaking, we can say that it propagates the regularizer  $\Sigma$  through the transition model since:

$$(\mathbf{H}^T)^{-1} = (I - \gamma P^T)^{-1} = \sum_{i=1}^{\infty} \gamma^i (P^T)^i,$$

but we believe that empirical study of the role of  $(\mathbf{H}^T)^{-1}$  is warranted.

In contrast, GPRL explicitly models the transition model of the system, but uses a very different starting point from our model-based approximation. However, we can show the final results are still closely related.

**Theorem 4.3.3.** *Using the same sample data and same kernel functions, the mean value function returned by GPRL is equivalent to the regularized model-based value function with  $\Sigma_R = \Sigma_P = \sigma^2 \Delta$ .*

*Proof.* Recall that  $\mathbf{W}_{ij} = \mathbb{E}[k(x'_i, x_j)]$ . GPRL's regularization term is part of the definition of the kernel, but GPRL uses a trick to ensure that regularization is applied only to the training data and not to test data: the regularizer is multiplied by a delta function which ensures that it is applied only to on-diagonal entries in  $\mathbf{K}$ . Since GPRL assumes that data are drawn from a Gaussian,  $P(\delta(x'_i, x_j) > 0) = 0$ , and  $\sigma^2$  is not expected to appear in  $\mathbf{K}'$ . We therefore assume  $\mathbf{W}_{ij} = \mathbf{K}'_{ij}$ . Beginning with the GPRL value function in Equation 4.5:

$$\begin{aligned} \hat{V} &= (\mathbf{I} - \gamma \mathbf{W}(\mathbf{K} + \sigma^2 \Delta)^{-1})^{-1} \mathbf{r} \\ &= (\mathbf{I} - \gamma \mathbf{K}'(\mathbf{K} + \sigma^2 \Delta)^{-1})^{-1} \mathbf{r} \\ &= (\mathbf{K} + \sigma^2 \Delta) ((\mathbf{K} + \sigma^2 \Delta) - \gamma \mathbf{K}'(\mathbf{K} + \sigma^2 \Delta)^{-1}(\mathbf{K} + \sigma^2 \Delta))^{-1} \mathbf{r} \\ &= (\mathbf{K} + \sigma^2 \Delta) (\mathbf{K} + \sigma^2 \Delta - \gamma \mathbf{K}')^{-1} \mathbf{r}. \end{aligned}$$

To change this value function from one that defines approximate values over all experienced points to one that defines a value function on an arbitrary point  $s$ , we

replace  $\mathbf{K} + \sigma^2 \mathbf{\Delta}$  with the kernel evaluated at  $s$ :

$$\hat{V}(s) = (\mathbf{k}(s)^T + \sigma^2 \boldsymbol{\delta}(s)^T) (\mathbf{K} + \sigma^2 \mathbf{\Delta} - \gamma \mathbf{K}')^{-1} \mathbf{r},$$

where  $\boldsymbol{\delta}(s)$  is a column vector with elements  $\delta_i(s) = \delta(s_i, s)$ . For an arbitrary point in a continuous space,  $\boldsymbol{\delta}(s)$  will be the zero vector, so

$$\begin{aligned} \hat{V}(s) &= \mathbf{k}(s) (\mathbf{K} + \sigma^2 \mathbf{\Delta} - \gamma \mathbf{K}')^{-1} \mathbf{r} \\ &= \mathbf{k}(s)^T [(\mathbf{K} + \boldsymbol{\Sigma}_R) - \gamma (\mathbf{K} + \boldsymbol{\Sigma}_R) (\mathbf{K} + \boldsymbol{\Sigma}_P)^{-1} \mathbf{K}']^{-1} \mathbf{r}, \end{aligned}$$

when  $\boldsymbol{\Sigma}_R = \boldsymbol{\Sigma}_P = \sigma^2 \mathbf{\Delta}$ . □

When  $\sigma^2 = 0$ , this is equivalent to the unregularized model-based value function in Equation 4.10.

In summary, KLSTD, GPTD, GPRL, and our novel model-based value function differ only in their approaches to regularization and their assumptions about the manner in which the samples are drawn; these relationships are tabulated in Table 4.1 below. Even though these algorithms are basically identical, it can nevertheless be useful to consider different approaches to regularization to get insight into parameter selection since overfitting with kernel methods is a genuine concern. This section also shows that even seemingly direct approaches to kernelized value function approximation have a model-based interpretation. In Section 5.2, we show that the model-based interpretation lends itself to a useful error decomposition.

Method	Value Function	Model-Based Equivalent
KLSTD	$\mathbf{w} = (\mathbf{KHK})^{-1} \mathbf{K} \mathbf{r}$	$\boldsymbol{\Sigma}_P = \boldsymbol{\Sigma}_R = \mathbf{0}$
GPTD	$\mathbf{w} = \mathbf{H}^T (\mathbf{HKH}^T + \boldsymbol{\Sigma})^{-1} \mathbf{r}$	$\boldsymbol{\Sigma}_P = \boldsymbol{\Sigma}_R = \boldsymbol{\Sigma} (\mathbf{H}^T)^{-1}$
GPRL	$\mathbf{w} = (\mathbf{K} + \sigma^2 \mathbf{\Delta} - \gamma \mathbf{K}')^{-1} \mathbf{r}$	$\boldsymbol{\Sigma}_P = \boldsymbol{\Sigma}_R = \sigma^2 \mathbf{\Delta}$
Model-Based	$\mathbf{w} = [(\mathbf{K} + \boldsymbol{\Sigma}_R) + \gamma (\mathbf{K} + \boldsymbol{\Sigma}_R) (\mathbf{K} + \boldsymbol{\Sigma}_P)^{-1} \mathbf{K}']^{-1} \mathbf{r}$	

Table 4.1: Previously introduced methods of kernelized value-function approximation are equivalent to the novel model-based approximation

## Analysis of Error

Chapters 3 and 4 demonstrated that many existing linear and kernel-based approximation schemes were equivalent to novel model-based approaches; it is clear that the better an approach approximates the reward and transition dynamics of the system, the better the value function approximation is likely to be. This chapter formalizes this intuition in order to aid with feature selection and regularization tuning.

We show that for fixed-point solutions, the Bellman error can be decomposed into *reward error* and *transition error*. Reward error expresses how well the approximation represents the reward function, while transition error expresses how well the approximation represents the transition function of the system. This view can provide insight into the performance of the approximation algorithm.

### 5.1 Error in Linear Value Function Approximations

Error in value function approximations can come from two sources. First, the feature set institutes bias; second, noise in sampling causes variance. For the duration of this section, we will assume that  $\Phi$  can be constructed with a row for every  $s \in \mathcal{S}$ , and that there is no noise in our samples. This assumption allows us to characterize

the representational power of the features as a separate issue from the variance introduced by sampling.

In the context of linear value functions and linear models, we shall define the Bellman error for a set  $\Phi$  of features as the error in the linear fixed-point value function for  $\Phi$ :

$$BE(\Phi) = BE(\Phi\mathbf{w}_\Phi) = R + \gamma P\Phi\mathbf{w}_\Phi - \Phi\mathbf{w}_\Phi.$$

To understand the relationship between the error in the linear model and the Bellman error, we define two components of the model error, the *reward error*:

$$\Delta_R = R - \hat{R},$$

and the *transition error*:

$$\Delta_\Phi = P\Phi - \widehat{P\Phi}.$$

The Bellman error can be related to these model errors with the following theorem.

**Theorem 5.1.1.** *For any MRP  $M$  and features  $\Phi$ ,*

$$BE(\Phi) = \Delta_R + \gamma\Delta_\Phi\mathbf{w}_\Phi. \tag{5.1}$$

*Proof.* Using the definitions of  $BE(\Phi)$ ,  $\Delta_R$ , and  $\Delta_\Phi$ :

$$\begin{aligned} BE(\Phi) &= R + \gamma P\Phi\mathbf{w}_\Phi - \Phi\mathbf{w}_\Phi \\ &= (\Delta_R + \hat{R}) + \gamma(\Delta_\Phi + \widehat{P\Phi})\mathbf{w}_\Phi - \Phi\mathbf{w}_\Phi \\ &= (\Delta_R + \gamma\Delta_\Phi\mathbf{w}_\Phi) + \hat{R} + (\gamma\Phi P_\Phi - \Phi)\mathbf{w}_\Phi \\ &= (\Delta_R + \gamma\Delta_\Phi\mathbf{w}_\Phi) + \hat{R} - \Phi(I - \gamma P_\Phi)\mathbf{w}_\Phi \\ &= (\Delta_R + \gamma\Delta_\Phi\mathbf{w}_\Phi) + \hat{R} - \Phi r_\Phi \\ &= \Delta_R + \gamma\Delta_\Phi\mathbf{w}_\Phi. \end{aligned}$$

The final step follows from the definition of  $\hat{R}$ , and the penultimate step follows from Equation 3.8. □

In Chapter 3, we discussed that the model-based and model-free equivalence meant every fixed-point method was actually approximating the model; this theorem formalizes that intuition. If a feature set’s reward and transition errors are both low, then the Bellman error must also be low. More pessimistically, it means that if a feature set is unable to capture the structure of the reward function or predict expected next features, then this may result in a poor value function. In Subsection 5.1.1 we show this view of the Bellman error can give insight into the problem of feature selection; in particular, we demonstrate that several methods fail to predict the reward function, resulting in high reward error and high Bellman error. This finding is consistent with the observation of Petrik (2007) of the problems that arise when the reward is orthogonal to the features.

We caution that there can be interactions between  $\Delta_R$  and  $\Delta_\Phi$ . For example, consider the basis composed of the single basis function  $\phi^* = V^*$ . Clearly,  $BE(\phi^*) = 0$ , but for any non-trivial problem and approximate model,  $\Delta_R$  and  $\Delta_\Phi$  will be nonzero and will cancel each other out in Equation 5.1.

### 5.1.1 *Experimental Results*

We present policy-evaluation results using linear value function approximation on three different problems, a 50-state chain, a two-room problem, and blackjack. Our objective is to demonstrate how our theoretical results can inform the feature-generation process and explain the behavior of known feature-generation algorithms. We consider four such algorithms:

*PVF*: This is the proto-value function (PVF) framework described by Mahadevan and Maggioni (2006). PVFs were introduced in Subsection 3.1.2. We reproduced their method as closely as possible, including adding links to the adjacency matrix for all policies, not just the policy under evaluation. Curiously, removing the off-policy

links seemed to produce worse performance. We avoided using samples to eliminate the confounding (for our purposes) issue of variance between experiments. Because it offered better performance, we used the combinatorial Laplacian for the 50-state and blackjack problems, but used the normalized Laplacian in the two-room problem to match Mahadevan and Maggioni (2006).

*PVF-MP:* This algorithm selects basis functions from the set of PVFs, but selects them incrementally based upon the Bellman error. Specifically, basis function  $k+1$  is the PVF that has highest dot product with the Bellman error resulting from the previous  $k$  basis functions. It can be interpreted as a form of matching pursuits (Mallat and Zhang, 1993) on the Bellman error with a dictionary of PVFs.

*Eig-MP:* The form of the Bellman error in Equation 5.1 suggests that features for which  $\Delta_\Phi = 0$  are particularly interesting. If a dictionary of such features were available, then the feature-selection problem would reduce to the problem of predicting the immediate reward using this dictionary. The condition  $\Delta_\Phi = 0$  means that, collectively, the features are a basis for a perfect linear predictor of their own next, expected values. More formally, feature  $\Phi$  are *subspace invariant* with respect to  $P$  if  $P\Phi$  is in  $\text{span}(\Phi)$ , which means there exists a  $\Lambda$  such that  $P\Phi = \Phi\Lambda$ .

It may seem like subspace invariance is an unreasonable requirement that could hold only for a complete basis of  $P$ . However, it turns out many such subspaces exist, including any set of eigenvectors of  $P$ . For eigenvectors  $\chi_1 \dots \chi_k$  with eigenvalues  $\lambda_1 \dots \lambda_k$ ,  $\Lambda = \text{diag}(\lambda_1 \dots \lambda_k)$ . Other subspaces are described by Parr et al. (2008), but are not germane to this discussion.

The Eig-MP approach, therefore, is similar to PVF-MP, but selects from a dictionary of the eigenvectors of  $P$ .

The Eig-MP approach is similar to one proposed by Petrik (2007), in which the

eigenvectors of  $P$  were added to the basis in order of decreasing eigenvalue.

*BEBF*: This is the Bellman Error Basis Function (BEBF) algorithm described by Parr et al. (2007), and introduced in Subsection 3.1.2. In this scheme, the first feature is equal to the reward function of the space. From this point, feature  $k + 1$  is the Bellman Error resulting from the previous  $k$  features.

Our experiments performed unweighted  $L_2$  projection and report unweighted  $L_2$  norm error. We also considered  $L_\infty$  error and  $L_2$  projections weighted by stationary distributions, but the results were not qualitatively different. We report the Bellman error, the reward error, and the *feature error*, which is the contribution of the transition error to the Bellman error:  $\gamma\Delta_\Phi\mathbf{w}_\Phi$ . These metrics are presented as a function of the number of basis functions.

These algorithms were applied to three different problems:

*50-state Chain* We applied all four algorithms to the 50-state chain problem from Lagoudakis and Parr (2003), with the results shown in Figure 5.2(a–c). This problem consists of 50 states, with reward given in states 10 and 41. The actions were a choice of moving left or right; the chosen action succeeded 90% of the time, but moved the agent in the opposite direction 10% of the time. The policy was the optimal policy of moving towards the nearest reward. The discount factor was  $\gamma = .9$ .

As demanded by the property of subspace invariance, Eig-MP has 0 feature error, which means that the entirety of the Bellman error is expressed in  $\Delta_R$ . This remaining error, however, is substantial. For this problem, PVFs appear to be approximately subspace invariant, resulting in low  $\Delta_\Phi$ . Because PVFs are defined entirely on transition data, this makes intuitive sense. However, both Eig-MP and the PVF methods do poorly because the reward is not easily expressed as linear

combination of a small number of features. This illustrates that targeting only one of the two sources of error may not succeed.

BEBFs represent the other extreme since  $\Delta_R = 0$  after the first basis function (which, after all, is  $R$  itself) is added and the entirety of the Bellman error is expressed through  $\Delta_\Phi$ . PVF-MP does better than plain PVFs because it is actively trying to reduce the error, while plain PVFs choose basis functions in an order that ignores the reward. In fact, PVF-MP accepts substantial feature error to more quickly minimize reward error.

*Two-room Problem* We tried all four algorithms on an optimal policy for the two-room navigation problem from Mahadevan and Maggioni (2006), illustrated in Figure 5.1. It consists of a discrete 20-by-20 grid bisected by a wall with a small door in the middle. This complicates the problem, as it is possible for two points on opposite sides of the wall to be very near each other in the state space, but have very different values and not be able to transition to each other. The policy was an optimal policy, and the discount factor was  $\gamma = .9$ .

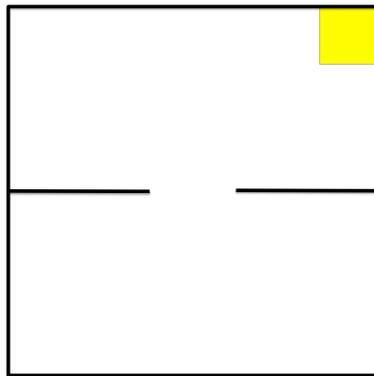


FIGURE 5.1: An illustration of the two-room problem from Mahadevan and Maggioni (2006). The yellow state in the upper-right corner provides a reward.

For this problem, we do not show results for the Eig-MP method; the transi-

tion matrix for this problem is not diagonalizable, and so its eigenvectors do not make a complete basis. In theory, the existing eigenvectors can be augmented with generalized eigenvectors, but methods for calculating these proved unreliable.

Figure 5.2(d–f) shows the breakdown of error for the remaining algorithms. In this case, the Laplacian approach produces features that behave less like an invariant subspace, resulting in high  $\Delta_R$  and  $\Delta_\Phi$ . However, there is some cancellation between them.

*Blackjack* We tested a version of the bottomless-deck blackjack problem from Sutton and Barto (1998), evaluating the policy they propose. In this game, the common rules of blackjack apply, but the dealer has a fixed policy. If the dealer has a 16 or lower, he hits; otherwise, he stays. The state space consists of the player’s cards, the dealer’s visible card, and whether or not he holds an ace which is usable. A usable ace is one which can be counted as 11 without going bust. The rewards of 1, -1, or 0 are given if the player wins, loses, or draws. The policy we evaluate is that of the player sticking if he has a 20 or 21, and hitting otherwise.

For the model described in the book, all methods except BEBF performed extremely poorly. To make the problem more amenable to eigenvector-based methods, we implemented an ergodic version that resets to an initial distribution over hands with a value of 12 or larger and used a discount of 0.999. The breakdown of error for the different algorithms is shown in Figure 5.2(g–i), where we again omit Eig-MP. As expected, BEBFs exhibit  $\Delta_R = 0$ , and drive the Bellman error down fairly rapidly. PVFs exhibit some interesting behaviors: when the PVFs are enumerated in order of increasing eigenvalue, they form an invariant subspace. As a result, the feature error for PVFs hugs the abscissa in Figure 5.2(i). However, this ordering completely fails to match  $R$  until the very last eigenvectors are added, resulting in very poor performance overall. In contrast, PVF-MP adds basis eigenvectors in an order that

does not result in subspace invariant features sets, but that does match  $R$  earlier, resulting in a more consistent reduction of error.

These examples demonstrate the benefit of the Bellman error decomposition; without the decomposition, it would have been challenging to explain why individual feature-selection algorithms succeeded or failed. The implicit model approximations provide a starting point for algorithm improvement.

## 5.2 Error in Kernel-Based Value Function Approximation

In this section, we perform a similar breakdown of the Bellman error. However, the difference between linear and kernelized approximations lead to this decomposition having a different use in the kernelized case. In the linear case, we worried about having sufficient expressiveness; we sought the very few features which were sufficient to approximate both components of the model. In the kernelized case, we have the opposite problem; kernelized methods are extremely expressive and tend to overfit, leading to a need for regularization. In the following subsection, we demonstrate that this breakdown can assist in the tuning of regularization parameters.

As in Section 5.1, we define *reward error* and *transition error*. First, however, we must elaborate on our notation, and introduce some new terms. Our goal is to represent an entire state space containing  $|\mathcal{S}|$  states with  $n$  sampled points, where  $n \ll |\mathcal{S}|$ . Therefore,  $\mathbf{r}$  is an  $n$ -vector, and  $\hat{P}$  and  $\mathbf{K}$  are  $n \times n$  matrices, where  $\hat{P}$  is our estimation of  $P$ , such that  $\hat{P}\mathbf{K} \approx \mathbf{K}'$ . Let  $\mathcal{K}$  be a  $|\mathcal{S}| \times n$  matrix, where  $\mathcal{K}_{ij}$  is equal to  $k(s_i, s_j)$ , where  $s_i$  is an element of the set of all states in the state space, and  $s_j$  is one of our experienced points. Therefore, our approximate reward values for the entirety of the state space follows from Equation 4.6, and is represented as

$$\hat{R} = \mathcal{K}\mathbf{K}^{-1}\mathbf{r}. \quad (5.2)$$

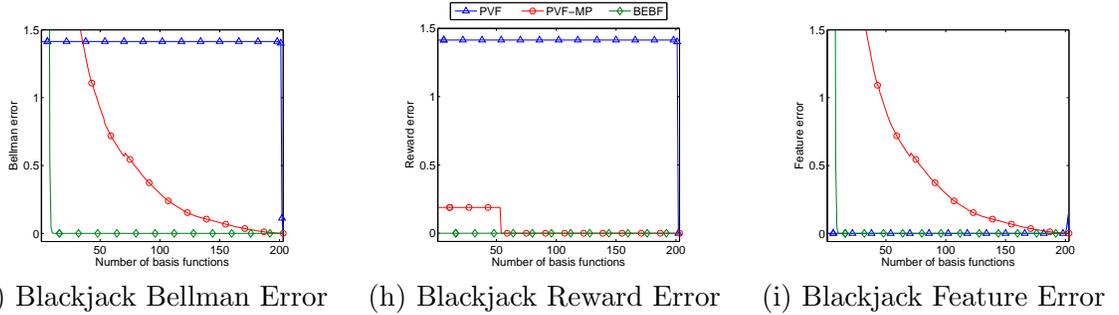
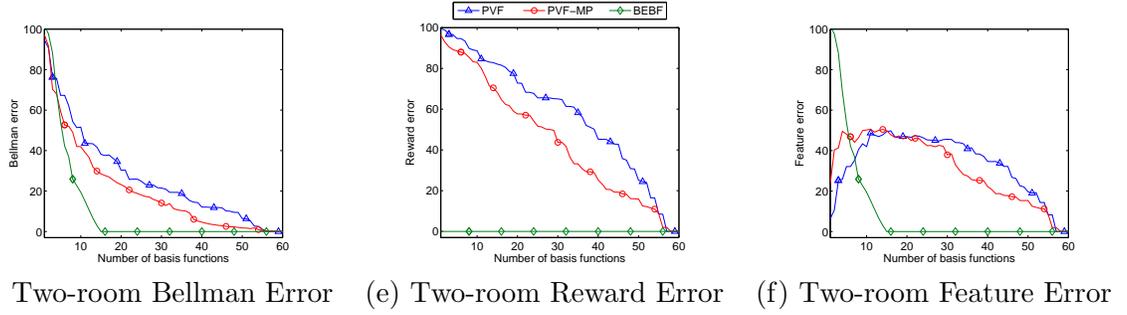
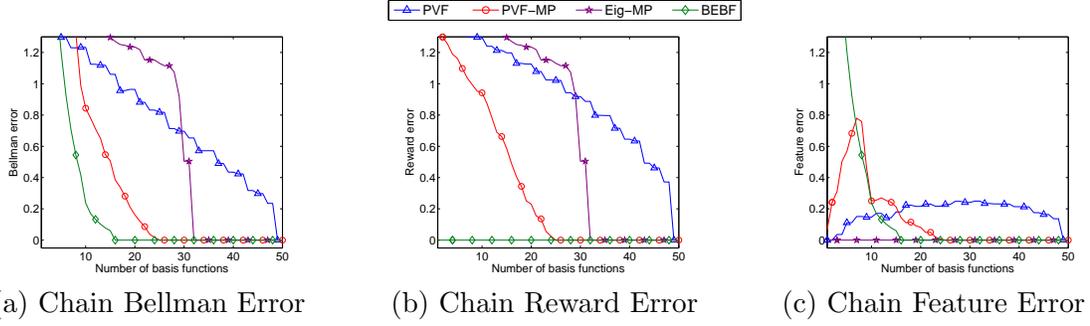


FIGURE 5.2: Decomposition of the Bellman error for three different problems. First row: 50-state chain; Second row: Two-room problem; Third row: Ergodic Blackjack. First column: Bellman error; Second Column: reward error; Third Column: feature error

We can also introduce *reward error* in the kernel context as a vector indicating the difference between the actual reward values, and our approximation  $\hat{R}$ .

$$\Delta_R = R - \hat{R} \tag{5.3}$$

Similarly, we introduce *transition error*. In the kernel context, this is a measure

of our error in predicting  $\mathbf{k}(s')$ .

$$\Delta_{\mathbf{K}'} = P\mathcal{K} - \mathcal{K}\mathbf{K}^{-1}\hat{P}\mathbf{K} \quad (5.4)$$

For readability, we define  $\mathbf{w} = (\mathbf{K} - \gamma P\mathbf{K})^{-1} \mathbf{r}$ , which is extracted from our value function defined in Equation 4.10.

**Theorem 5.2.1.** *For any MRP  $M$  and kernel matrix  $\mathbf{K}$ ,*

$$BE(\mathbf{K}) = \Delta_R + \gamma\Delta_{\mathbf{K}'}\mathbf{w}. \quad (5.5)$$

*Proof.* Our proof closely follows the proof of Theorem 5.1.1. We again begin with the definition of the Bellman error, and use our definitions of  $\Delta_R$  and  $\Delta_{\mathbf{K}'}$  from Equations 5.3 and 5.4.

$$\begin{aligned} BE(\mathbf{K}) &= R + \gamma V(s') - V(s) \\ &= R + \gamma P\mathcal{K}\mathbf{w} - \mathcal{K}\mathbf{w} \\ &= \Delta_R + \hat{R} + \gamma (\Delta_{\mathbf{K}'} + \mathcal{K}\mathbf{K}^{-1}P\mathbf{K}) \mathbf{w} - \mathcal{K}\mathbf{w} \\ &= \Delta_R + \hat{R} + \gamma\Delta_{\mathbf{K}'}\mathbf{w} + \gamma\mathcal{K}\mathbf{K}^{-1}P\mathbf{K}\mathbf{w} - \mathcal{K}\mathbf{w} \\ &= \Delta_R + \hat{R} + \gamma\Delta_{\mathbf{K}'}\mathbf{w} + \mathcal{K} (\gamma\mathbf{K}^{-1}P\mathbf{K} - \mathbf{I}) \mathbf{w} \\ &= \Delta_R + \hat{R} + \gamma\Delta_{\mathbf{K}'}\mathbf{w} + \mathcal{K}\mathbf{K}^{-1} (\gamma P\mathbf{K} - \mathbf{K}) \mathbf{w} \\ &= \Delta_R + \hat{R} + \gamma\Delta_{\mathbf{K}'}\mathbf{w} - \mathcal{K}\mathbf{K}^{-1} (\mathbf{K} - \gamma P\mathbf{K}) \mathbf{w} \\ &= \Delta_R + \hat{R} + \gamma\Delta_{\mathbf{K}'}\mathbf{w} - \mathcal{K}\mathbf{K}^{-1} (\mathbf{K} - \gamma P\mathbf{K}) (\mathbf{K} - \gamma P\mathbf{K})^{-1} \mathbf{r} \\ &= \Delta_R + \hat{R} + \gamma\Delta_{\mathbf{K}'}\mathbf{w} - \mathcal{K}\mathbf{K}^{-1} \mathbf{r} \\ &= \Delta_R + \hat{R} + \gamma\Delta_{\mathbf{K}'}\mathbf{w} - \hat{R} \\ &= \Delta_R + \gamma\Delta_{\mathbf{K}'}\mathbf{w}. \end{aligned}$$

□

This result means that just as in the linear model case, when using kernel-based methods we can view the Bellman Error as having two direct sources of error, the reward error and transition error. We can use this information to analyze the behavior of value-function approximation algorithms.

### 5.2.1 *Experimental Results*

In Section 4.3, we showed several kernel-based value function approximators to be equivalent, except for their treatment of regularization. The Bellman error decomposition provides insight into the behavior of regularization and can facilitate the selection of regularizing parameters.

To demonstrate this use of the Bellman error decomposition, we consider a continuous, two-room maze navigation problem similar to problems explored by Mahadevan and Maggioni (2006) and Engel et al. (2005). A vertical wall separates the state space into equal-sized left and right rooms. A small opening in the middle of this wall acts as a passage between the rooms. An agent can be at any empty point. The actions are 1-unit long steps in one of the four cardinal directions, with added two-dimensional Gaussian noise with covariance  $0.5\mathbf{I}$ . The agent cannot travel through walls, so movements that would cross walls are truncated at the point of intersection. When the agent reaches a 1-unit-wide goal region along the entire right wall of the right room, it deterministically receives a reward of 1. In our version of the problem, the agent can loiter in the reward region, making the maximum achievable value  $1/(1 - \gamma)$ . The discount factor was  $\gamma = 0.9$ . The domain is illustrated in Figure 5.3(a), and the optimal value function for an optimal policy is shown in Figure 5.3(b).

Our training set  $\dot{\Sigma}$  consisted of 3750 samples drawn uniformly from the space. The action used at each state was from an optimal policy for this problem. We used a Gaussian kernel with a diagonal covariance matrix  $3\mathbf{I}$ , and varied the  $\Sigma_P$

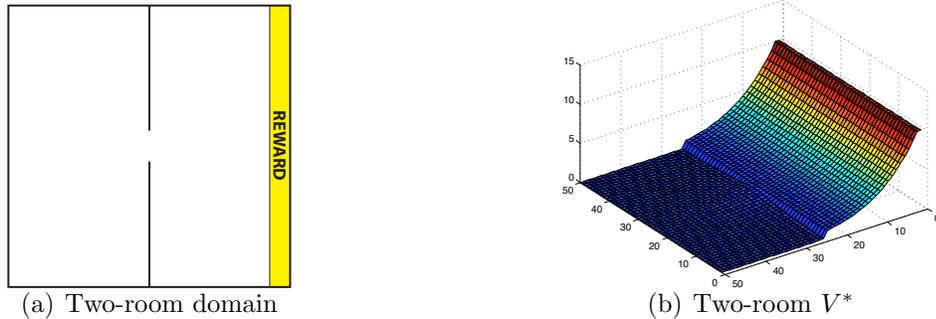


FIGURE 5.3: Two-room domain

regularization term to demonstrate the effects on the value function and Bellman error decomposition. Figures 5.4(a), 5.4(b), 5.4(c), and 5.4(d) show the approximate value function and absolute value of the error components at the grid points resulting for  $\Sigma_R = \Sigma_P = 0$ . We used SVD to handle the ill-conditioned matrix inversion problems that resulted in this case. Note that the scale of the error plots is different from the scale of the value function plot. The large errors at the edges of the space are due to our sampling strategy, which provided fewer neighboring points at the edges and corners. More interestingly, however, the value function is extremely uneven and inconsistent throughout the space. It is difficult to identify why by looking only at the Bellman error in Figure 5.4(b). However, it is clear from examining 5.4(c) that the Bellman error is dominated by the transition error component and that there is overfitting across much of the domain. However, the reward error in Figure 5.4(d) is much less uneven, indicating that the reward approximation is not overfitting. This is not surprising since the reward is deterministic for this problem. Overall, these results are consistent with a need for greater regularization in the approximate transition function. This is precisely what one would expect with *a priori* knowledge of a noisy transition function and deterministic rewards; in practice, however, we would not have such knowledge of the noise sources in the model.

Figures 5.4(e) and 5.4(f) show the value function and transition error resulting

from the same approximation, but with  $\Sigma_P = 0.1\mathbf{I}$ . Reward error is not shown as it remained unchanged. As we expected, the value function is smoother. Figure 5.4(f) shows the transition error of the regularized approximation. Note again that the scale of the error plot is different from the scale of the value function plot. There are still areas of high error, as the large amount of noise relative to the number of samples still makes an approximation difficult. However, the transition error is much better controlled, demonstrating that the regularization has performed as we might expect. Overall, the value function has a far more reasonable shape, and the remaining problems with the value function are primarily attributable to a scarcity of samples in the corners.

In this particular example, the Bellman error decomposition is less useful for revealing an excess of regularization. Under the optimal policy for this problem, the agent does not bump into any walls, except in cases where it is caused by noise. The transition function is therefore a fairly smooth function and excessive regularization has the effect of making the approximation too flat. This has a global effect of squashing the value function, but, in contrast with the case of too little regularization, the effect is not particularly salient from a graph of the transition error alone.

This simple example demonstrates both the advantage of separate reward and transition regularizers, and the insight offered by the Bellman error decomposition. From a wider perspective, the model-based viewpoint decouples the transition and reward approximation aspects of the problem, and the Bellman error decomposition provides a window into understanding the behavior of otherwise opaque approximation schemes.

### 5.3 Generalized Analysis

This section presents the somewhat parenthetical note that the Bellman error decompositions presented in the previous two sections are special cases of a decomposition

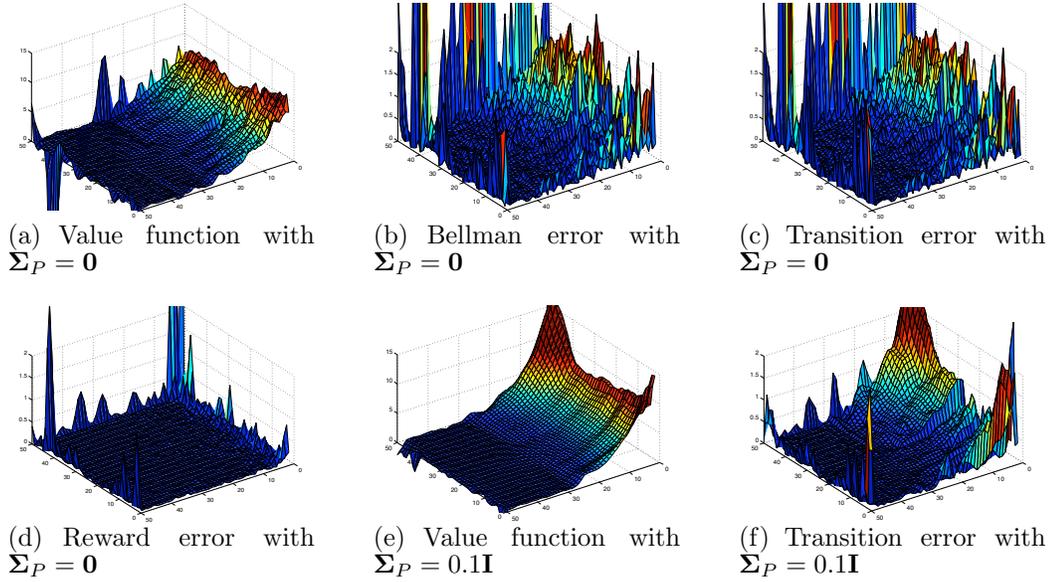


FIGURE 5.4: Bellman error decomposition for the continuous two-room problem, with two different values for the transition regularization matrix  $\Sigma_p$ . Note that the scale on the value function plots is different from the scale on the error plots.

on any fixed-point approximation architecture. That is, let us define  $\hat{R}$  and  $\hat{P}$  as our reward and transition approximations in an arbitrary approximation architecture. In the previous sections, our  $\hat{P}$  predicted next feature values and next kernel values. Because we are working in an arbitrary approximation architecture, we cannot be so precise; instead, we define  $\hat{P}$  such that  $\hat{P}V$  is approximately equal to the values at the next states. Similar to previous sections, if we define  $\Delta_R = R - \hat{R}$  and  $\Delta_P = P - \hat{P}$ , and we define an approximation  $\hat{V}$  to be the fixed point, that is  $\hat{V} = \hat{R} + \gamma\hat{P}\hat{V}$ , we can prove the following theorem:

**Theorem 5.3.1.** *Regardless of the approximation architecture of  $\hat{V}$ ,  $\hat{R}$ , and  $\hat{P}$ , if  $\hat{V}$  is a fixed point, then*

$$BE(\hat{V}) = \Delta_R + \gamma\Delta_P\hat{V}.$$

*Proof.* Using the fact that  $\hat{V}$  is a fixed-point, plus our definitions of  $\Delta_R$  and  $\Delta_P$ , we

can perform the following:

$$\begin{aligned}
BE(\hat{V}) &= R + \gamma P \hat{V} - \hat{V} \\
&= (\Delta_R + \hat{R}) + \gamma(\Delta_P + \hat{P})\hat{V} - \hat{V} \\
&= (\Delta_R + \gamma\Delta_P\hat{V}) + \hat{R} + (\gamma\hat{P} - \mathbf{I})\hat{V} \\
&= (\Delta_R + \gamma\Delta_P\hat{V}) + \hat{R} - (\mathbf{I} - \gamma\hat{P})\hat{V} \\
&= \Delta_R + \gamma\Delta_P\hat{V}
\end{aligned}$$

□

This theorem demonstrates that analysis of all fixed-point approximations would benefit from understanding the underlying model approximation. At the risk of oversimplifying, we note that the concept of model-based and model-free approximations is misleading, as a good implicit model approximation always leads to a good value function approximation.

## $L_1$ -Regularization For Feature Selection

The previous chapters discussed methods for analyzing previously chosen sets of features. This analysis is important, as there tends to be a process of approximating the function, analyzing features, tweaking them, and re-running the approximator until the approximation is good. This is ultimately unsatisfying, as this iterative process of trying different feature sets until success depends on human intuition, scuttling the idea of autonomous, artificially intelligent behavior. Additionally, we would prefer to have faith in our methods even when we don't have good intuition about the solution. After all, if we know the solution, why are we bothering?

The remainder of this document instead focuses on methods for approximating value functions while simultaneously, and automatically, choosing basis functions from a large, possibly overcomplete dictionary. To ease the feature selection problem, this dictionary can consist of the union of all candidate feature sets. By combining feature selection with value function approximation in the same step, we allow the approximator to choose the most useful linear space, rather than employing human intuition.

The work in Chapter 4, along with the work of Farahmand et al. (2008) and Kolter

and Ng (2009), has demonstrated the usefulness of regularization in value function approximation. We can combine this benefit from regularization with automatic feature selection by using  $L_1$  regularization.

The contributions of this chapter include the presentation of a novel value function approximation scheme, called  $L_1$ -Regularized Approximate Linear Programming (RALP) (Petrik et al., 2010), which has the following desirable characteristics:

- RALP performs automatic feature selection during the approximation step.
- The RALP approximation approximates  $V^*$ , that is, the value function of the *optimal* policy, not of a predetermined intermediate policy from policy iteration.
- The average approximation error  $\|\hat{V} - V^*\|_{1,\rho}$  is bounded.
- The empirical performance of greedy policies based on RALP value function approximations outperform those of existing value function approximation techniques.

## 6.1 Previous Work

### 6.1.1 Approximate Linear Programming

The work in this chapter is built on Approximate Linear Programming (ALP). ALP addresses a weakness of the LP formulation originally presented in Subsection 2.2.5. Because the number of variables in the LP formulation equals  $|\mathcal{S}|$ , this formulation is extremely unwieldy and time-consuming to solve in large state spaces. Therefore, Schweitzer and Seidmann (1985) introduced the ALP, which takes the following form:

$$\begin{aligned} \min_{\mathbf{w}} \quad & \rho^T \Phi \mathbf{w} \\ \text{s.t.} \quad & T_a \Phi(s) \mathbf{w} \leq \Phi(s) \mathbf{w} \quad \forall s \in \mathcal{S}, a \in \mathcal{A}. \end{aligned}$$

To ensure feasibility, one of the features is assumed to be the constant vector  $\mathbf{1}$ , where  $\mathbf{1}$  is a vector of all ones; we will keep this assumption throughout all our LP discussions.

Note that while the number of variables has diminished from the number of states to the number of features, the number of required constraints in ALP is still  $|\mathcal{S}| \times |\mathcal{A}|$ , which is oftentimes impractically large or infinite. One solution, which was presented by de Farias and Van Roy (2004), is to sample a small set of constraints from a distribution based on the optimal policy and a Lyapunov function on the state space. It is then possible to state that with high probability, the solution will not change too much. Unfortunately, if the optimal policy is not known, as is usually the case, guarantees are lost not only on the solution quality, but also on the boundedness of the linear program.

To understand the source of unboundedness, let us consider the simple MDP shown in Figure 6.1. This gives us the following ALP:

$$\begin{aligned} \min_{\mathbf{w}} \quad & \rho^T \Phi \mathbf{w} \\ \text{s.t.} \quad & R(s_1) + \gamma \Phi(s_2) \mathbf{w} \leq \Phi(s_1) \mathbf{w} \\ & R(s_2) + \gamma \Phi(s_3) \mathbf{w} \leq \Phi(s_2) \mathbf{w} \\ & R(s_3) + \gamma \Phi(s_4) \mathbf{w} \leq \Phi(s_3) \mathbf{w} \\ & R(s_4) + \gamma \Phi(s_4) \mathbf{w} \leq \Phi(s_4) \mathbf{w} \end{aligned}$$

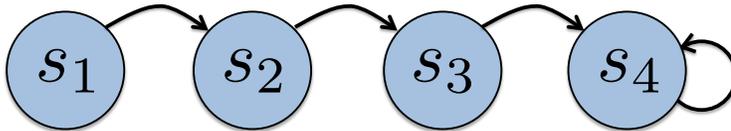


FIGURE 6.1: A simple four-state chain with no action choices.

If we use indicator functions as our feature set, where  $\phi_1 = \mathbf{1}$ , and  $\phi_i(s_j) = 1$  if  $i = j$  and 0 otherwise, and remove the third constraint, it is clear the LP is free to drop the value of state  $s_3$  to negative infinity, demonstrating the unboundedness of the LP.

It is clear this example is an extreme case. However, any time a feature has significantly more support on the left side of the constraints than the right, the LP is susceptible to this behavior.

In addition to its theoretical problems, ALP has also often under-performed approximate dynamic programming methods in practice; this issue has been recently studied and only partially remedied (Petrik and Zilberstein, 2009; Desai et al., 2009).

### 6.1.2 $L_1$ Regularization for Regression

$L_1$  regularization for feature selection was originally devised as a method applicable to linear regression. Regression and value function approximation differ in that in regression the data are sampled directly from the target function (plus noise), while this is obviously untrue for value function approximation. However, the goal of feature selection remains the same: choose a small set of features  $\Phi$  and weights  $\mathbf{w}$  such that for a given function  $f$ ,  $\hat{f}(x) = \Phi(x)\mathbf{w}$  is close to the desired  $f(x)$ . Noise in the function or a lack of sampled points may cause this approach to overfit, so it is common to use regularization.

Regularization forces the parameters to be small. To understand the effects of small parameters, we take the derivative of  $\hat{f} = \Phi\mathbf{w}$ :

$$\frac{d\hat{f}}{dx} = \frac{d\Phi}{dx}\mathbf{w}.$$

By forcing the weights to be small, we are also limiting the rate at which our approximation can change. This helps to reduce overfitting, by not allowing the approximation to change quickly enough to fit every data point.

Traditionally, regularization has taken the form of  $L_2$  regularization, which encourages the sum of the squares of the parameters to be small. However,  $L_1$  regularization, while still requiring weights to be small by limiting the sum of absolute values of the parameters, has the additional benefit of encouraging *sparsity* (Tibshi-

rani, 1996). A weight vector is sparse when many weights are equal to 0. Therefore,  $L_1$ -regularized methods can be given a large, overcomplete basis, and the regularization will force only a small number of features to have non-zero weights, effectively performing feature selection.

To understand the geometry of  $L_1$  constraints and the source of sparsity, consider the illustration in Figure 6.2, in which the shape of the  $L_1$  ball encourages a sparse solution at a vertex, while the  $L_2$  ball encourages a non-sparse solution.

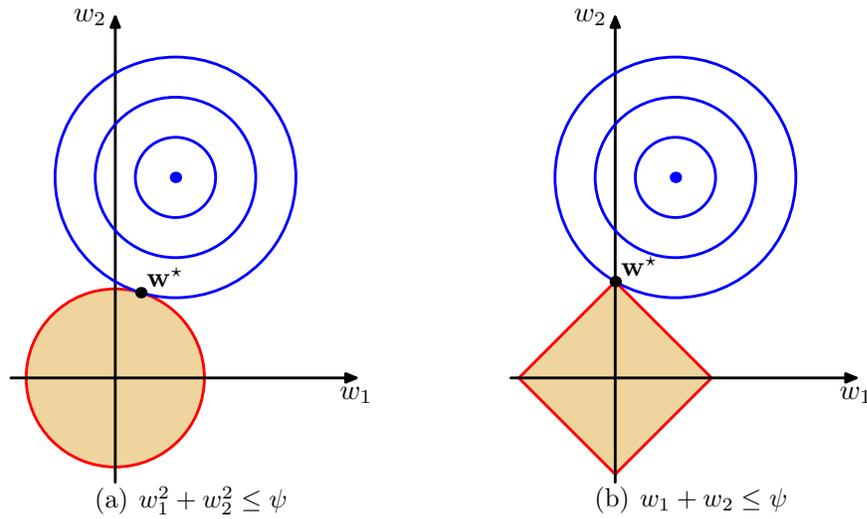


FIGURE 6.2: The concentric circles indicate error from the best solution in  $span(\Phi)$ , while the orange regions indicate the region allowed by the regularization constraint. In Figure 6.2(a), we see that the shape of the  $L_2$  ball causes the solution to be non-sparse. In comparison, in Figure 6.2(b), we see that the shape of the  $L_1$  ball has encouraged a solution at a vertex, where  $w_2 = 0$ . This figure was taken from Bishop (2006).

Tibshirani (1996) first applied  $L_1$  regularization to regression via the lasso estimate, which was

$$\begin{aligned} \underset{\mathbf{w}}{\operatorname{argmin}} \quad & \sum_{x \in \mathcal{S}} (f(x) - \Phi(x)\mathbf{w})^2 \\ \text{s.t.} \quad & \|\mathbf{w}_{-1}\|_1 \leq \psi, \end{aligned}$$

where  $\mathbf{w}_{-1}$  is the vector of all weights that do not correspond to the feature  $\mathbf{1}$ . The weight on the constant vector is not included in the constraint because it is unreasonable to penalize constant shifts in the approximation. The lasso estimate can be calculated with quadratic programming techniques. Figure 6.3 illustrates both the smoothing and sparsity effects of increasing regularization by decreasing the value of  $\psi$ .

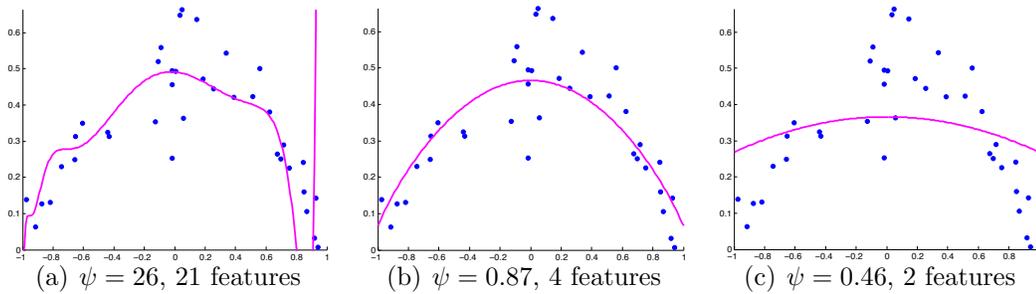


FIGURE 6.3: This series of graphs illustrates the effect of tightening the  $L_1$  regularization parameter in the lasso estimate. The features for this illustration were monomials of  $x$ . It is clear that as  $\psi$  decreases, there is an increase in smoothness and a decrease in number of features weighted with a nonzero.

It was later shown that the lasso estimate could be calculated quicker with a technique called Least Angle Regression (LARS) (Efron et al., 2004). The authors show that the optimal solution of the lasso estimate could be computed iteratively, updating subsets of weights until solutions for all values of  $\psi$  are calculated. LARS takes advantage of the fact that the path through  $\mathbf{w}$ -space as  $\psi$  increases is piecewise-linear; additionally, the points at which the slope changes can be calculated in closed form. As a result, a homotopy method which gradually increases the value of  $\psi$  can calculate weights  $\mathbf{w}$  for all values of  $\psi$  without solving the approximator for every intermediate value of  $\psi$ .

### 6.1.3 $L_1$ Regularization for Value Function Approximation

LARS was later brought into the value-function approximation domain with LARS-TD (Kolter and Ng, 2009). LARS-TD applies  $L_1$  regularization to the LSTD solution for finding an approximate value function given a set of features  $\Phi$  and a policy  $\pi$ ; this was the first method to find an  $L_1$  regularized fixed point while performing automatic feature selection. Under certain conditions, LARS-TD was guaranteed to arrive at a solution for each value of  $\psi$ .

Kolter and Ng (2009) implemented LARS-TD as a value function generation step for LSPI (Lagoudakis and Parr, 2003); however, because LARS-TD approximates the value function of a specific policy, a change in policy forces LARS-TD to completely retrace its solution path. This is slow and wastes many of the benefits of the homotopy method. An improvement to address this problem, called LC-MPI, was introduced by Johns et al. (2010). LC-MPI returns not just a value for every value of  $\psi$ , but also the optimal policy. This removes the need to surround it with a policy improvement step, and greatly improves the computational speed.

While LC-MPI improves on the guarantees of LARS-TD by promising a *unique* solution for each setting of  $\psi$ , neither method offers a guarantee on approximation quality. In fact, even these guarantees only hold if samples are drawn from the policy being evaluated; this is rare, especially for LC-MPI, where the policy being evaluated is itself changing.

## 6.2 $L_1$ -Regularized Approximate Linear Programming

In this section, we introduce  $L_1$ -regularized ALP (RALP) as an approach to automate feature selection and approximate value functions. Adding  $L_1$  regularization to ALP permits the user to supply an arbitrarily rich set of features while decreasing the risk of overfitting. Additionally, RALP alleviates the problems of ALP presented

in Subsection 6.1.1; in particular, the requirement to have a constraint for every possible state-action pair can be removed. In comparison to the work by de Farias and Van Roy (2004), this requirement can be removed without knowing the optimal policy.

We first present RALP for sampling *with expectation*, in which our samples are of the form defined by  $\Sigma$ ; this corresponds to sampling in noiseless domains. We then present RALP for sampling without expectation, in which our samples are of the form defined by  $\dot{\Sigma}$  in Section 2.1.

The noiseless RALP for basis  $\Phi$  and  $L_1$  constraint  $\psi$  is defined as follows:

$$\begin{aligned} \min_{\mathbf{w}} \quad & \rho^T \Phi \mathbf{w} \\ \text{s.t.} \quad & T_{\sigma^a} \Phi(\sigma^s) \mathbf{w} \leq \Phi(\sigma^s) \mathbf{w} \quad \forall \sigma \in \Sigma \\ & \|\mathbf{w}\|_{1,e} \leq \psi, \end{aligned} \tag{6.1}$$

where  $\rho$  is a distribution over initial states, and  $\|w\|_{1,e} = \sum_i |e(i)w(i)|$ . It is generally assumed that  $\rho$  is a constant vector and  $e = \mathbf{1}_{-1}$ , which is a vector of all ones but for the position corresponding to the constant feature, where  $e(i) = 0$ . Note we are using the *expected* Bellman operator  $T$ .

Note that noiseless RALP is a generalization of ALP; when  $\psi$  approaches infinity, the RALP solution approaches the ALP solution.

The noisy RALP looks very similar to the noiseless version, except in the use of the sampled Bellman operator  $\dot{T}$ :

$$\begin{aligned} \min_{\mathbf{w}} \quad & \rho^T \Phi \mathbf{w} \\ \text{s.t.} \quad & \dot{T}_{\dot{\sigma}^a} \Phi(\dot{\sigma}^s) \mathbf{w} \leq \Phi(\dot{\sigma}^s) \mathbf{w} \quad \forall \dot{\sigma} \in \dot{\Sigma} \\ & \|\mathbf{w}\|_{1,e} \leq \psi, \end{aligned} \tag{6.2}$$

RALP also introduces a new parameter  $\psi$ , which needs to be tuned. We offer two possible methods for doing this. The first is a homotopy method reminiscent of LARS-TD presented by Petrik et al. (2010). The second is cross-validation on

solutions calculated at many different values of  $\psi$ . This process is extremely fast; if we consider two regularization values,  $\psi_1$  and  $\psi_2 = \psi_1 + \epsilon$ , the feasible space in the RALP when  $\psi = \psi_2$  is a superset of the feasible space when  $\psi = \psi_1$ . Therefore, a warm start with the solution for  $\psi_1$  can eliminate most of the processing time for  $\psi_2$ .

The RALP is faster to solve than the ALP with all constraints; assuming the samples RALP is defined on do not consist of all possible state-action pairs, the constraints required are a subset of the ALP constraints. Additionally, assuming sufficient regularization, most variables are equal to 0; this means only a very few Bellman constraints are tight, making constraint generation an attractive approach.

In constraint generation, we iterate between solving an LP with only a subset of the constraints, and identifying the constraint which was left out that is most violated by our resulting solution. That most violated constraint is then added to the LP, which is then re-run. This process continues until no constraints are violated.

Obviously, an LP with fewer constraints results in a faster solve time; however, it would seem there would be a tradeoff between this faster solve time and the repeated solving of the LP. Fortunately, we can warm-start each LP with the dual solution of the previously solved LP, which is still feasible.

In our LP, with only a few constraints which end up being tight, this approach works very well; our initial constraint set is all constraints necessary to perform  $L_1$  regularization and a small, randomly chosen set of the Bellman constraints. Problems with several thousand samples which took several hours to run without constraint generation took merely minutes with it.

The next two sections demonstrate the effectiveness of RALP, first by proving theoretical bounds on the approximation error, and second by comparing policies generated from RALP value functions with the policies generated by other value function approximation algorithms.

### 6.3 Theoretical Bounds

The purpose of this section is to show that RALP offers two main benefits over ALP. First, even when the constraints are sampled and incomplete, it is guaranteed to provide a feasible and bounded solution (Lemma 6.3.1). Since feasibility does not imply that the solution is close to optimal, we then show that under easy-to-satisfy assumptions — such as smooth reward and transition functions — RALP guarantees that the error due to both the missing constraints and noise in sampling is small. While this bound is too loose to be of practical value, it does provide a perspective on the potential for  $L_1$  regularization to control the damage caused by missing constraints. We first present a bound for the noiseless RALP of LP 6.1, and then for the more realistic noisy RALP of LP 6.2.

$L_1$ -regularized ALP differs from standard ALP in two significant ways. First, most obviously,  $L_1$ -regularized ALP adds an additional constraint on the  $L_1$  norm of the weights. Additionally, we claim that given reward, transition, and basis functions which change a bounded amount between states,  $L_1$ -regularized ALP can still provide a reasonable value function when performed with fewer than a complete set of constraints, rather than enumerating every state in the space with rows in the design matrix. The construction of our performance bound will help us tie  $L_1$ -regularized ALP to standard ALP despite these differences.

We will assume that every feature that is not  $\mathbf{1}$  is of mean 0 and scaled such that  $\|\phi\|_1 = 1$ . Features are also standardized in lasso and LARS, but are scaled such that  $\|\phi\|_2 = 1$ ; using the  $L_1$ -norm will be more useful for our performance bound. Additionally, using the  $L_1$ -norm is arguably more natural for features that are Gaussian kernels, as they already exhibit an  $L_1$  norm of 1. Intuitively, standardized features ensure that features are added because shape is helpful, not because they are of larger magnitude. Also, standardization ensures orthogonality with  $\mathbf{1}$ , and

keeps constant shifts from being penalized by the  $L_1$  regularization.

### 6.3.1 Noiseless RALP

We first show that even with missing constraints, the  $L_1$  regularization guarantees a feasible, bounded solution as long as  $\mathbf{1} \in \text{span}(\Phi)$ .

**Lemma 6.3.1.** *Let  $M_1$  be an MDP on which we have defined a RALP, with constraints defined on a set of samples  $\Sigma$ . Assume  $\mathbf{1} \in \text{span}(\Phi)$ . The RALP will be feasible and bounded.*

*Proof.* We first show that a feasible solution exists. Let us set  $\mathbf{w}_{-1} = \mathbf{0}$ , where  $\mathbf{0}$  is the vector of zeros. Clearly, the  $L_1$  constraint is satisfied.

Let us now define  $R_{max}$  to be the maximum reward attainable in the MDP. Every other constraint now takes the form

$$R_{max} + \gamma \mathbf{w}_1 \leq \mathbf{w}_1,$$

where  $\mathbf{w}_1$  is the weight corresponding to the  $\mathbf{1}$  feature. As long as  $\mathbf{w}_1 \geq \frac{R_{max}}{1-\gamma}$ ,  $\mathbf{w}$  is a feasible solution.

We now show the RALP is bounded. Let  $\psi$  be the  $L_1$  regularization parameter, so  $\|\mathbf{w}_{-1}\|_1 = \psi$ . This guarantees  $\mathbf{w}_{-1}$  is bounded. We now show  $\mathbf{w}_1$  is also bounded.

If we expand the Bellman operator, every constraint has the form

$$R + \gamma P_{\sigma^a} \Phi_{-1}(\sigma^s) \mathbf{w}_{-1} + \gamma \mathbf{w}_1 \leq \Phi_{-1}(\sigma^s) \mathbf{w}_{-1} + \mathbf{w}_1,$$

for some  $\sigma \in \Sigma$ . Therefore,

$$\mathbf{w}_1 \geq \frac{R + \gamma P_{\sigma^a} \Phi_{-1}(\sigma^s) \mathbf{w}_{-1} - \Phi_{-1}(\sigma^s) \mathbf{w}_{-1}}{1 - \gamma}.$$

Because  $\|\phi\|_1 = 1$  for all  $\phi \in \Phi_{-1}$ , and because  $\|\mathbf{w}_{-1}\|_1 = \psi$

$$-\psi \leq \Phi_{-1} \mathbf{w}_{-1} \leq \psi.$$

We now define  $R_{min}$  to be the minimum reward attainable in the MDP. If we replace  $R$  with  $R_{min}$ , and every instance of  $\Phi_{-1}(s)\mathbf{w}_{-1}$  with either  $\psi$  or  $-\psi$ , we show that the constraints guarantee

$$\mathbf{w}_1 \geq \frac{R_{min} - (1 + \gamma)\psi}{1 - \gamma}.$$

The RALP is both bounded and feasible.  $\square$

We now show that for discrete state spaces, the RALP minimizes  $\|V^* - \Phi\mathbf{w}\|_{1,\rho}$  with constraints on  $\|\mathbf{w}_{-1}\|_1$  in a lemma that will be used later to prove a performance bound. This lemma extends a lemma presented by de Farias and Van Roy (2003).

**Lemma 6.3.2.** *If every state-action pair is represented with a constraint in the RALP, a vector  $\mathbf{w}^*$  solves the RALP if and only if it solves*

$$\begin{aligned} \underset{\mathbf{w}}{\operatorname{argmin}} \quad & \|V^* - \Phi\mathbf{w}\|_{1,\rho} \\ \text{s.t.} \quad & T_a\Phi(s)\mathbf{w} \leq \Phi(s)\mathbf{w} \quad \forall s \in \mathcal{S}, a \in \mathcal{A} \\ & \|\mathbf{w}_{-1}\|_1 \leq \psi \end{aligned}$$

*Proof.* For any policy  $\pi$ , the Bellman operator  $T\pi$  is a contraction in max norm. If the Bellman error is one-sided,  $T$  is also monotonic. Therefore, for any  $V$  such that  $V \geq TV$ ,

$$V \geq TV \geq T^2V \geq V^*.$$

Therefore, any  $\mathbf{w}$  that is a feasible solution to a RALP satisfies  $\Phi\mathbf{w} \geq V^*$ . From this, we can conclude

$$\begin{aligned} \|V^* - \Phi\mathbf{w}\|_{1,\rho} &= \sum_{x \in \mathcal{S}} \rho(x) |V^*(x) - \Phi(x)\mathbf{w}| \\ &= \rho^T \Phi\mathbf{w} - \rho^T V^*. \end{aligned}$$

Because  $V^*$  is constant, minimizing  $\rho^T \Phi\mathbf{w}$  with RALP constraints is equivalent to minimizing  $\|V^* - \Phi\mathbf{w}\|_{1,\rho}$  with RALP constraints.  $\square$

We next assume every state *is* represented by rows in  $\Phi$ , and calculate a similar bound to the one demonstrated for the standard ALP by de Farias and Van Roy (2003). This lemma shows the addition of the  $L_1$  constraint does not radically change this bound.

**Lemma 6.3.3.** *Assume  $\mathbf{1}$  is in  $\text{span}(\Phi)$ , and let  $n = |\mathcal{S}|$ . Then, if  $\tilde{\mathbf{w}}$  is an optimal solution to the RALP, and  $\mathcal{W}$  is the set of weight vectors such that  $\|\mathbf{w}_{-1}\| \leq \psi$ ,*

$$\|V^* - \Phi\tilde{\mathbf{w}}\|_{1,\rho} \leq \frac{2}{1-\gamma} \min_{\mathbf{w} \in \mathcal{W}} \|V^* - \Phi\mathbf{w}\|_{\infty}.$$

*Proof.* Let  $\mathbf{w}^*$  be a vector minimizing  $\|V^* - \Phi\mathbf{w}\|_{\infty}$  such that  $\|\mathbf{w}_{-1}^*\|_1 \leq \psi$ . For ease of notation, define  $\epsilon = \|V^* - \Phi\mathbf{w}^*\|_{\infty}$ . We look to construct a point  $\bar{\mathbf{w}}$  such that  $\bar{\mathbf{w}}$  is both a feasible point in our RALP and such that  $\Phi\bar{\mathbf{w}}$  is within  $O(\epsilon)$  of  $V^*$ .

Because the Bellman operator is a contraction in max norm,

$$\|T\Phi\mathbf{w}^* - V^*\|_{\infty} \leq \gamma\|\Phi\mathbf{w}^* - V^*\|_{\infty}.$$

From this,

$$T\Phi\mathbf{w}^* \leq V^* + \gamma\epsilon\mathbf{1}. \tag{6.3}$$

Additionally, by the definition of the Bellman operator, for any vector  $V$  and any scalar  $k$ ,

$$\begin{aligned} T(V + k\mathbf{1}) &= \max_{\pi} [R + \gamma P_{\pi}(V + k\mathbf{1})] \\ &= \max_{\pi} [R + \gamma P_{\pi}V + \gamma k\mathbf{1}] \\ &= \max_{\pi} [R + \gamma P_{\pi}V] + \gamma k\mathbf{1} \\ &= TV + \gamma k\mathbf{1} \end{aligned}$$

Combining this with Equation 6.3,

$$\begin{aligned}
T(\Phi \mathbf{w}^* + k \mathbf{1}) &= T\Phi \mathbf{w}^* + \gamma k \mathbf{1} \\
&\leq V^* + \gamma \epsilon \mathbf{1} + \gamma k \mathbf{1} \\
&\leq \Phi \mathbf{w}^* + \epsilon \mathbf{1} + \gamma \epsilon \mathbf{1} + \gamma k \mathbf{1} \\
&= \Phi \mathbf{w}^* + (1 + \gamma) \epsilon \mathbf{1} + \gamma k \mathbf{1}
\end{aligned}$$

Now, let  $k = \frac{1+\gamma}{1-\gamma} \epsilon$ .

$$\begin{aligned}
T\left(\Phi \mathbf{w}^* + \frac{1+\gamma}{1-\gamma} \epsilon \mathbf{1}\right) &\leq \Phi \mathbf{w}^* + (1 + \gamma) \epsilon \mathbf{1} + \gamma \frac{1+\gamma}{1-\gamma} \epsilon \mathbf{1} \\
&= \Phi \mathbf{w}^* + \frac{(1-\gamma)(1+\gamma) + \gamma(1+\gamma)}{1-\gamma} \epsilon \mathbf{1} \\
&= \Phi \mathbf{w}^* + \frac{(1+\gamma)(1-\gamma + \gamma)}{1-\gamma} \epsilon \mathbf{1}
\end{aligned}$$

After cancellation,

$$T\left(\Phi \mathbf{w}^* + \frac{1+\gamma}{1-\gamma} \epsilon \mathbf{1}\right) \leq \Phi \mathbf{w}^* + \frac{1+\gamma}{1-\gamma} \epsilon \mathbf{1}. \quad (6.4)$$

Because  $\mathbf{1}$  is within  $\text{span}(\Phi)$ , we can construct a vector  $\bar{\mathbf{w}}$  such that

$$\Phi \bar{\mathbf{w}} = \Phi \mathbf{w}^* + \frac{1+\gamma}{1-\gamma} \epsilon \mathbf{1}.$$

Because we do not count the weight on  $\mathbf{1}$  when calculating  $\|\mathbf{w}_{-1}\|_1$ , and because Equation 6.4 guarantees that we have satisfied our Bellman constraints,  $\bar{\mathbf{w}}$  is a feasible solution to our  $L_1$ -regularized ALP. By the triangle inequality,

$$\begin{aligned}
\|\Phi \bar{\mathbf{w}} - V^*\|_\infty &\leq \|V^* - \Phi \mathbf{w}^*\|_\infty + \|\Phi \mathbf{w}^* - \Phi \bar{\mathbf{w}}\|_\infty \\
&\leq \epsilon \left(1 + \frac{1+\gamma}{1-\gamma}\right) \\
&= \frac{2\epsilon}{1-\gamma}
\end{aligned}$$

So, if  $\tilde{\mathbf{w}}$  is an optimal solution to the RALP, then by Lemma 6.3.2,

$$\begin{aligned} \|V^* - \Phi \tilde{\mathbf{w}}\|_{1,\rho} &\leq \|V^* - \Phi \bar{\mathbf{w}}\|_{1,\rho} \\ &\leq \|V^* - \Phi \bar{\mathbf{w}}\|_\infty \\ &\leq \frac{2\epsilon}{1-\gamma}, \end{aligned}$$

where the second inequality is due to  $\rho$  being a probability distribution.  $\square$

This lemma demonstrates that if every state-action pair has a constraint in the LP, and if  $V^*$  can be closely approximated with an approximation in  $\text{span}(\Phi)$  with  $\|\mathbf{w}_{-1}\|_1 \leq \psi$ , then the  $L_1$ -regularized ALP will produce a good solution. In other words, the solution offered by the LP will not be too far off from the actual minimizing solution  $\mathbf{w}^*$ .

We next seek to show that if we have missing constraints, our resulting error at unconstrained states can be controlled with  $L_1$  regularization.

**Assumption 6.3.4.** *Assume sufficient sampling, that is, for all  $s \in \mathcal{S}$  and  $a \in \mathcal{A}$ , there exists a  $\sigma \in \Sigma$  such that  $\sigma^a = a$  and:*

$$\begin{aligned} \|\phi(\sigma^s) - \phi(s)\|_\infty &\leq \delta_\phi \\ |R(\sigma^s) - R(s)| &\leq \delta_R \\ |p(s|\sigma^s, \sigma^a) - p(s|s, a)| &\leq \delta_P \quad \forall s \in \mathcal{S} \end{aligned}$$

Note that this assumption is easy to fulfill. For example, this assumption is met if the reward function, basis functions, and transition functions are Lipschitz continuous. More specifically, for all  $s_1, s_2, s_3 \in \mathcal{S}$  and all  $\phi \in \Phi$ , define  $K_R$ ,  $K_P$ , and  $K_\phi$  such that

$$\begin{aligned} \|R(s_1) - R(s_2)\| &\leq K_R \|s_1 - s_2\| \\ \|p(s_3|s_1, a) - p(s_3|s_2, a)\| &\leq K_P \|s_1 - s_2\| \quad \forall a \in \mathcal{A} \\ \|\phi(s_1) - \phi(s_2)\| &\leq K_\phi \|s_1 - s_2\|. \end{aligned}$$

Define the subset  $\Sigma_a \subset \Sigma$ , where a sample  $\sigma \in \Sigma_a$  if  $\sigma \in \Sigma$  and  $\sigma^a = a$ . Let  $c$  be the worst case minimum distance between an unsampled state-action pair  $(s, a)$ , and a sample  $\sigma \in \Sigma_a$ :

$$c = \max_{s \in \mathcal{S}, a \in \mathcal{A}} \min_{\sigma \in \Sigma_a} \|\sigma^s - s\|.$$

Then, Assumption 6.3.4 is met with  $\delta_\phi = cK_\phi$ ,  $\delta_R = cK_R$ , and  $\delta_P = cK_P$ . Other common scenarios also apply; for example, the reward portion is satisfied with a reward function which is 1 for a goal state and 0 elsewhere.

**Lemma 6.3.5.** *Let  $M_1$  be an MDP. Define an incomplete set of samples  $\Sigma$  taken from  $M_1$ , such that not all state-action pairs are sampled, and such that Assumption 6.3.4 is true. Assume for all samples  $\sigma \in \Sigma$ , the RALP for  $M_1$  has the constraint  $T_{\sigma^a} \Phi(\sigma^s) \mathbf{w} \leq \Phi(\sigma^s) \mathbf{w}$ , but is missing all other possible RALP constraints.*

*There exists an MDP  $M_2$  with no missing constraints such that the RALP solution to  $M_1$  is equal to the RALP solution to  $M_2$  and  $\|R_1 - R_2\|_\infty \leq 2(\delta_\phi \psi + \delta_R + \delta_P \psi)$ .*

*Proof.* Our sampling assumption allows us to create *implied* constraints for an arbitrary unsampled state-action pair  $s \in \mathcal{S}, a \in \mathcal{A}$  for MDP  $M_1$ . The constraint we wish we had is

$$R_1(s) + \gamma \sum_{x \in \mathcal{S}} [p(x|s, a) \Phi(x)] \mathbf{w} \leq \Phi(s) \mathbf{w}. \quad (6.5)$$

Let us refer to the sample in  $\Sigma$  which fulfills the sampling assumption with  $s$  and  $a$  as  $\sigma$ . We can now construct a bound for how incorrect each component of Equation 6.5 can be if we use the constraint at  $\sigma$  and our sampling assumption to replace our desired constraint. For instance, our reward function  $R(s)$  is easily bounded.

$$R(\sigma^s) - \delta_R \leq R(s) \leq R(\sigma^s) + \delta_R \quad (6.6)$$

We now try to bound  $\Phi(s) \mathbf{w}$ . Because the sampling assumption allows each basis function to change only a finite amount, and because  $\|\mathbf{w}_{-1}\|_1 \leq \psi$ , and  $\mathbf{1}(s) = \mathbf{1}(\sigma^s)$ ,

$$\Phi(\sigma^s) \mathbf{w} - \delta_\phi \psi \leq \Phi(s) \mathbf{w} \leq \Phi(\sigma^s) \mathbf{w} + \delta_\phi \psi \quad (6.7)$$

Our final component is  $\gamma \sum_{x \in \mathcal{S}} p(x|s, a) \Phi(x) \mathbf{w}$ , which expresses our expected value at the next state. It will be convenient to separate  $\mathbf{1}$  from the rest of  $\Phi$ . We will denote the remainder of the design matrix as  $\Phi_{-1}$ , and the weights that correspond to  $\Phi_{-1}$  as  $\mathbf{w}_{-1}$ . Similarly, we will denote the weight corresponding to  $\mathbf{1}$  as  $\mathbf{w}_1$ .

$$\begin{aligned} \sum_{x \in \mathcal{S}} p(x|s, a) \Phi(x) \mathbf{w} &= \sum_{x \in \mathcal{S}} p(x|s, a) \mathbf{w}_1 + \sum_{x \in \mathcal{S}} p(x|s, a) \Phi_{-1}(x) \mathbf{w}_{-1} \\ &= \mathbf{w}_1 + \sum_{x \in \mathcal{S}} p(x|s, a) \Phi_{-1}(x) \mathbf{w}_{-1} \end{aligned}$$

Again, we have bounded the allowable change in our expression of probability.

$$\begin{aligned} \mathbf{w}_1 + \sum_{x \in \mathcal{S}} p(x|s, a) \Phi_{-1}(x) \mathbf{w}_{-1} &\leq \mathbf{w}_1 + \sum_{x \in \mathcal{S}} [p(x|\sigma^s, \sigma^a) + \delta_P] \Phi_{-1}(x) \mathbf{w}_{-1} \\ &= \mathbf{w}_1 + \sum_{x \in \mathcal{S}} p(x|\sigma^s, \sigma^a) \Phi_{-1}(x) \mathbf{w}_{-1} + \delta_P \sum_{x \in \mathcal{S}} \Phi_{-1}(x) \mathbf{w}_{-1} \end{aligned}$$

Because each basis function  $\phi$  is standardized such that  $\|\phi\|_1 = 1$ , and because  $\|\mathbf{w}_{-1}\|_1 \leq \psi$ , the second summation can be at most  $\psi$ . So,

$$\sum_{x \in \mathcal{S}} p(x|\sigma^s, \sigma^a) \Phi(x) \mathbf{w} - \delta_P \psi \leq \sum_{x \in \mathcal{S}} [p(x|s, a) \Phi(x)] \mathbf{w} \leq \sum_{x \in \mathcal{S}} p(x|\sigma^s, \sigma^a) \Phi(x) \mathbf{w} + \delta_P \psi. \quad (6.8)$$

We showed in Lemma 6.3.1 that a feasible, bounded solution to the RALP on  $M_1$  exists. So, we now combine Equations 6.6, 6.7, and 6.8, and construct our implied constraint to take the place of the missing constraint expressed by Equation 6.5. We see that the maximum possible change by the approximate value function is  $\delta_\phi \psi + \delta_R + \delta_P \psi$ . So, the total cumulative error in the constraint is at most  $2(\delta_\phi \psi + \delta_R + \delta_P \psi)$ . So, we effectively have the following constraint:

$$R_1(s) + q - \gamma \sum_{x \in \mathcal{S}} [p(x|s, a) \Phi(x)] \mathbf{w} \geq \Phi(s) \mathbf{w}, \quad (6.9)$$

where  $|q| \leq 2(\delta_\phi \psi + \delta_R + \delta_P \psi)$ .

Let  $M_2$  be an MDP which is identical in every way to  $M_1$ , except  $R_2(s) = R_1(s) + q$ . The RALP solution for  $M_1$  will be equivalent to the RALP solution for  $M_2$ , and  $\|R_1 - R_2\|_\infty \leq 2(\delta_\phi\psi + \delta_R + \delta_P\psi)$ .  $\square$

For ease of notation, we will define  $\epsilon_p$  to be the maximum possible change in value from a missing sample. More precisely,  $\epsilon_p = \delta_\phi\psi + \delta_R + \delta_P\psi$ . Figure 6.4 contains an illustration of the intuition behind this error and proof.

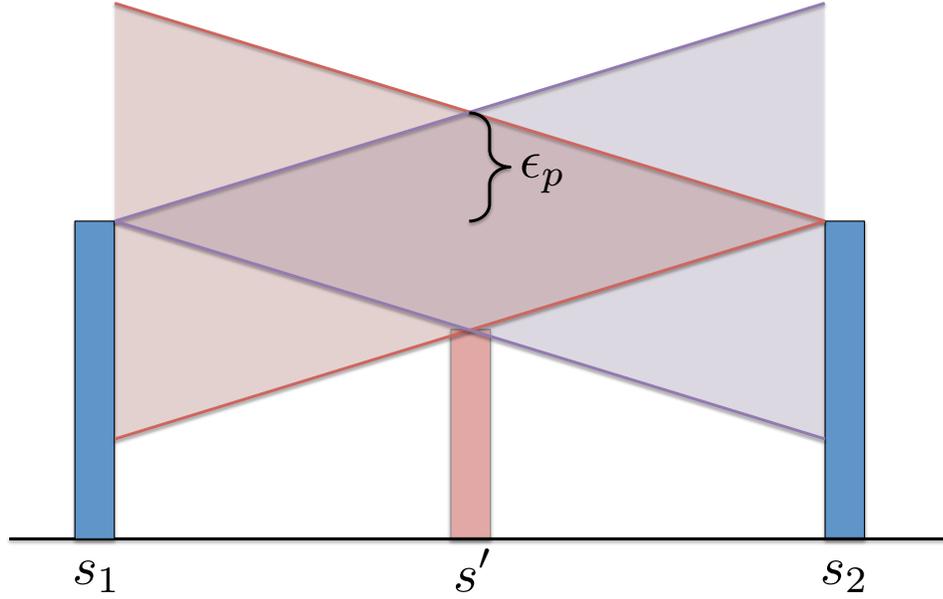


FIGURE 6.4: An illustration of Lemma 6.3.5. The blue bars are constraints at sampled points  $s_1, s_2 \in \mathcal{S}$ . The red and purple lines indicate the maximum rate of change of the value function, given our settings of  $\psi, \delta_\phi, \delta_R$ , and  $\delta_P$ . The center diamond is therefore the feasible area for the approximate value function, and the red bar is the *implied* constraint at some novel point  $s' \in \mathcal{S}$ . Because  $\epsilon_p$  is the maximum change, we see that the difference between the best possible setting of  $\Phi(s')\mathbf{w}$  and the worst possible setting of  $\Phi(s')\mathbf{w}$  is at most  $2\epsilon_p$ .

**Lemma 6.3.6.** *Let  $M_1$  and  $M_2$  be MDPs that differ only in their reward vectors  $R_1$  and  $R_2$ . Let  $V_1^*$  and  $V_2^*$  be their optimal value functions. Then, for  $\|R_1 - R_2\|_\infty \leq \delta$ ,  $\|V_1^* - V_2^*\|_\infty \leq \frac{\delta}{1-\gamma}$ .*

*Proof.* Let  $s$  be an arbitrary point in the sets  $\mathcal{S}_1$  and  $\mathcal{S}_2$ , and define  $r_{1i}(s)$  and  $r_{2i}(s)$  to be the  $i$ -th reward received in exploring  $M_1$  and  $M_2$  from state  $s$ , respectively. Note that

$$V_1^*(s) = \sum_{i=0}^{\infty} \gamma^i \mathbb{E} [r_{1i}(s)]$$

and

$$\begin{aligned} V_2^*(s) &\leq \sum_{i=0}^{\infty} (\gamma^i \mathbb{E} [r_{1i}(s) + \delta]) \\ &= \sum_{i=0}^{\infty} \gamma^i \mathbb{E} [r_{1i}(s)] + \sum_{i=0}^{\infty} \gamma^i \delta \end{aligned}$$

Therefore,

$$\begin{aligned} |V_1^*(s) - V_2^*(s)| &\leq \sum_{i=0}^{\infty} \gamma^i \mathbb{E} [r_{1i}(s)] - \left( \sum_{i=0}^{\infty} \gamma^i \mathbb{E} [r_{1i}(s)] + \sum_{i=0}^{\infty} \gamma^i \delta \right) \\ &= \sum_{i=0}^{\infty} \gamma^i \delta \\ &= \frac{\delta}{1 - \gamma} \end{aligned}$$

Because this is true for an arbitrary  $s$ ,  $\|V_1^* - V_2^*\|_{\infty} \leq \frac{\delta}{1 - \gamma}$ .  $\square$

We can now construct our performance bound for the  $L_1$ -regularized ALP.

**Theorem 6.3.7.** *Let  $M_1$  and  $M_2$  be MDPs as described in Lemma 6.3.5 with reward functions  $R_1$  and  $R_2$  and optimal value functions  $V_1^*$  and  $V_2^*$ . Let  $\tilde{\mathbf{w}}$  be the RALP solution to  $M_1$ .*

$$\|V_1^* - \Phi \tilde{\mathbf{w}}\|_{1,\rho} \leq \frac{2}{1 - \gamma} \min_{\mathbf{w} \in \mathcal{W}} \|V_2^* - \Phi \mathbf{w}\|_{\infty} + \frac{2\epsilon_p}{1 - \gamma}$$

*Proof.* This theorem follows easily from the lemmas in this section. First, we consider  $M_2$ , which we demonstrated existed in Lemma 6.3.5; Lemma 6.3.3 showed that the RALP solution to  $M_2$  is equal to  $\tilde{\mathbf{w}}$ , and

$$\|V_2^* - \Phi\tilde{\mathbf{w}}\|_{1,\rho} \leq \frac{2}{1-\gamma} \min_{\mathbf{w} \in \mathcal{W}} \|V_2^* - \Phi\mathbf{w}\|_\infty. \quad (6.10)$$

Given  $M_2$ , Lemma 6.3.5 showed

$$\|R_1 - R_2\|_\infty \leq 2\epsilon_p, \quad (6.11)$$

which Lemma 6.3.6 demonstrated meant

$$\|V_1^* - V_2^*\|_\infty \leq \frac{2\epsilon_p}{1-\gamma}.$$

Because  $\rho$  is a probability distribution,

$$\begin{aligned} \|V_1^* - V_2^*\|_{1,\rho} &\leq \|V_1^* - V_2^*\|_\infty \\ &\leq \frac{2\epsilon_p}{1-\gamma}. \end{aligned}$$

By the triangle inequality,

$$\begin{aligned} \|V_1^* - \Phi\tilde{\mathbf{w}}\|_{1,\rho} &\leq \|V_2^* - \Phi\tilde{\mathbf{w}}\|_{1,\rho} + \|V_1^* - V_2^*\|_{1,\rho} \\ &\leq \frac{2}{1-\gamma} \min_{\mathbf{w} \in \mathcal{W}} \|V_2^* - \Phi\mathbf{w}\|_\infty + \frac{2\epsilon_p}{1-\gamma} \end{aligned}$$

□

It is clear this is an extremely loose and unwieldy bound for practical purposes. However, it does provide some insight into the effects of  $L_1$  regularization in the RALP setting. A smaller value of  $\psi$  causes the bound in Lemma 6.3.5 to tighten, but causes the set  $\mathcal{W}$  to shrink and become more restrictive; this suggests a tradeoff to be considered when setting the regularization parameter. Additionally, the importance

of covering the space with samples is reinforced; as our sampling coverage improves, we can often tighten the values of  $\delta_\phi$ ,  $\delta_R$ , and  $\delta_P$ . This is particularly intuitive in the case of Lipschitz continuous functions; as  $c$  approaches zero, the bound tightens. Taken another step, as  $\psi$  approaches infinity, and  $c$  approaches zero, the bound approaches that reached by de Farias and Van Roy (2003).

### 6.3.2 Noisy RALP

We now turn our attention to the noisy case, in which we cannot collect sets of samples with expectation  $\Sigma$ , but must instead collect sets of simple samples  $\dot{\Sigma}$ ; as it is unlikely that samples with expectation can be calculated, this is the more likely scenario. Intuitively, this will be another source of error for RALP; not only are constraints missing, resulting in error  $\epsilon_p$ , but the constraints that do exist may be significantly different from the “correct” noiseless constraints. We quantify this noise as  $\epsilon_s$ , where for a set of samples  $\dot{\Sigma}$ ,

$$\epsilon_s = \max_{\dot{\sigma} \in \dot{\Sigma}} \left| T\Phi(\dot{\sigma}^s)\mathbf{w} - \dot{T}\Phi(\dot{\sigma}^s)\mathbf{w} \right|. \quad (6.12)$$

To understand why noise is especially damaging in the LP setting, consider an MDP with no noise, with the exception of a tiny chance of winning the lottery at every state. Figure 6.5(a) illustrates the constraints and resulting value function if the lottery is never won, while Figure 6.5(b) illustrates the results if the lottery is won once. Regardless of the number of times the second state is sampled without winning the lottery, the value at that state will be determined by the one time it was won. In this illustration  $\epsilon_s$  is the difference between the orange constraint and the underlying gray constraint.

The goal of this section is to bound the error in a noisy RALP with missing constraints. We start with the following Lemma:

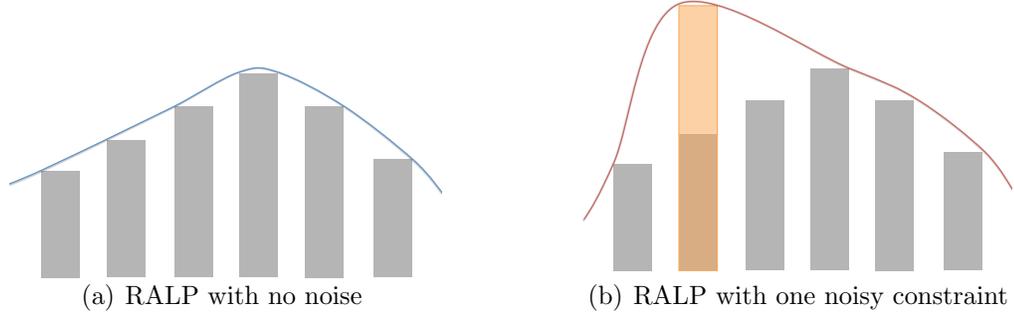


FIGURE 6.5: Illustration of the effect of noise in RALP. The gray bars correspond to sampled constraints, while the orange bar corresponds to the one time the lottery was won.

**Lemma 6.3.8.** *Let  $M_1$  be an MDP which meets the sampling assumption 6.3.4 for some  $\delta_R, \delta_\phi$ , and  $\delta_P$ . Define a set of RALP constraints on simple samples  $\dot{\Sigma}$  taken from  $M_1$ . There exists an MDP  $M_2$  identical in every way to  $M_1$  but for the reward function, with related samples with expectation  $\Sigma$  that generate the same constraints and same RALP solution, and  $\|R_1 - R_2\|_\infty \leq \epsilon_s$ .*

*Proof.* By definition of  $M_1, M_2, \dot{\Sigma}$ , and  $\Sigma$ , for every simple sample  $\dot{\sigma} \in \dot{\Sigma}$ , there exists a sample with expectation  $\sigma \in \Sigma$ , where  $\dot{\sigma}^s = \sigma^s, \dot{\sigma}^a = \sigma^a$ . By our definition of  $\epsilon_s$  in Equation 6.12,  $\dot{\sigma}^r + q + \gamma\Phi(\dot{\sigma}^{s'})\mathbf{w} = \sigma^r + \gamma\Phi(\sigma^{s'})\mathbf{w}$ , where  $|q| \leq \epsilon_s$ .  $\square$

**Lemma 6.3.9.** *Let  $M_1, M_2, \dot{\Sigma}$ , and  $\Sigma$  be defined as in Lemma 6.3.8. Let  $V_1^*$  and  $V_2^*$  be their respective optimal value functions, and let  $\tilde{\mathbf{w}}$  be the RALP solution for both.*

$$\|V_1^* - \Phi\tilde{\mathbf{w}}\|_{1,\rho} \leq \|V_2^* - \Phi\tilde{\mathbf{w}}\|_{1,\rho} + \frac{\epsilon_s}{1-\gamma}.$$

*Proof.* By Lemmas 6.3.8 and 6.3.6, we have  $\|V_1^* - V_2^*\|_\infty \leq \frac{\epsilon_s}{1-\gamma}$ . Because  $\rho$  is a probability distribution,

$$\begin{aligned} \|V_1^* - V_2^*\|_{1,\rho} &\leq \|V_1^* - V_2^*\|_\infty \\ &\leq \frac{\epsilon_s}{1-\gamma}. \end{aligned}$$

So, using the triangle inequality,

$$\begin{aligned}\|V_1^* - \Phi\tilde{\mathbf{w}}\|_{1,\rho} &\leq \|V_2^* - \Phi\tilde{\mathbf{w}}\|_{1,\rho} + \|V_1^* - V_2^*\|_{1,\rho} \\ &\leq \|V_2^* - \Phi\tilde{\mathbf{w}}\|_{1,\rho} + \frac{\epsilon_s}{1-\gamma}.\end{aligned}$$

□

**Theorem 6.3.10.** *Let  $M_1, M_2, \dot{\Sigma}, \Sigma, \delta_R, \delta_\phi,$  and  $\delta_P$  be defined as in Lemma 6.3.8. Furthermore, let  $M_3$  be an MDP with the same RALP solution as the RALP defined on  $\Sigma$  but with expected constraints for every  $(s, a), s \in \mathcal{S}, a \in \mathcal{A}$ . Let  $V_1^*, V_2^*, V_3^*$  be their respective optimal value functions.*

$$\|V_1^* - \Phi\tilde{\mathbf{w}}\|_{1,\rho} \leq \frac{2}{1-\gamma} \min_{\mathbf{w} \in \mathcal{W}} \|V_3^* - \Phi\mathbf{w}\|_\infty + \frac{2\epsilon_p + 3\epsilon_s}{1-\gamma}$$

*Proof.* From Lemma 6.3.9, we know

$$\|V_1^* - \Phi\tilde{\mathbf{w}}\|_{1,\rho} \leq \|V_2^* - \Phi\tilde{\mathbf{w}}\|_{1,\rho} + \frac{\epsilon_s}{1-\gamma}.$$

We now apply Theorem 6.3.7 to relate MDPs  $M_2$  and  $M_3$ . Note that due to the changes in the reward function,  $M_2$  is no longer guaranteed to satisfy our sampling assumption. So, we must replace  $\delta_R$  with  $\delta_R + \epsilon_s$ . If we then apply Theorem 6.3.7 with our new sampling assumption, we know

$$\|V_2^* - \Phi\tilde{\mathbf{w}}\|_{1,\rho} \leq \frac{2}{1-\gamma} \min_{\mathbf{w} \in \mathcal{W}} \|V_3^* - \Phi\mathbf{w}\|_\infty + \frac{2\epsilon_p + 2\epsilon_s}{1-\gamma}.$$

When we combine these two bounds to relate  $M_1$  and  $M_3$ , our theorem follows easily:

$$\|V_1^* - \Phi\tilde{\mathbf{w}}\|_{1,\rho} \leq \frac{2}{1-\gamma} \min_{\mathbf{w} \in \mathcal{W}} \|V_3^* - \Phi\mathbf{w}\|_\infty + \frac{2\epsilon_p + 3\epsilon_s}{1-\gamma}.$$

□

This section has proven that the  $L_1$  regularization allows us to bound error resulting from missing constraints and noise in sampling. However, while it is nice to be reassured of approximation quality, these bounds are obviously quite loose. The next section demonstrates that the method performs much better than these guarantees.

Following that, Chapter 7 introduces a modification to RALP which minimizes the effects of  $\epsilon_s$  on the approximation.

## 6.4 Experimental Results

In this section, we present results indicating that RALP effectively selects from rich feature spaces to outperform ALP and other common algorithms, such as LSPI. First, we use small, toy problems to illustrate the effects of regularization; then, we demonstrate performance on a more difficult suite of problems.

### 6.4.1 Benefits of Regularization

First, we demonstrate and analyze the properties of RALP on a simple chain problem with 200 states, in which the transitions move to the right by one step with a centered Gaussian noise with standard deviation 3. The reward for reaching the right-most state was +1 and the reward in the 20th state was -3. This problem is small to enable calculation of the optimal value function and to control sampling. We uniformly selected every fourth state on the chain. The approximation basis in this problem is represented by piecewise linear features, of the form  $\phi(s_i) = [i - c]_+$ , for  $c$  from 1 to 200; these features were chosen due to their strong guarantees for the sampling bounds.

Figure 6.6 demonstrates the solution quality of RALP on the chain problem as a function of the regularization coefficient  $\psi$ . The figure shows that although the objective of RALP keeps decreasing as  $\psi$  increases, the error from  $\epsilon_p$  overtakes that reduction. It is clear that a proper selection of  $\psi$  improves the quality of the re-

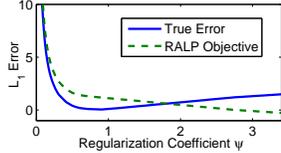


FIGURE 6.6: Comparison of the objective value of RALP with the true error.

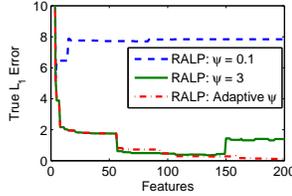


FIGURE 6.7: Comparison of the performance of RALP with two values of  $\psi$  and the one chosen adaptively using the homotopy method.

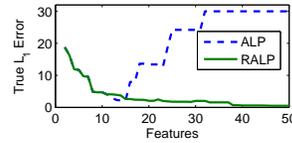


FIGURE 6.8: Average of 45 runs of ALP and RALP as a function of the number of features. Coefficient  $\psi$  was selected using the homotopy method.

sulting approximation. To demonstrate the benefits of regularization as it relates to overfitting, we compare the performance of ALP and RALP as a function of the number of available features in Figure 6.8. While ALP performance improves initially, it degrades severely with more features. The value  $\psi$  in RALP is selected automatically using the homotopy method presented by Petrik et al. (2010). Figure 6.7 demonstrates that RALP may also overfit or perform poorly when the regularization coefficient  $\psi$  is not selected properly.

The next two experiments do not use the homotopy method. In practice, RALP often works much better than what is suggested by our bounds, which can be loose for sparsely sampled large state spaces. In the following experiments, we determined  $\psi$  empirically by solving the RALP for several different values of  $\psi$  and selecting the one that produced the best policy.

#### 6.4.2 Benchmark Problems

*Inverted Pendulum* We now offer experimental results demonstrating RALP’s ability to create effective value functions in balancing an inverted pendulum, a standard benchmark problem in reinforcement learning (Wang et al., 1996; Lagoudakis and Parr, 2003). Samples of the form  $\hat{\Sigma}$  were drawn from random trajectories

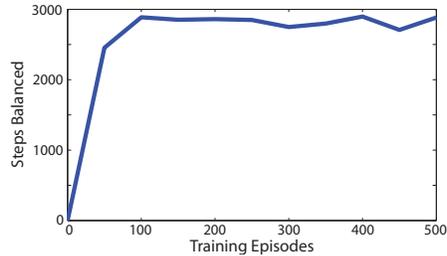


FIGURE 6.9: RALP performance on pendulum as a function on the number of episodes.

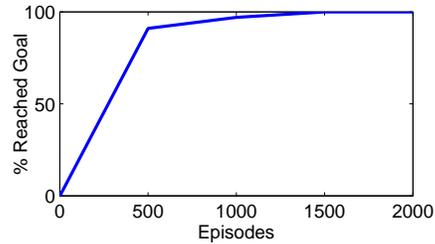


FIGURE 6.10: RALP performance on bicycle as a function on the number of episodes.

with the pendulum starting in an upright state, referred to as episodes. Features were Gaussian kernels of three different widths, a polynomial kernel of degree 3,  $s[1], s[2], s[1] * s[2], s[1]^2, s[2]^2$ , and a constant feature, where  $s[1]$  is the angle, and  $s[2]$  is the angular velocity. For a number of samples  $n$ , this was therefore a total of  $4n + 6$  features.  $\psi$  was 1.4, and an average of 4 features had nonzero weights.

The policy was evaluated based on the number of steps it could balance the pendulum, with an upper limit of 3000. This served to evaluate the policy resulting from the approximate value function. We plot the average number of steps the pendulum was balanced as a function of the number of training episodes in Figure 7.1, as an average of 100 runs. It is clear the controller produced by RALP was effective for small amounts of data, balancing the pendulum for the maximum number of steps nearly all of the time, even with only 50 training episodes. Similarly, it was able to leverage the larger number of available features to construct an effective controller with fewer trajectories than LSPI, which needed 450 training episodes before achieving an average of 2500 balanced steps (Lagoudakis and Parr, 2003).

*Bicycle Balancing and Riding* We also present experimental results for the bicycle problem, in which the goal is to learn to balance and ride a bicycle to a target position (Randløv and Alstrøm, 1998; Lagoudakis and Parr, 2003). This is a challenging benchmark domain in reinforcement learning. Training data consisted of samples for

every action on states drawn from trajectories resulting from random actions up to 35 states long, similar to the inverted pendulum domain. The feature set consisted of monomials up to degree 4 on the individual dimensions and products of monomials up to degree 3, for a total of 159 features.  $\psi$  was 0.03, and an average of 34 features had nonzero weights. We plot the number of runs out of 100 in which the bicycle reached the goal region as a function of number of training episodes in Figure 6.10. Again, a high percentage of runs were successful, even with only 500 training episodes. In comparison, LSPI required 1500 training episodes to pass 80% success.

These results are extremely gratifying and demonstrate that automated feature selection through  $L_1$  regularization is a great improvement over existing methods. While practical performance is outstanding, the bounds offer a source of worry. There are three sources of error: the representation error, which can be reduced through addition of features to the dictionary, error from missing constraints ( $\epsilon_p$ ), which can be managed through tuning of  $\psi$ , and sampling error ( $\epsilon_s$ ), for which there is no obvious technique for management. The next chapter focuses on a solution for noisy environments which introduce a large amount of sampling error.

## $L_1$ -Regularized Approximate Linear Programming in Noisy Domains

As discussed in the preceding chapter, the existence of a large amount of noise in the MDP can be a source of approximation error. This error is bounded, as demonstrated in Subsection 6.3.2, but can nonetheless be problematic.

The intuition behind our solution to this source of error is that while one sample may be inaccurate, there are likely additional samples nearby. This group of samples, *on average*, behaves as if we were taking samples with expectation. Therefore, our contribution is a simple addition of a smoothing function to our Bellman constraints from Equation 6.2.

This chapter begins with a discussion of previous use of smoothers and averagers in Value Function Approximation in Section 7.1, followed by the introduction of our approach, Locally Smoothed  $L_1$ -Regularized Approximate Linear Programming (LS-RALP) in Section 7.2. Section 7.3 presents a proof that error from noise approaches zero as the number of sampled data approaches infinity, while Section 7.4 demonstrates the approach improves results dramatically with even a very small number

of samples.

## 7.1 Smoothers and Averagers for Value Function Approximation

We begin by introducing smoothing functions. A smoothing function spreads the contribution of each sampled data point over its local neighborhood. The simplest smoothers are called averagers, and are denoted  $k_b(\dot{\sigma}, \dot{\sigma}_i)$ ; the size of the neighborhood is defined by the bandwidth parameter  $b$ . Averagers stipulate that  $\sum_{\dot{\sigma}_i \in \dot{\Sigma}} k_b(\dot{\sigma}, \dot{\sigma}_i) = 1$  for all  $\dot{\sigma}$ , and that  $k_b(\dot{\sigma}, \dot{\sigma}_i) \geq 0$  for all  $\dot{\sigma}, \dot{\sigma}_i$ ; they can therefore be thought of as weighted averages, where weights are a function of the samples' proximity and the bandwidth parameter. Smoothing functions are commonly used to perform nonparametric regression; as sampled data are added, the bandwidth parameter is tuned to shrink the neighborhood, to allow for a balance between the bias from including increasingly irrelevant samples which are farther away and the variance from having too few samples in the neighborhood.

Smoothers were first applied to reinforcement learning by Gordon (1995), who demonstrated that value function approximations with averagers would converge. He did this by demonstrating that regression using averagers was *compatible* with value iteration, and so alternating this regression with a value backup step would result in a contraction, and therefore, convergence. He was also able to bound the error in the resulting approximation.

Ormoneit and Sen (2002) took the ideas behind nonparametric regression and applied them directly to value iteration. They approximated  $TV$  with

$$T_a V(s) \approx \sum_{\dot{\sigma} \in \dot{\Sigma}} k_b(\dot{\sigma}^s, s, a) \left[ \dot{\sigma}^r + \gamma V(\dot{\sigma}^{s'}) \right],$$

where  $k_b(\dot{\sigma}^s, s, a) = 0$  if  $\dot{\sigma}^a \neq a$ . They were again able to prove convergence, as well as define an optimal shrinkage rate for the bandwidth parameter.

These works were important in that they provided helpful solution guarantees. However, they contain some inherent drawbacks. First, in nonparametric approximations with smoothers, extending the approximation to a novel state can be slow, as it is necessary to calculate kernel values for all sampled data; in parametric approximations, it is only necessary to calculate the likely smaller number of feature values. Second, while all of machine learning struggles without sufficient data, this is perhaps even more true of nonparametric approximations. Because approximation weights are defined by proximity to samples, areas with low proximity to all samples are likely to be extremely inaccurate. In comparison, features in parametric approximation can be designed to provide support throughout the space. Fortunately, we can leverage the convergence and averaging properties of smoothing functions to handle noise in RALP.

## 7.2 Locally Smoothed $L_1$ -Regularized Approximate Linear Programming

To address RALP’s vulnerability to noise, demonstrated in Section 6.3, we introduce Locally Smoothed  $L_1$ -Regularized Approximate Linear Programming (LS-RALP). The concept behind LS-RALP is simple; by averaging between nearby, similar samples, we can smooth out the effects of noise and produce a more accurate approximation. We define “nearby” to mean two things: one, the states should be close to each other in the state space, and two, the actions taken should be identical. The goal is to achieve the performance of RALP with expected samples with the more realistic simple samples.

For our application, we define  $k_b(\dot{\sigma}, \dot{\sigma}_i)$  such that if  $\dot{\sigma}_i^a \neq \dot{\sigma}^a$ , the function returns zero.

**Observation 7.2.1.** *Consider the regression problem of estimating the function  $f$  of the response variable  $y = f(x) + \epsilon$ , given  $n$  observations  $(x_i, y_i)(i = 1, \dots, n)$ ,*

where  $\epsilon$  represents mean 0 noise. Assume  $f(x)$  is continuously differentiable, and that samples  $x_i$  are sampled uniformly. For a given smoothing function  $k_b(x, x_i)$ , there exists a bandwidth function  $b(n)$  such that as  $n \rightarrow \infty$ ,  $\sum_i k_{b(n)}(x, x_i)y_i \rightarrow f(x)$ .

In particular, we point out that this trait of consistency is true for kernel estimators (Silverman, 1978; Hwang et al., 1994),  $k$ -nearest-neighbor averagers (Mack, 1981), and smoothing splines (Rice and Rosenblatt, 1981). This literature on non-parametric regression also contains extensive theory on optimal shrinkage rates for the bandwidth parameter; in practice, however, for a given number of samples, this is commonly done by cross-validation.

Similar results exist for cases without uniform sampling (e.g. the results of Silverman (1984) and Jennen-Steinmetz and Gasser (1988)), but this needlessly complicates the analysis.

We now modify our LP to include our smoothing function:

$$\begin{aligned} \min_{\mathbf{w}} \quad & \rho^T \Phi \mathbf{w} \\ \text{s.t.} \quad & \sum_{\dot{\sigma}_i \in \dot{\Sigma}} k_b(\dot{\sigma}, \dot{\sigma}_i) \dot{T}_{\dot{\sigma}_i} \Phi(\dot{\sigma}_i^s) \mathbf{w} \leq \Phi(\dot{\sigma}_i^s) \mathbf{w} \quad \forall \dot{\sigma} \in \dot{\Sigma} \\ & \|\mathbf{w}\|_{1,e} \leq \psi. \end{aligned} \tag{7.1}$$

In this formulation, we are using our smoothing function to estimate the constraint with expectation  $T\Phi(\sigma^s)w$  by smoothing across our easily obtained  $\dot{T}_{\dot{\sigma}}\Phi(\dot{\sigma}^s)w$ .

Note that RALP is a special case of LS-RALP, where the bandwidth of the smoothing function is shrunk until the function is a delta function. Therefore, LS-RALP, with correct bandwidth choice, can always do at least as well as RALP.

### 7.3 Theoretical Results

The goal of this section is to demonstrate that the application of a smoothing function to the constraints of the Estimated ALP will mitigate the effects of noise. In particular, we show that as the number of samples approaches infinity, the LS-RALP

solution using simple samples approaches the RALP solution using samples with expectation.

We now introduce and prove our theorem. We denote the number of samples  $|\dot{\Sigma}|$  as  $n$ .

**Theorem 7.3.1.** *If the reward function, transition function, and features are continuously differentiable, and samples are drawn uniformly from the state space, there exists a bandwidth function  $b(n)$  such that as  $n \rightarrow \infty$ , the LS-RALP solution using one-step samples approaches the Sampled RALP solution.*

*Proof.* We begin by restating the definition of  $\epsilon_s$ :

$$\epsilon_s = \max_{\dot{\sigma} \in \dot{\Sigma}} \left| T\Phi(\dot{\sigma}^s)\mathbf{w} - \dot{T}_{\dot{\sigma}}\Phi(\dot{\sigma}^s)\mathbf{w} \right|.$$

We then expand upon the portion within the max operator for some arbitrary  $\dot{\sigma}$ :

$$\begin{aligned} & T\Phi(\dot{\sigma}^s)\mathbf{w} - \dot{T}_{\dot{\sigma}}\Phi(\dot{\sigma}^s)\mathbf{w} \\ &= \left[ R(\dot{\sigma}^s) + \gamma \sum_{s' \in \mathcal{S}} P(s'|\dot{\sigma}^s, \dot{\sigma}^a)\Phi(s')\mathbf{w} \right] - \left[ \dot{\sigma}^r + \gamma\Phi(\dot{\sigma}^{s'})\mathbf{w} \right] \\ &= R(\dot{\sigma}^s) - \dot{\sigma}^r + \gamma \left( \mathbb{E} \left[ \sum_{s' \in \mathcal{S}} P(s'|\dot{\sigma}^s, \dot{\sigma}^a)\Phi(s')\mathbf{w} \right] - \Phi(\dot{\sigma}^{s'})\mathbf{w} \right). \end{aligned}$$

We now introduce the smoothing function.

$$\begin{aligned} & T\Phi(\sigma^s)\mathbf{w} - \dot{T}_{\dot{\sigma}}\Phi(\dot{\sigma}^s)\mathbf{w} \\ &= R(\dot{\sigma}^s) - \sum_{\dot{\sigma}_i \in \dot{\Sigma}} k_b(\dot{\sigma}, \dot{\sigma}_i)\dot{\sigma}_i^r \\ & \quad + \gamma \mathbb{E} \left[ \sum_{s' \in \mathcal{S}} P(s'|\dot{\sigma}^s, \dot{\sigma}^a)\Phi(s')\mathbf{w} \right] - \gamma \sum_{\dot{\sigma}_i \in \dot{\Sigma}} k_b(\dot{\sigma}, \dot{\sigma}_i)\Phi(\dot{\sigma}_i^{s'})\mathbf{w} \end{aligned}$$

It is clear our smoothing function is filling the role of a predictive function on the reward and next feature value functions; the first two terms are the difference between the expected reward and the predicted reward, while the second two are the difference between the expected next value and the predicted next value. The more accurate our regression, the smaller  $\epsilon_s$  will be.

It follows from Observation 7.2.1 that as  $n$  increases, if  $R(s)$ ,  $P(s'|s, a)$ , and  $\Phi(s)$  are all continuously differentiable with respect to the state space, there exists a bandwidth shrinkage function  $b(n)$  such that  $\epsilon_s$  will approach 0.

When  $\epsilon_s = 0$ , the smoothed constraints from one-step samples of LS-RALP are equivalent to the constraints with expectation of the Sampled RALP.  $\square$

This theorem shows that sample noise, the weak point of all ALP algorithms, can be addressed by smoothing in a principled manner. Even though these results address primarily the limiting case of  $n \rightarrow \infty$ , our experiments demonstrate that in practice we can obtain vastly improved value functions even with very few samples.

We note the smoothness assumptions on  $R(s)$ ,  $P(s'|s, a)$ , and  $\Phi(s)$  may not always be realistic, and are stronger than the bounded change assumptions made by RALP in Assumption 6.3.4. However, we will demonstrate in the experimental section that meeting these assumptions is not necessary for the method to be effective.

## 7.4 Experimental Results

In this section, we apply LS-RALP to noisy versions of common Reinforcement Learning benchmark problems to demonstrate the advantage of smoothing. We will use two domains, the inverted pendulum (Wang et al., 1996) and the mountain-car (Sutton and Barto, 1998). While we have proved that LS-RALP will improve upon the value function approximation of RALP as the number of samples approaches infinity, our experiments demonstrate that there is significant improvement even with a

relatively small number of samples. In both problems, we approximate the value function, and then apply the resulting greedy policy. We generate the policy using a model; at each state, we evaluate each candidate action, and calculate the approximate value at the resulting state. The action which resulted in the highest value is then chosen in the trajectory.

We note that the smoothing function could be used as a form of approximate model by using it to estimate the right hand side of the Bellman equation for arbitrary state action pairs, then choosing the action with the highest estimated value. In principle this would eliminate the need for a model for action selection. In practice, the parametric nature of the value function approximation led to good approximate value functions over large regions of the state space even when the samples were too sparse to provide a reliable approximate model for all states that might be encountered during policy evaluation. Since our emphasis is on value function approximation, we used a model for action selection and defer the use of smoothing for action selection to future work.

*Inverted Pendulum* This is the same inverted pendulum problem used in the experiments using RALP in Subsection 6.4.2, with one change. Usually, the noise in the pendulum problem is applied as Gaussian noise of standard deviation 10 on the force applied to the table; however, as estimated RALP already handled this amount of noise gracefully, we made the problem more interesting by increasing the standard deviation to 20. This makes the problem significantly more difficult, as even with an optimal policy, an unlucky agent can fail to balance the pendulum.

We compare RALP and LS-RALP, using the same set of features; Gaussian kernels of three different widths, a polynomial kernel of degree 3,  $s[1], s[2], s[1] * s[2], s[1]^2, s[2]^2$ , and a constant feature, where  $s[1]$  is the angle, and  $s[2]$  is the angular velocity. For a number of samples  $n$ , this was therefore a total of  $4n + 6$  features.

As a reference point for the difficulty of the problem, we also compare against LSPI, using the radial basis functions used in Lagoudakis and Parr (2003).

Sampling was done by running 50 trajectories under a random policy, until the pendulum fell over (around six steps). For the LP methods, the regularization parameter was set to 1.2, and for LS-RALP, the smoothing function was a multivariate Gaussian kernel with a standard deviation covering approximately 1/30 of the state space in each direction. Both of these parameters were chosen by cross validation. 100 trials were run for each method, and the number of successful steps averaged; if a trial reached 3000 steps, it was terminated at that point. Results are presented in Figure 7.1. We note that performance on this difficult domain did not noticeably improve if more than 50 trajectories were used to collect data.

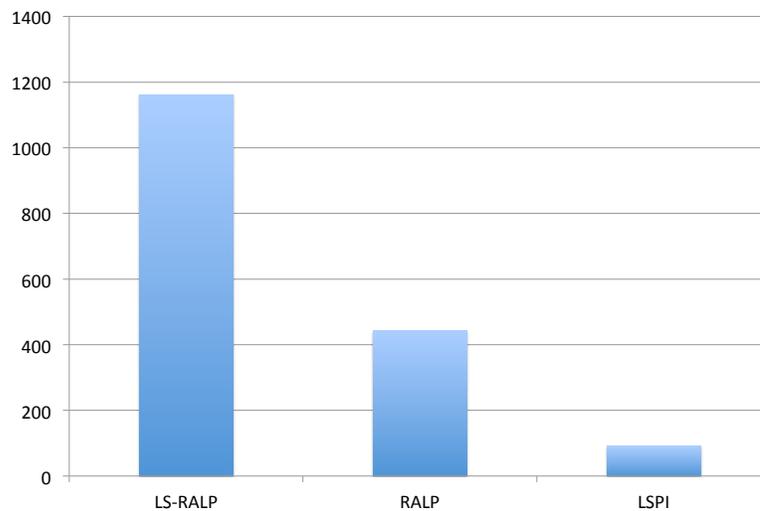


FIGURE 7.1: Average steps to failure over 100 trials with samples from 50 trajectories

*Mountain Car* In the mountain car problem, an underpowered car must climb a hill by gaining momentum. The state space is two-dimensional (position and velocity), the action space is three-dimensional (again, push, pull, or coast), and the discount factor is 0.99. Traditionally, a reward of  $-1$  is given for any state that is not in the

goal region, and a reward of 0 is given for any state that is. For this problem we applied Gaussian noise with standard deviation .5 to the reward to demonstrate the ability of LS-RALP to handle noisy rewards.

Features are the distributed radial basis functions suggested by Kolter and Ng (2009); however, because LSPI with LARS-TD requires a distinct set of basis functions for each action, while RALP methods do not, our dictionary is one-third of the size. For sake of a baseline comparison, we note that in the deterministic version of this problem, we repeated experiments by Kolter and Ng (2009) and Johns et al. (2010) on LARS-TD and LC-MPI, respectively and RALP outperformed both.

Sample trajectories are begun at a random state, and allowed to run with a random policy for 20 steps, or until the goal state is reached. After the value function is calculated, it was tested by placing the car in a random state; the policy then had 500 steps for the car to reach the goal state. Figure 7.4 contains the percentage completion for both RALP and LS-RALP over 50 trials. The regularization parameter for the RALP methods was 1.4, and the smoothing function for LS-RALP was a multivariate Gaussian kernel with a standard deviation spanning roughly 1/85 of the state space in both dimensions. These parameters were chosen using cross validation.

LS-RALP	40%
RALP	22%

Table 7.1: Percentage of 50 trials successful over 500 steps with 100 training trajectories

This chapter demonstrates that the introduction of a smoothing function can mitigate the effects of noise on RALP. Furthermore, we proved that as the amount of data is increased, error from inaccurate samples approaches zero. However, our experiments show drastic improvement with only a small number of samples.

In fairness, we also point out some limitations of our approach: LS-RALP makes stronger smoothness assumptions about the model and features than does RALP

(continuous differentiability vs. bounded change given samples). LS-RALP also requires an additional parameter that must be tuned - the kernel bandwidth. We note, however, that there is a rich body of work in non-parametric regression that may give practical guidance on the selection of the bandwidth parameter.

The nonparametric regression literature notes that the bandwidth of a smoother can be adjusted not just for the total number of data points, but also for the density of data available in the particular neighborhood of interest (Jennen-Steinmetz and Gasser, 1988). The addition of this type of adaptiveness might not only increase accuracy in areas of unusual density, but could also increase the accuracy of model approximation for policy generation. While LS-RALP currently approximates value function without unrealistic sampling assumptions, this addition would extend that ability all the way through policy generation.

## Future Work

This document has demonstrated that choosing a set of features *a priori* is much more difficult than performing automated feature selection from a large dictionary. As a result, we believe the most fruitful future work stems from this automated approach. However, there are several unanswered questions and scenarios which are poorly addressed by  $L_1$  regularization.

The first question pertains to some preliminary experiments which have not behaved as expected. A value function is only useful if it is an intermediate step towards the generation of a policy. It seemed an obvious extension to apply the principles of RALP to Q-functions, by concatenating a continuous action space with the state space, and approximating the function over the whole, augmented space. That is, we define features not on the state  $s$ , but on the state  $[s a]$ . So, constraints were of the form

$$\dot{\sigma}^r + \gamma \Phi([\dot{\sigma}^{s'} a']) \mathbf{w} \leq \Phi([\dot{\sigma}^s \dot{\sigma}^a]) \mathbf{w},$$

for several choices of  $a'$ . For reasons that are unclear, results were poor. This raises the question of what qualities of an action space are poorly suited to RALP.

Second, the analysis in the document expects nearby states to have similar feature

values. We have performed no analysis or experimentation using features which are discreet. It may at times be convenient to consider binary features on some aspect of the state space; it is not clear that forcing a smooth, continuous approximator on a discreet feature space will always make sense or perform well.

Third, we note that Chapters 6 and 7 have not made the chapters that precede them obsolete. While the “just add more” method of feature generation has been effective in our experiments on benchmark problems, this will inevitably have its limits in more complicated spaces. A possible solution is to consider the analysis of the earliest chapters of this document. For example, if we consider the two-room problem illustrated in Figure 5.3, it may make sense to use PVFs to capture the transition space around the walls, while using Gaussian kernels to express the reward function. While the analysis of Section 5.1 doesn’t apply to non-fixed-point approximations, it seems likely the intuitions would transfer.

While this interaction would be an interesting study on its own, it would likely also provide illumination on the RALP solution. As an illustration, Sutton (1988) demonstrated the effectiveness of  $TD(\lambda)$  well before Tsitsiklis and Van Roy (1997) described the solution as the linear fixed point. Additionally, there is discussion on the relative merits of the linear fixed-point solution compared to other methods (see, for example, the discussion by Scherrer (2010)); the RALP solution should now be included in these discussions. For RALP, we have demonstrated effectiveness, and have shown it cannot be too wrong, but we cannot describe much about why the solution is so effective when other solutions may be less so. This understanding would explain the gap between our extremely satisfying experimental results and our loose theoretical promises.

Fourth, we have performed some interesting experiments in a more complicated problem, the helicopter domain, in which the controller tried to keep a simulated helicopter in a stable hover on a windy day. The challenge of the helicopter problem

is that if sampling is done with a random policy, the vast majority of sampled states are already “failure” states, in that the helicopter has quickly lost equilibrium and is falling. The value functions in this domain were unimpressive and resulted in the helicopter staying aloft for only a short period of time. One solution to this problem is to use apprenticeship learning, in which samples are drawn by a human operator who keeps the helicopter in the interesting portion of the state space (Abbeel and Ng, 2005). Another solution may be to recognize states in the interesting portion of the space, and tune the  $\rho$  weighting vector to encourage accurate value functions at those states, at the expense of other, less interesting states. It has been shown in the full ALP that tuning  $\rho$  can have a large effect (de Farias and Van Roy, 2003); it would be interesting to explore the practical effects of this.

Finally, the RALP itself may suggest good features to be added to the dictionary. Extremely loose constraints indicate areas in which the chosen active feature set is unable to adequately represent the value function at those states. The addition of these gaps as features suggest an iterative approach to feature generation for an improved RALP approximation. This may be extremely useful in complex state spaces for gaining appropriate expressiveness without using human intelligence in feature dictionary construction.

## Summary and Conclusions

This document reflects my own changing attitudes towards linear value function approximation. It is clear the feature selection problem is at the crux of linear value function approximation; while there is disagreement as to how to best weight features (see, for example, the discussion by Scherrer (2010), or the newly relevant question of the LP solution as opposed to the fixed-point solution), the much harder question is in constructing the linear space from which to pluck a value function.

### 9.1 Linear Analysis

Chronologically, the work of Chapter 3 and Section 5.1 occurred first; we showed that regardless of the feature set, model-free fixed-point approximations were equivalent to a model-based approximation. We also showed that the Bellman error could be broken down into a sum of reward and transition error. The experiments related to the last contribution were interesting for me. The proto-value function work by Mahadevan and Maggioni (2006) was the first reinforcement learning paper that really got me excited; the construction of features on a graph made from sample trajectories struck me as a brilliant way to make features tailor-made for the problem.

The fact that PVFs often don't adequately cover the reward function showed me that despite the great benefits of the compact nature of linear approximation, the existing approach of *a priori* feature selection was a tremendously difficult one, and potentially not tenable.

## 9.2 Kernel Analysis

The next work, which appears in Chapter 4 and Section 5.2, was an attempt to sidestep this problem with kernelized methods. We showed a similar set of results for kernelized approximations to those that we showed for linear approximations. The first was that previous work in kernelized value function approximation was equivalent but for their approaches to regularization. It was interesting that this was where the differences lay, as none of the previous work discussed regularization in any great detail; it was often included solely to allow the inversion of an otherwise ill-conditioned kernel matrix. We also demonstrated a Bellman error breakdown similar to the one we showed in linear approximation; again, a kernel matrix must allow for coverage of both the reward and transition functions of the system. This is much easier to achieve with kernelized methods due to the great expressiveness of kernelized approaches; however, the negative of this expressiveness is in a tendency to overfit. As a result, our experiments focuses on using the Bellman error breakdown to tune regularization parameters and control the overfitting.

Besides overfitting, the other problem with kernelized methods is in computational cost. The construction of a value function from  $n$  samples takes  $O(n^3)$  time, due to the inversion of the kernel matrix. While this may not be a huge problem for batch reinforcement learning, it rules out real-time applications. In addition, extending this value function to each novel state costs  $O(n)$  time; for linear approximations, this is a likely much cheaper  $O(k)$  time, where  $k$  is the number of features.

However, I believe there is some unexplored promise in the area of using the

variance in the Gaussians returned by GPTD as an expression of confidence; this can be used to guide sampling or even potentially as information for kernel tuning. Additionally, the use of kernel functions as linear features offers a great deal of promise; when kernels are viewed as similarity functions, they can be powerful and intuitive features (“I have seen good states and bad states; if a state is similar to a good state, I conclude it is good, if it is similar to a bad state, I conclude it is bad”).

### 9.3 Automatic Feature Selection

This last impression fed well into the RALP work of Chapter 6, in that we have an unnecessarily large set of potentially useful features: which ones are most useful? The idea by Kolter and Ng (2009) to apply the ideas of Tibshirani (1996) to value function approximation opened the door to automated feature selection, giving the ability to choose such a set of features from a kernel matrix or other large dictionary of features.

When we applied  $L_1$  regularization to the ALP, we were able to show several important theoretical results. First, we showed that we could extend ALP to continuous or large state spaces, because error from missing constraints was bounded. In fact, we showed that despite not only missing constraints, but also noisy data, and tremendously large feature sets, we could bound the error in the approximation.

More importantly, though, we also showed that empirically, the value functions from RALP were superior to those generated by other methods; they performed extremely well, despite using virtually no human intuition in the construction of the feature dictionary. Previously, a great deal of effort was put into designing feature sets like PVFs and Krylov basis functions designed to model some aspect of the problem; we demonstrated this approach was a difficult one, particularly if  $P$  and  $R$  are not known. By offering a huge number of options to the solver, we can overcome the need for these types of feature sets, overwhelming the need for quality with

quantity.

Finally, we addressed the issue of noise in data as RALP was particularly poorly suited to handle this noise. By applying a smoothing function to the constraints of the RALP, we were able to overcome this problem. We first showed that, given an infinite amount of data, we could guarantee no error from noise. Much more importantly, though, we also showed empirically that this benefit existed even for very small amounts of data, and presented good performance even in extremely noisy environments.

Linear value function approximation can be a powerful, useful tool, as long as the field continues to attack the problem on two fronts. First, it is clear features must be chosen automatically; developing better ways to do this and understanding the solutions that result allow us to remove human intelligence from the loop. Second, we must continue to look at the feature set options we provide and understand their characteristics; as we try larger and more complex problems, it is unclear that simple features will provide sufficient coverage and expressiveness. Fortunately, these areas continue to be areas of substantial research, and strides will continue to be made.

# Bibliography

- Abbeel, P. and Ng, A. (2005), “Exploration and Apprenticeship Learning in Reinforcement Learning,” in *Proceedings of the 22nd International Conference on Machine Learning*.
- Aizerman, A., Braverman, E., and Rozoner, L. (1964), “Theoretical Foundations of the Potential Function Method in Pattern Recognition Learning,” *Automation and Remote Control*, 25, 821–837.
- Bertsekas, D. P. and Ioffe, S. (1996), “Temporal Differences-Based Policy Iteration and Applications in Neuro-Dynamic Programming,” Tech. rep., MIT, LIDS Technical Report LIDS-P-2349.
- Bishop, C. M. (2006), *Pattern Recognition and Machine Learning*, Springer.
- Boyan, J. A. (1999), “Least-Squares Temporal Difference Learning,” in *In Proceedings of the Sixteenth International Conference on Machine Learning*, pp. 49–56, Morgan Kaufmann.
- Boyan, J. A. and Moore, A. (1995), “Generalization in Reinforcement Learning: Safely Approximating the Value Function,” in *Neural Information Processing Systems*, pp. 369–376.
- Bradtke, S. J. and Barto, A. G. (1996), “Linear Least-Squares Algorithm for Temporal Difference Learning,” *Machine Learning*, 22, 33–57.
- Dayan, P. (1992), “The Convergence of TD( $\lambda$ ) for General  $\lambda$ ,” *Machine Learning*, 8, 341–362.
- de Farias, D. P. and Van Roy, B. (2003), “The Linear Programming Approach to Approximate Dynamic Programming,” *Operations Research*.
- de Farias, D. P. and Van Roy, B. (2004), “On Constraint Sampling for the Linear Programming Approach to Approximate Dynamic Programming,” *Mathematics of Operations Research*, pp. 462–478.
- d’Epenoux, F. (1963), “A Probabilistic Production and Inventory Problem,” *Management Science*.

- Desai, V. V., Farias, V. F., and Moallemi, C. C. (2009), “A Smoothed Approximate Linear Program,” in *Advances in Neural Information Processing Systems*, vol. 22.
- Efron, B., Hastie, T., Johnstone, I., and Tibshirani, R. (2004), “Least Angle Regression,” *Annals of Statistics*, pp. 407–451.
- Engel, Y., Mannor, S., and Meir, R. (2005), “Reinforcement Learning with Gaussian Processes,” in *Machine Learning-International Workshop then Conference*, vol. 22, pp. 201–208.
- Ernst, D., Geurts, P., and Wehenkel, L. (2006), “Tree-Based Batch Mode Reinforcement Learning,” *Journal of Machine Learning Research*, 6, 503–557.
- Farahmand, A. M., Ghavamzadeh, M., Szepesvári, C., and Mannor, S. (2008), “Regularized Policy Iteration,” in *Advances in Neural Information Processing Systems*, vol. 21, pp. 441–448.
- Girard, A., Rasmussen, C. E., Candela, J. Q., and Murray-Smith, R. (2003), “Gaussian Process Priors with Uncertain Inputs-Application to Multiple-Step Ahead Time Series Forecasting,” in *Advances in Neural Information Processing Systems*, pp. 545–552.
- Gordon, G. (1995), “Stable Function Approximation in Dynamic Programming,” in *International Conference of Machine Learning*, pp. 261–268.
- Hwang, J.-N., Lay, S.-R., and Lippman, A. (1994), “Nonparametric Multivariate Density Estimation: A Comparative Study,” *IEEE Transactions on Signal Processing*, 42, 2795–2810.
- Jennen-Steinmetz, C. and Gasser, T. (1988), “A Unifying Approach to Nonparametric Regression Estimation,” *Journal of the American Statistical Association*, 83, 1084–1089.
- Johns, J., Painter-Wakefield, C., and Parr, R. (2010), “Linear Complementarity for Regularized Policy Evaluation and Improvement,” in *Advances in Neural Information Processing Systems 23*, eds. J. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, pp. 1009–1017.
- Keller, P. W., Mannor, S., and Precup, D. (2006), “Automatic Basis Function Construction for Approximate Dynamic Programming and Reinforcement Learning,” in *Proceedings of the 23rd international conference on Machine learning*, p. 456, ACM.
- Kolter, J. Z. and Ng, A. (2009), “Regularization and Feature Selection in Least-Squares Temporal Difference Learning,” in *Proceedings of the 26th International Conference on Machine Learning*, eds. L. Bottou and M. Littman, pp. 521–528, Montreal, Canada, Omnipress.

- Lacey, K., Zaharia, M., Griffiths, J., Ravindran, A., Merali, Z., and Anisman, H. (2000), “A Prospective Study of Neuroendocrine and Immune Alterations Associated With the Stress of an Oral Academic Examination Among Graduate Students,” *Psychoneuroendocrinology*, 25, 339–356.
- Lagoudakis, M. G. and Parr, R. (2003), “Least-Squares Policy Iteration,” *The Journal of Machine Learning Research*.
- Mack, Y. P. (1981), “Local Properties of  $k$ -NN Regression Estimates,” *SIAM Journal on Algebraic and Discrete Methods*, 2, 311–323.
- Mahadevan, S. and Maggioni, M. (2006), “Proto-value Functions: A Laplacian Framework for Learning Representation and Control in Markov Decision Processes,” Tech. rep., University of Massachusetts.
- Mallat, S. and Zhang, Z. (1993), “Matching Pursuit With Time-Frequency Dictionaries,” in *IEEE Transactions on Signal Processing*, vol. 41, pp. 3397–3415.
- Mercer, J. (1909), “Functions of Positive and Negative Type, and their Connection with the Theory of Integral Equations.” *Philosophical Transactions of the Royal Society of London*, 209, 415–446.
- Ormoneit, D. and Sen, S. (2002), “Kernel-based reinforcement learning,” *Machine Learning*, 49, 161–178.
- Parr, R., Painter-Wakefield, C., Li, L., and Littman, M. (2007), “Analyzing Feature Generation for Value-Function Approximation,” in *Proceedings of the 24th International Conference on Machine Learning*, pp. 744–751.
- Parr, R., Li, L., Taylor, G., Painter-Wakefield, C., and Littman, M. (2008), “An Analysis of Linear Models, Linear Value-Function Approximation, and Feature Selection for Reinforcement Learning,” in *International Conference of Machine Learning*, pp. 752–759, ACM New York, NY, USA.
- Petrik, M. (2007), “An Analysis of Laplacian Methods for Value Function Approximation in MDPs,” in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 2574–2579.
- Petrik, M. and Zilberstein, S. (2009), “Constraint Relaxation in Approximate Linear Programs,” in *Proceedings of the 26th International Conference on Machine Learning*, eds. L. Bottou and M. Littman, Montreal, Canada, Omnipress.
- Petrik, M., Taylor, G., Parr, R., and Zilberstein, S. (2010), “Feature Selection Using Regularization in Approximate Linear Programs for Markov Decision Processes,” in *Proceedings of the 27th International Conference on Machine Learning*.

- Randløv, J. and Alstrøm, P. (1998), “Learning to Drive a Bicycle using Reinforcement Learning and Shaping,” in *Proceedings of the 15th International Conference on Machine Learning*, pp. 463–471.
- Rasmussen, C. E. and Kuss, M. (2004), “Gaussian Processes in Reinforcement Learning,” in *Advances in Neural Information Processing Systems*, vol. 16, pp. 751–759.
- Rice, J. and Rosenblatt, M. (1981), “Integrated Mean Square Error of a Smoothing Spline,” *Journal of Approximation Theory*, 33, 353–369.
- Sanner, S. and Boutilier, C. (2005), “Approximate Linear Programming for First-Order MDPs,” in *Uncertainty in Artificial Intelligence*, pp. 509–517.
- Scherrer, B. (2010), “Should One Compute the Temporal Difference Fix Point or Minimize the Bellman Residual? The Unified Oblique Projection View,” in *Proceedings of the 27th International Conference on Machine Learning*.
- Schoknecht, R. (2002), “Optimality of Reinforcement Learning Algorithms with Linear Function Approximation,” in *Advances in Neural Information Processing Systems 15*, pp. 1555–1562.
- Schweitzer, P. J. and Seidmann, A. (1985), “Generalized Polynomial Approximations in Markovian Decision Processes,” *Journal of mathematical analysis and applications*, 110, 568–582.
- Silverman, B. W. (1978), “Weak and Strong Uniform Consistency of the Kernel Estimate of a Density and its Derivatives,” *The Annals of Statistics*, 6, 177–184.
- Silverman, B. W. (1984), “Spline Smoothing: The Equivalent Variable Kernel Method,” *The Annals of Statistics*, pp. 898–916.
- Sutton, R. S. (1988), “Learning to Predict by the Methods of Temporal Differences,” *Machine Learning*.
- Sutton, R. S. and Barto, A. G. (1998), *Reinforcement Learning: an Introduction*, MIT Press.
- Sutton, R. S., Szepesvári, C., Geramifard, A., and Bowling, M. H. (2008), “Dyna-Style Planning with Linear Function Approximation and Prioritized Sweeping,” in *Proceedings of the 24th Conference in Uncertainty in Artificial Intelligence*, pp. 528–536.
- Szepesvári, C. (2010), *Algorithms for Reinforcement Learning*, Morgan and Claypool.
- Taylor, G. and Parr, R. (2009), “Kernelized Value Function Approximation for Reinforcement Learning,” in *Proceedings of the 26th International Conference on Machine Learning*, eds. L. Bottou and M. Littman, pp. 1017–1024, Montreal, Canada, Omnipress.

- Tibshirani, R. (1996), “Regression Shrinkage and Selection via the Lasso,” *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 267–288.
- Tsitsiklis, J. N. and Van Roy, B. (1996), “Feature-Based Methods for Large Scale Dynamic Programming,” *Machine Learning*, 22, 59–94.
- Tsitsiklis, J. N. and Van Roy, B. (1997), “An Analysis of Temporal-Difference Learning With Function Approximation,” *IEEE Transactions on Automatic Control*, 42, 674–690.
- Wang, H. O., Tanaka, K., and Griffin, M. F. (1996), “An Approach to Fuzzy Control of Nonlinear Systems: Stability and Design Issues,” *IEEE Transactions on Fuzzy Systems*, 4, 14–23.
- Williams, R. J. and Baird, L. C. (1993), “Tight Performance Bounds on Greedy Policies Based on Imperfect Value Functions,” Tech. Rep. NU-CCS-93-14, Northeastern University.
- Wu, J.-H. and Givan, R. (2004), “Feature-Discovering Approximate Value Iteration Methods,” Tech. Rep. TR-ECE-04-06, Purdue University.
- Xu, X., Xie, T., Hu, D., and Lu, X. (2005), “Kernel Least-Squares Temporal Difference Learning,” in *International Journal of Information Technology*, vol. 11, pp. 54–63.
- Yu, H. and Bertsekas, D. P. (2006), “Convergence Results for Some Temporal Difference Methods Based on Least Squares,” *Lab. for Information and Decision Systems Report*, 2697.

# Biography

Gavin Taylor was born in Illinois in 1983 and grew up in St. Louis. His first moment of demonstrated computer aptitude was in slapping the monitor of an Apple IIe in his elementary school classroom to improve the picture. After graduating from Parkway West High School in 2002, he went on to Davidson College in Davidson, NC, where he would do his first research, both in Mathematics and Computer Science. He obtained his B.S. degree in Mathematics in 2006.

Gavin knew he eventually wanted to teach as academic faculty, and so pursued a Ph.D. He chose to do this Ph.D. in Computer Science so that his math background would be applied to real problems.

At Duke, Gavin earned a Certificate of Excellence in Teaching and a Departmental Service Award. Aside from the publications which make up the contributions in this dissertation, he also pursued his interest in undergraduate education. He obtained an M.S. in 2009 and graduated with a Ph.D. in May of 2011.