Sparse Value Function Approximation for Reinforcement Learning

by

Christopher Painter-Wakefield

Department of Computer Science
Duke University

Date: _____

Approved:

_____
Ronald Parr, Supervisor

_____
Vincent Conitzer

_____
Kamesh Munagala

_____
Lawrence Carin

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
2013

Abstract

Sparse Value Function Approximation for Reinforcement Learning

by

Christopher Painter-Wakefield

Department of Computer Science
Duke University

Date: _____

Approved:

_____
Ronald Parr, Supervisor

_____
Vincent Conitzer

_____
Kamesh Munagala

_____
Lawrence Carin

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
2013

Abstract

A key component of many reinforcement learning (RL) algorithms is the approximation of the value function. The design and selection of features for approximation in RL is crucial, and an ongoing area of research. One approach to the problem of feature selection is to apply sparsity-inducing techniques in learning the value function approximation; such sparse methods tend to select relevant features and ignore irrelevant features, thus automating the feature selection process. This dissertation describes three contributions in the area of sparse value function approximation for reinforcement learning.

One method for obtaining sparse linear approximations is the inclusion in the objective function of a penalty on the sum of the absolute values of the approximation weights. This L_1 regularization approach was first applied to temporal difference learning in the LARS-inspired, batch learning algorithm LARS-TD. In our first contribution, we define an iterative update equation which has as its fixed point the L_1 regularized linear fixed point of LARS-TD. The iterative update gives rise naturally to an online stochastic approximation algorithm. We prove convergence of the online algorithm and show that the L_1 regularized linear fixed point is an equilibrium fixed point of the algorithm. We demonstrate the ability of the algorithm to converge to the fixed point, yielding a sparse solution with modestly better performance than unregularized linear temporal difference learning.

Our second contribution extends LARS-TD to integrate policy optimization with

sparse value learning. We extend the L_1 regularized linear fixed point to include a maximum over policies, defining a new, "greedy" fixed point. The greedy fixed point adds a new invariant to the set which LARS-TD maintains as it traverses its homotopy path, giving rise to a new algorithm integrating sparse value learning and optimization. The new algorithm is demonstrated to be similar in performance with policy iteration using LARS-TD.

Finally, we consider another approach to sparse learning, that of using a simple algorithm that greedily adds new features. Such algorithms have many of the good properties of the L_1 regularization methods, while also being extremely efficient and, in some cases, allowing theoretical guarantees on recovery of the true form of a sparse target function from sampled data. We consider variants of orthogonal matching pursuit (OMP) applied to RL. The resulting algorithms are analyzed and compared experimentally with existing L_1 regularized approaches. We demonstrate that perhaps the most natural scenario in which one might hope to achieve sparse recovery fails; however, one variant provides promising theoretical guarantees under certain assumptions on the feature dictionary while another variant empirically outperforms prior methods both in approximation accuracy and efficiency on several benchmark problems.

For Patty, Ian, and Timothy.

Contents

List of Tables

List of Figures

Acknowledgements

Graduate school is a long journey, and I have been fortunate to have a lot of support. Thank you to my family and many friends, mentors, and collaborators who have helped me along my way.

To my wife, Patty Painter-Wakefield, a special thank you for your love, support and encouragement. Thank you for carrying the family through paper deadlines, teaching preparations, and all the other times I was unavailable, stressed, or just plain exhausted.

Thank you to my advisor, Ron Parr, for setting the example and teaching me how to do academic research. Thank you also for your patience these long years.

Thank you to my parents, who have been my cheerleaders, and to my siblings and assorted other relatives who have helped ground me.

I have been blessed with some extraordinary friends and colleagues during my time in graduate school, all of who have contributed spiritually if not materially to this dissertation. I especially treasure the time spent with Shashidhara Ganjugunte, Gavin Taylor, Mac Mason, Susanna Ricco, and Monika Schaeffer. Thank you to George Konidaris, for some perspective at a time when I much needed it.

I have been fortunate to work with many interesting and talented collaborators. Thank you to Michael Littman, Lihong Li, and Jeff Johns, and also to Susan Murphy and Eric Laber.

Finally, thank you to Duke University and to the agencies which have supported

me financially in my time as a graduate student. My work has been supported by grants from NSF (NSF-IIS-0209088, NSF-IIS-0713435, NSF-IIS-0546709, NSF-IIS-1147641, and NSF-IIS-1218931), DARPA (HR0011-07-1-0027), and DHS (HSHQDC-11-C-0083).

1

Introduction

Reinforcement learning is a branch of machine learning in which an agent learns from experience how to act to maximize rewards. For instance, the agent might be a robot navigating a maze; the robot needs to choose its actions carefully to avoid running into walls or getting stuck. A more complex example is an automated greenhouse, in which the goal is to grow a crop for the most profit; in this case the agent must balance the sale price of the crop against the cost of the resources used to produce the crop. The catch in both of these problems is that the agent does not know *a priori* what the outcomes of its actions will be and cannot therefore predict the most profitable sequence of actions to take; instead, the agent must explore and gather experience from which to learn a profitable action strategy.

While there are many approaches to reinforcement learning (RL), all of the work in this dissertation is based on the learning of a *value function*, which assigns a value to every situation, or *state*, in which an agent is or can be. The agent uses the value function to predict the worth of its various action choices, and make the choice which produces the most value. An accurate value function is thus of prime importance.

Unfortunately, value functions are rarely simple in form, especially for real world

problems; instead we approximate value functions as best as we are able. These approximations are dependent on the information about the current state which we can observe at any given time. We say that the observable *features* of the state (e.g., the position of the robot, or the temperature and humidity inside the greenhouse) form the support for our approximation.

The approximate value function, then, is like a building held up by columns; all of the columns support the building, but which ones are truly important? Are some of them decorative only? Can we remove some of them and still support the building? The choice of features (columns) used to support the value function is important. Frequently we want to know which features are relevant and which are not; and in some cases, we can "build" less expensively if we know which information is truly important.

We also are interested in avoiding situations which produce a bad value function by using too many features. This can happen when the information we have is incomplete or corrupted with random noise. With enough features we can reproduce the noise in our training data, causing us to *overfit*. When we overfit, we find that our value function performs poorly in practice, even though having extra features gave us a reduced error on our training data.

A primary interest of this document is *sparsity*, which is the quality of a solution (such as an approximate value function) which makes use of only the most relevant information. Irrelevant or superfluous features are ignored, and features of minor import may be discarded if they contribute little to the solution. Sparsity inducing techniques help determine the relevance of features, produce solutions less expensively, and prevent overfitting.

1.1 Document Organization and Contributions

In this dissertation we consider two of the more popular routes to sparsity, adopted from the supervised learning (regression) community. The use of L_1 regularization in regression problems has received particular interest and attention of late (Tibshirani, 1996; Efron et al., 2004; Daubechies et al., 2004; Osborne et al., 2000); more recently, L_1 regularization has been adapted for use with RL (Loth et al., 2007; Kolter and Ng, 2009). Another approach that has received renewed attention in the supervised learning community is that of using a simple algorithm that greedily adds new features (Tropp, 2004; Zhang, 2009); while a brief exploration of greedy techniques applied to RL was made by Johns (2010), little previous work exists on the topic.

Chapter 2 gives a detailed background on reinforcement learning, regression, and regularization. We follow with a brief overview of prior and related work in chapter 3.

In chapter 4 we explore the application of L_1-regularization to online reinforcement learning in work originally presented by Painter-Wakefield and Parr (2012b). We present a novel online algorithm and relate it to previous work on L_1 regularization in batch RL methods (Kolter and Ng, 2009; Johns et al., 2010). We also prove convergence of the online algorithm and demonstrate its convergence to a sparse, fixed point solution.

In chapter 5 we present an algorithm for simultaneous feature selection and optimization using L_1 regularization. This algorithm, LARQ, extends previous work by Kolter and Ng (2009) which applies L_1 regularization to policy evaluation only. The LARQ algorithm previously appears in work by Johns et al. (2010) as motivation for a modified policy iteration algorithm approximating the LARQ solution path. Here we provide the derivation of the LARQ algorithm, analysis, and experiments not in

the Johns et al. work.

Chapter 6 contributes to the theoretical and practical understanding of greedy techniques for RL, in work previously presented by Painter-Wakefield and Parr (2012a). Greedy methods are analyzed and compared experimentally with existing L_1 regularized approaches. The greedy methods analyzed are our own OMP-BRM and the OMP-TD method previously given by Johns (2010), while L_1 regularized approaches are represented by LARS-BRM (Loth et al., 2007) and LARS-TD (Kolter and Ng, 2009). In addition to an analysis of some theoretical properties of OMP-BRM and OMP-TD, the main contribution of this chapter is in demonstrating the preferability of the greedy methods, and OMP-TD in particular, over their L_1 regularized counterparts.

We close with discussion and concluding remarks in chapter 7.

2

Background

In this chapter, I review the models and methods that will be the foundation for subsequent sections, and develop the notation that will be used throughout.

2.1 Markov Decision Process

Markov decision processes (MDPs) model controlled, discrete-time stochastic processes in which the performance of a controlling agent is measured by the accrual of (action and state dependent) rewards. Formally, an MDP is a tuple (S, A, P, R, γ), with elements

S	state space
A	action space
P	transition function
R	reward function
γ	discount factor

At any time step t, the system is in some state $s_t \in S$, the agent chooses some action $a_t \in A$, and observes a transition to some state $s_{t+1} \in S$ with probability $P(s_t, s_{t+1}, a_t)$. In addition, the agent receives a real-valued reward determined by

the starting state: $r_t = R(s_t)$. State transitions are stochastic, and obey the *Markov property*; given the starting state and action, the new state is conditionally independent of previous history. We also assume time independence (stationarity). In the context of a single transition, we drop the t subscripts, and notate the starting state as s, the end state as s', and the reward and action as r and a.

A *trajectory* is a sequence $(s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_{n-1}, a_{n-1}, r_{n-1}, s_n)$ summarizing a history of n successive state transitions, rewards, and actions. The value of the trajectory is the γ-discounted sum of rewards

$$\sum_{t=0}^{n} \gamma^t r_t = \sum_{t=0}^{n} \gamma^t R(s_t)$$

where $\gamma \in [0, 1]$. The number of time steps over which an agent's performance is measured (i.e., the maximum length n of any trajectory) is the *horizon* of the problem. Here we only consider infinite horizon problems with γ strictly less than 1, which guarantees that the value of any trajectory is well-defined.

A *policy* is a mapping $\pi : S \mapsto A$ which specifies the action an agent will take in each state. The *value function* at state s for policy π is the expected trajectory value when following π from s. The value function V^π of policy π is the fixed point of the Bellman equation:

$$V^\pi(s) = R(s) + \gamma \sum_{s' \in S} P(s, s', \pi(s)) V^\pi(s'), \forall s \in S.$$

It is useful to consider the case in which the state space S and action space A are finite and enumerable. In this case, we can rewrite the Bellman equation compactly in matrix-vector form:

$$\begin{aligned} V^\pi &= R + \gamma P^\pi V^\pi \\ &= T^\pi V^\pi \end{aligned} \tag{2.1}$$

where $V^\pi[s] = V^\pi(s)$ and P^π is a transition matrix with entries $p^\pi_{ij} = Pr(s' = j|s = i, a = \pi(s)) = P(i, j, \pi(i))$. T^π is the *Bellman operator* for policy π, and V^π is the fixed point of this operator. When the policy context is clear, it is common to drop the superscripts, and simply write V, P, and T.

There are numerous ways to compute V^π, a task known as *policy evaluation*. For MDPs with small state-spaces, the system of equations above (equation 2.1) can be solved directly, giving $V^\pi = (I - \gamma P^\pi)^{-1} R$. For larger state-spaces, the fixed point can be found using *value iteration*, an indirect method effectively equivalent to Richardson iteration (Richardson, 1910); at each step we update a value function $V^{t+1} = T^\pi V^t$, starting with an arbitrary initial value function V^0 (a typical choice is $V^0 = 0$). As is well known, T is a contraction in max-norm (the spectral radius of γP^π is less than 1), ensuring that value iteration converges to V^π.

The goal in planning and reinforcement learning is to find a policy π^*, which maximizes the value at every state. The value function of all optimal policies for a given MDP is the optimal value function $V^* = V^{\pi^*}$. The optimal value function and optimal policies are connected by the greedy form of the Bellman equation:

$$V^*(s) = \max_a \left[R(s) + \gamma \sum_{s' \in S} P(s, s', a) V^*(s') \right], \tag{2.2}$$

$$\pi^*(s) = \arg\max_a \left[R(s) + \gamma \sum_{s' \in S} P(s, s', a) V^*(s') \right]. \tag{2.3}$$

We can also define the greedy Bellman operator, T^*:

$$T^* V = \max_\pi \left[R + \gamma P^\pi V \right].$$

Some algorithms make use of a (state, action) value function known as the *Q-function*. The Q-value for policy π at a (state, action) pair (s, a) is defined as the

value of taking action a in state s, following π for all subsequent steps:

$$Q^{\pi}(s,a) = R(s) + \gamma \sum_{s' \in S} P(s,s',a) V^{\pi}(s')$$

The optimal Q-function Q^* yields and can be derived from the optimal value function and policy:

$$\begin{aligned} Q^*(s,a) &= R(s) + \gamma \sum_{s' \in S} P(s,s',a) V^*(s') \\ &= R(s) + \gamma \sum_{s' \in S} P(s,s',a) \max_{a'} Q^*(s',a'); \\ \pi^*(s) &= \arg\max_a Q^*(s,a). \end{aligned}$$

Numerous algorithms exist to solve for the optimal policy of an MDP. Two algorithms that are foundational for the approximate algorithms to be discussed in later sections are *value iteration* and *policy iteration*.

Value iteration shares the name and the same basic procedure as value iteration for policy evaluation. In this instance, value iteration simply solves for the fixed point of equation 2.2 by iterative application of the greedy Bellman operator, T^*. Like the non-greedy Bellman operator, T^* is a contraction in max-norm, ensuring that value iteration converges to V^*. The optimal policy is then simply recovered as the "greedy" policy with respect to V^* (equation 2.3).

Policy iteration finds successive policies $\{\pi_0, \pi_1, \ldots, \pi^*\}$ by interleaving policy evaluation and *policy improvement*, in which the policy is updated to be the greedy policy with respect to the current value function. Starting from some initial policy, we 1) compute the value function (e.g., by solving the system of equations (2.1) or using value iteration), and 2) determine our next policy by applying equation (2.3). These two steps are then repeated until the policies converge.

From equation (2.2), it is easy to see that an improvement in the value at any state cannot reduce the value at any other state. Thus the value of each new policy in policy iteration is at least as good as the previous policy at every state, and is better at one or more states unless the previous policy is optimal. Since for finite, enumerable state spaces there are a finite number of possible policies, policy iteration is guaranteed to converge to an optimal policy.

2.2 Reinforcement Learning

Given a model, planning is simply the process of finding an optimal policy, i.e., one which maximizes the discounted expected value at every state. *Reinforcement learning* (RL) can be viewed as planning in an environment that conforms to (or at least can be approximated by) an unknown Markov decision process model. The learner interacts with the environment, obtaining rewards and observing state transitions, with the goal of learning an optimal or near-optimal policy for acting in the environment. Planning algorithms almost universally rely on being able to predict future value in order to select actions which optimize that value, and RL algorithms are no different. The key differentiator between most RL algorithms is the way in which value functions are computed. How data is collected, and how and when policies are evaluated and updated are complementary choices which combine to yield many variations in the basic algorithms.

RL problem domains can exist as mathematical models, simulators, or physical systems. Models are useful for validating RL algorithms, particularly when they permit exact solution. Models and simulators can be constructed for physical systems which would otherwise be impractical to use in RL; it is, for example, very expensive to crash real helicopters in order to train an agent how to fly them. Simulators also have the benefit of providing virtually unlimited training data, which can further be collected from arbitrary parts of the state space. Physical systems have the

benefit of realism; even in controlled experimental situations, real physical systems often exhibit behaviors that are difficult to capture in simulation (for example, wheel slippage on varying surfaces).

Much of the focus in this paper will be on *offline* RL. In offline learning, data collected in one or more training episodes is used to learn a value function or policy. One advantage of offline learning is that the collected data is all available for use in any manner desired; it can be used in either iterative or batch operations, it can be organized as needed for subtask learning or cross validation, and it can be revisited as often as needed. Data collection for offline learning is generally focused on *exploration*; the agent is given policies or put in initial states with the goal of obtaining as much useful information as possible.

In contrast, some problem domains imply a need to learn at the same time data is collected. For instance, an autonomous robot searching for survivors in a collapsed building will not have the luxury of performing "practice runs". In these cases, the agent needs to learn *online*; each interaction with the environment is used to update the current value function or policy, without waiting for a separate learning step as in offline learning. Data collection in online learning typically must balance exploration with *exploitation*; the agent needs to use what it has learned so far for maximal gain, but must also continue to gather data to ensure that it is not missing better alternatives.

RL algorithms generally follow the basic solution techniques used in planning: value iteration and policy iteration. Recall that value iteration has the learner iteratively update a value function estimate while always choosing greedy actions with respect to its current estimate. In policy iteration, the agent alternately learns the value of a fixed policy, and improves the policy greedily with respect to the learned value. Broadly speaking, RL algorithms can all be used in some sense to evaluate fixed policies, and therefore can also be used in a policy iteration setting.

In online settings, however, a strict policy iteration approach will not work; instead, in most online algorithms the learner does not wait until a current policy is fully evaluated before improving its policy. A typical implementation uses an ϵ-greedy policy; with probability $1-\epsilon$, the agent chooses the greedy action for its current state, and with probability ϵ it chooses a random action. This is a common workaround for the exploration/exploitation dilemma described previously; the smaller we make ϵ, the more the policy favors exploitation over exploration. The ϵ parameter can be fixed, but usually decreases on a predetermined schedule to facilitate convergence.

There is a strong interaction between problem domains, the above choices we can make, and the algorithms we can apply. Some problem domains dictate certain choices; some choices may dictate certain algorithms; and some choices and algorithms may perform better or worse on a particular problem. In this section I summarize some of the algorithms which give a foundation to topics in later chapters; more in-depth discussion of these algorithms and related topics can be found in standard texts on reinforcement learning and dynamic programming (Sutton and Barto, 1998; Bertsekas and Tsitsiklis, 1996).

2.2.1 Monte Carlo Estimation

The most straightforward algorithm for estimating $V(s)$ for some state s under a policy is to run the policy starting from state s, summing (discounted) rewards, until a terminal or absorbing state is reached. Averaging over many such trajectories gives a sample mean estimate of the value function, which converges on the true value as the number of trajectories increases to infinity. With care, one can also use the subtrajectories for each state through which the trajectory passes as estimates for the values of those states. With sufficient trajectories starting at all states, an accurate value function is obtained. The obvious disadvantage of this approach (called Monte Carlo estimation) is that it potentially requires collecting a great deal of data before

accurate estimates can be obtained. All trajectories need to be run to completion, or the method must be modified in some fashion to accommodate truncated trajectories.

2.2.2 Fitted Value Iteration

Value iteration (dynamic programming) is an attractive algorithm for planning with MDPs. For small, discrete MDPs, it is possible to sample n rewards and transitions from every state, and update a value function by applying the Bellman operator iteratively, replacing the expectation with respect to the transition model with the sample mean. That is,

$$\hat{V}_{t+1}(s) = \frac{1}{n}\sum_{i=1}^{n} r_i + \gamma \hat{V}_t(s'_i). \tag{2.4}$$

This procedure always converges to some final $\hat{V}$, with $\lim_{n\to\infty} \hat{V} = V^*$.

In a large or continuous state space domain, the situation is somewhat more complicated. Now we must sample starting states as well as transitions from those states. *Fitted*, or approximate, value iteration (FVI) computes the values of the sample states using the RHS of equation 2.4 and uses the updated values as training data for constructing a new approximate value function. This procedure is repeated, with the hope of converging to an accurate approximate value function. FVI can be used for control in a variant known as fitted Q-iteration.

Unfortunately, FVI can perform poorly with some approximation architectures, including linear approximators (Boyan and Moore, 1994). Gordon (1995) provides sufficient (but not necessary) conditions which guarantee convergence, and gives a number of example approximation architectures which meet the conditions. More recently, work by Ernst et al. (2005) has shown promising results for fitted Q-iteration with certain non-linear approximators.

2.2.3 Temporal Difference Learning

Perhaps the most influential algorithm in RL is *temporal difference learning* (TD) (Sutton, 1988). The basic TD algorithm applies a stochastic iterative approximation to find the fixed point $V^\pi = T^\pi V^\pi$ for a given policy π. TD starts with an initial value function V_0 and then samples on-policy interactions with the environment. For each sampled transition (s, r, s'), TD updates V using

$$V'(s) = V(s) + \alpha(r + \gamma V(s') - V(s)), \tag{2.5}$$

The above equation can be seen as the update of a Robbins-Monro (Robbins and Monro, 1951) stochastic approximation procedure solving the Bellman equation (2.1). The rate parameter α is decreased over time. With sufficient samples and a proper rate decrease schedule, the approximation converges to V^π. The quantity in the parentheses, $r + \gamma V(s') - V(s)$, is the *temporal difference*, which can be viewed as a sample of the Bellman error.

Another view of TD is that it attempts to bring the benefits of dynamic programming to Monte Carlo policy evaluation. By itself, dynamic programming requires a model, which is generally not available in an RL setting; on the other hand, Monte Carlo evaluation requires (potentially many) complete trajectories with which to build up estimates of the value of each state. TD updates V with each sample transition by using the current estimate of V to stand in for all future transitions in the current trajectory. While TD is susceptible to the usual problems that arise in any gradient procedure (poor or slow convergence, difficulty in choosing the rate parameter), TD allows for significantly greater flexibility and data efficiency than the more stable Monte Carlo approach.

TD(λ)

The connection with Monte Carlo estimation is drawn more explicitly in a generalization of TD known as *TD(λ)*. In TD(λ), the TD update is applied not only to the most recently visited state, but also to previously visited states. At a high level, what TD(λ) does is to replace the estimated next state value ($V(s')$ in equation 2.5) with a weighted average of the estimated next state value and the TD(λ) update of the next state. The actual mechanism used by TD(λ) keeps track of a quantity known as the *eligibility* for each state, which gives the weight with which the current temporal difference is applied to the state, and which implies in some sense the "recentness" of visitation of the state. The use of eligibilities lets TD(λ) perform immediate online updates, without having to store and remember trajectories for eventual update.

The parameter $\lambda \in [0, 1]$ controls the apportioning of weight between dynamic programming updates (trusting the current value approximation) and Monte Carlo evaluation (using the actual observed trajectory rewards). At the extremes, TD(0) is equivalent to the basic TD algorithm above, while TD(1) is equivalent to Monte Carlo estimation. In general, when we refer to TD, we mean TD(0), although many applications generalize to TD(λ).

Q-learning and Sarsa

TD can be used for simple policy evaluation in a policy iteration loop, but in an online setting a different approach may be preferred. Two closely related approaches for applying TD in an online control setting are *Q-learning* (Watkins, 1989) and *Sarsa* (Rummery and Niranjan, 1994). Both of these approaches apply temporal difference updates to Q-function learning.

Q-learning is an *off-policy* algorithm; sample transitions (s, a, r, s') are collected following an exploration policy for which the only requirement is that every action is taken from every state infinitely many times (over an infinite number of steps).

For each sampled transition (s, a, r, s'), Q-learning applies a greedy TD update to the Q-function:

$$Q'(s,a) = Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a)).$$

Under typical assumptions on the stepsize (e.g., Assumption 4 in chapter 4), Q-learning will converge on Q^*.

Sarsa uses TD to learn the Q-function with respect to an ϵ-greedy policy. In Sarsa, sample transitions include the next action to be taken, that is, samples are tuples (s, a, r, s', a'). The initial action a is chosen based on the initial state from the ϵ-greedy policy based on the current Q-function; subsequently a is set to be the a' from the previous step. The next action a' is chosen based on the next step from the ϵ-greedy policy. The Sarsa update is

$$Q'(s,a) = Q(s,a) + \alpha(r + \gamma Q(s',a') - Q(s,a)).$$

Under typical assumptions on the stepsize, if every state-action pair is visited infinitely often and the policy converges to the greedy policy, then Sarsa will also converge on Q^* (Gordon, 2001).

Approximate TD

For many problems with large or infinite state spaces, it is impossible to exactly represent either value functions or policies. In this case it is customary to use some form of approximation. If we replace the true value function V with some approximation $\hat{V}_\theta$ with parameter vector θ, then the stochastic gradient procedure solving for the fixed point $\hat{V}_\theta = T\hat{V}_\theta$ looks like

$$\theta' = \theta + \alpha(r + \gamma \hat{V}(s') - \hat{V}(s)) \nabla_\theta \hat{V}(s).$$

In this dissertation we are concerned primarily with the approximation of value functions using linear approximation. Under linear approximation, $\hat{V}(s) = \varphi(s)^T w$,

with feature vector φ and parameter ("weight") vector w. In this setting $\nabla_w \hat{V} = \varphi$, so the above becomes

$$w' \quad = \quad w + \alpha(r + \gamma\varphi(s')^T w - \varphi(s)^T w)\varphi(s). \tag{2.6}$$

Linear approximate TD (linear TD) for a fixed policy converges under certain assumptions, to a fixed point with bounded error (Van Roy, 1998; Bertsekas and Tsitsiklis, 1996); more attention will be paid to linear TD and its fixed point in section 2.4.

2.2.4 LSTD

Least-squares temporal difference learning (LSTD), due to Bradtke and Barto (1996), replaces the gradient approach of linear TD with an algorithm derived from least-squares regression. In the Bradtke and Barto derivation, a sample trajectory is seen as a process yielding training data for the approximation problem

$$\begin{aligned} \Phi w &= R + \gamma P \Phi w \\ (\Phi - \gamma P \Phi) w &= R \\ A w &= b, \end{aligned}$$

where we make the substitutions $A = \Phi - \gamma P \Phi$ and $b = R$. Here Φ is a design matrix formed using a set of features $\{\varphi_i\}$ evaluated at a set of sampled states.

The ordinary least squares solution (OLS) to this system (see section 2.3 for more) is $w = (A^T A)^{-1} A^T b$. However, in the usual RL setting, the matrix P is not available, and so the matrix A is approximated using sampled state, next state transitions: $\hat{A} = (\Phi - \gamma \Phi')$, where Φ' is the design matrix of features evaluated at next state samples. The introduction of sampling noise into the predictor $\hat{A}$ leads to bias into the OLS solution; to fix this, Bradtke and Barto apply the method of

instrumental variables, using Φ as the instrument:

$$\begin{aligned} w &= (\Phi^T \hat{A})^{-1} \Phi^T b) \\ &= (\Phi^T (\Phi - \gamma \Phi'))^{-1} \Phi^T R. \end{aligned} \tag{2.7}$$

An alternate derivation of LSTD is given by Boyan (2002), who considers the fixed point to which linear TD converges (see section 2.4); this fixed point is, in expectation, equivalent to the LSTD fixed point. Boyan further generalizes LSTD to LSTD(λ) using his derivation.

The advantage that LSTD holds over linear TD for policy evaluation is that it makes efficient use of all samples seen at any point in time, essentially finding the "best" solution given the information received. In contrast, TD is able to make only incremental updates based on the most recent information, leading to slower convergence. LSTD can be used either online or offline, as the matrices used in its solution can be constructed incrementally; however, the cost of computing the fixed point, which requires inverting the $k \times k$ matrix A (k is the number of features), is significantly larger than the cost of the gradient update step in linear TD, so it is generally not desirable to solve for the fixed point at every step.

Least-squares policy iteration (LSPI)(Lagoudakis and Parr, 2003) gives a practical off-policy method for applying LSTD to control. At each iteration, LSPI uses a set of sample transitions to approximate Q-values over all actions with respect to the current policy. The current policy, in turn, is defined implicitly as the greedy policy with respect to the most recent Q-function approximation.

2.3 Linear Approximation

Linear approximation is a fundamental technique for approximating an unknown function given (possibly noisy) sample outputs from the function. The linear approximation is a weighted combination of *features*, which are any numerical functions on

the inputs. In some settings, the purpose is to find causal or predictive relationships using the elements of a vector-valued input such as medical diagnostic or meteorological data; it is usually assumed that the features, or some subset of the features, have a strong correlation with the function of interest. Linear approximation may also serve the purpose of transforming a set of data into a representation that facilitates a specific application, for example denoising or compressing signals. In a broader learning context, individual input data may be scalar or low-dimensional, and the unknown function may have no obvious structure with respect to the input space; features are chosen to be linearly independent functions of the inputs, defining a (possibly high dimensional) vector space in which it is hoped the unknown function can be closely approximated. In this context, the terms *feature* and *basis function* are used interchangeably.

As used in this document, the linear *regression* problem is that of finding an approximation to some unknown function $f : \mathbb{R}^d \mapsto \mathbb{R}$ given samples $(x_i, y_i) : y_i = f(x_i) + \omega$, where ω represents noise. Defining some set of k linearly independent features $\{\varphi_i : i = 1..k\}$ on $\mathbb{R}^d$, the linear approximation is

$$f(x) \approx \hat{f}(x) = \sum_{i=1}^{k} \varphi_i(x) w_i,$$

where the learned parameters w_i are known as the *approximation weights*. Constructing matrix $\Phi : \Phi_{ij} = \varphi_j(x_i)$, known as the *design matrix*, the regression problem is to find vector w such that $\Phi w \approx y$, where y is the vector of sample targets.

A starting point for most regression approaches is to form an objective function that includes the sum of squared errors. The *ordinary least squares* (OLS) approach chooses w to minimize

$$\sum_i \left[\sum_{j=1}^{k} \varphi_j(x_i) w_j - y_i \right]^2 = \|\Phi w - y\|^2.$$

When the columns of Φ are linearly independent, the OLS solution is unique and well-defined, and is given by

$$w = (\Phi^T \Phi)^{-1} \Phi^T y.$$

2.3.1 Regularization

Ordinary least squares regression minimizes the squared error between a linear approximation of a function and a training set of outputs sampled from the function. In general, as more features are available to the approximator, more of the error can be eliminated. In the extreme case, if there are as many (linearly independent) features as samples, OLS reduces to solving a system of equations, and error is zero. However, samples typically include some noise, in which case OLS can "overfit", eliminating error with respect to the training set by fitting the noise. The resulting approximation will generally have a higher error rate on test data (data drawn independent of the training data) than an approximation produced with fewer features.

Various forms of *regularization* have been proposed to smooth, simplify, or otherwise modify least-squares approximations to reduce overfitting. Regularization typically takes the form of a penalty, depending on the approximation weights, added to the least-squares objective function:

$$\min_w \|\Phi w - y\| + \lambda h(w). \tag{2.8}$$

Here the function h is a generic penalty term, and λ is a regularization parameter which adjusts the trade-off between overfitting and underfitting. Of particular interest are penalty terms which involve a norm of the weight vector: $h(w) = \frac{1}{p}\|w\|_p^p$.

With $p = 2$, the above regression method is known as *Tikhonov* or *ridge* regression, or simply as L_2-regularized regression. The weight vector for an L_2-regularized problem can be found in closed form,

$$w = (\Phi^T \Phi + \lambda I)^{-1} \Phi^T y,$$

making this form of regression the most commonly used.

When $p = 1$, the form of regularization is called L_1 regularization. A key benefit of L_1 regularization, in addition to reducing overfitting, is that it tends to promote sparse solutions, that is, solutions in which some number of approximation weights are zero. This reduces the computational complexity when using the linear model for prediction, since the zero-weighted features can be eliminated from the model, and also can be useful in interpretation of the model. In particular, we can consider the use of L_1 regularization as a technique for *automatic feature selection*, that is, choosing which features are relevant to the process generating our training data.

Tibshirani introduced L_1 regularized regression in a form he named *the Lasso* (Tibshirani, 1996). The Lasso is a least-squares error minimization problem with a constraint on the L_1 norm of the weight vector:

$$\text{minimize } \|\Phi w - y\|^2 \text{ subject to } \|w\|_1 \leqslant \lambda$$

Columns of Φ are assumed to be 0-mean and normalized; the target vector y is assumed to be 0-mean. The parameter λ is used to manipulate the trade-off between regularization (and sparsity), and training error minimization. Above a certain threshold, the constraint is no longer tight, and the minimizing value of w is simply the OLS solution.

The Lasso problem can be solved directly using linear programming, but this approach is not very efficient for large problems. This motivates the LARS algorithm (Efron et al., 2004), which can be thought of as a homotopy method for solving the Lasso, as well as various gradient descent (Hale et al., 2007; Duchi and Singer, 2009; Langford et al., 2008) and pursuit (Chen et al., 2001) algorithms. Conveniently, these approaches are more readily adapted to reinforcement learning than the direct, linear programming implementation of the Lasso.

LARS and other methods formulate L_1 regularization using unconstrained mini-

mization:

$$\min_w \frac{1}{2}\|\Phi w - y\|^2 + \beta\|w\|_1. \tag{2.9}$$

It is a straightforward application of Lagrange multiplier theory to show that the solution of an instance of the above matches a solution to a Lasso instance with the same objective term, and that furthermore there is a continuous mapping between λ and β preserving equivalence. We can construct the Lagrange function for the constrained minimization problem (equation 2.3.1) as

$$L(w, \beta) = \frac{1}{2}\|\Phi w - y\|^2 + \beta(\|w\|_1 - \lambda).$$

The subdifferential of L is:

$$\nabla_w L = \Phi^T(\Phi w - y) + \beta\,\mathrm{SGN}(w)$$

where $\mathrm{SGN}(w)$ is a set-valued function defined componentwise as

$$\mathrm{SGN}(w)_i = \begin{cases} \{+1\}, & w_i > 0 \\ [-1, 1], & w_i = 0 \\ \{-1\}, & w_i < 0 \end{cases}$$

Note that the subdifferential of L is also the subdifferential of the objective function in equation 2.9; with respect to w, these equations have the same optima. Setting the subdifferential to zero (since the subdifferential is set-valued, the actual requirement is that zero is in the subdifferential) yields the (well-known) optimality conditions

$$[\Phi^T(\Phi w - y)]_i \begin{cases} = -\beta, & w_i > 0 \\ \in [-\beta, \beta], & w_i = 0 \\ = +\beta, & w_i < 0 \end{cases} \tag{2.10}$$

We can determine the relationship between λ and β by imposing the additional condition $\beta(\|w\|_1 - \lambda) = 0$. For a given input target y, this condition together with the optimality conditions defines a system of inequalities which can be used to determine the mapping between λ and β.

2.4 Linear Approximation in Reinforcement Learning

We now consider in more depth the combination of linear approximation and reinforcement learning. We are interested in approximating the value function as a linear combination of features of the state. That is, given sampled state transitions and rewards and some set of k linearly independent features $\{\varphi_i : i = 1..k\}$ defined on the state, we want to find $\hat{V} = \Phi w \approx V$.

Technically this only gives us approximation of V; for policy improvement, we typically want to approximate Q-functions. However, algorithms for approximating V can be modified for approximating Q-functions by simply defining features over state-action pairs instead of over states only. This the approach used, for instance, by LSPI (Lagoudakis and Parr, 2003).

Our approach to finding w should be guided by the Bellman fixed point equation. Given some current value approximation $\hat{V}$, the Bellman error (BE) or residual is simply defined as the difference between the RHS and LHS of the Bellman equation; we write

$$BE = T\hat{V} - \hat{V}.$$

In reinforcement learning, we do not have access to the exact operator T, but instead have samples of the reward and next states, thus we have a sampled or approximate Bellman residual:

$$\hat{T}\hat{V} - \hat{V} = R + \gamma\Phi' w - \Phi w, \tag{2.11}$$

where R is the vector of sample rewards, Φ is the design matrix of features evaluated at sampled states, and Φ' is the design matrix of features evaluated at sampled next states.

There are two broad classes of algorithms for finding the vector w, both of which use the Bellman error as a guide. Perhaps the most obvious approach is to find w which minimizes the Bellman error. This approach, known as *Bellman residual min-*

imization (BRM), can be approached much like regression, and thus easily inherits much of the rich regression toolbox. The other approach, which we will called the *linear fixed point* (LFP) approach, also uses the Bellman error, but in a fixed point formulation that is subtly different than BRM. Both of these approaches are explored in more detail below.

2.4.1 Bellman Residual Minimization

The Bellman residual minimization approach, espoused by Baird (1995), finds w minimizing

$$\|\hat{T}\hat{V} - \hat{V}\|.$$

If we expand this and group terms depending on w, we quickly see that we have a least-squares regression problem:

$$\begin{aligned} w &= \arg\min_w \|R + \gamma\Phi' w - \Phi w\| \\ &= \arg\min_w \|R - (\Phi - \gamma\Phi')w\| \\ &= \arg\min_w \|b - Aw\|, \end{aligned}$$

where $b = R$ and $A = \Phi - \gamma\Phi'$. The OLS solution to this regression problem is simply $w = (A^T A)^{-1} A^T b$.

However, recall from the discussion of LSTD (section 2.2.4) that the introduction of sampling noise into the predictor A leads to bias into the OLS solution. This occurs because of the product term $\Phi'^T\Phi'$, which is not an unbiased sample of the product $(P\Phi)^T(P\Phi)$ (Sutton and Barto, 1998). Two next state samples for each transition are required to eliminate the bias, which is generally possible only when we have available a model or simulator of the Markov process. Assuming we can do this, we generate *two* next state design matrices, Φ'^A and Φ'^B, and solve for w as

$$w = ((\Phi - \gamma\Phi'^A)^T(\Phi - \gamma\Phi'^B))^{-1}(\Phi - \gamma\Phi'^A)^T R.$$

Treated as a simple regression problem, it is straightforward to add regularization to this problem; in fact, any regression algorithm can be applied. In general, however, the results from doing this will be unsatisfactory unless a way is found to incorporate double sampling, which is not always trivial to do. An example of this is in the algorithm LARS-BRM[1] (Loth et al., 2007), examined in section 6.1.2.

2.4.2 Linear Fixed Point

The linear fixed point solution finds w such that

$$w = \arg\min_u \|\hat{T}\Phi w - \Phi u\|^2. \tag{2.12}$$

In words, w is the vector which gives the least-squares regression fit to the (approximate) Bellman operator applied to Φw. Equivalently we can write

$$\Phi w = \Pi \hat{T} \Phi w,$$

where Π is the least-squares projection operator onto the vector space spanned by Φ. Using ordinary least-squares projection, $\Pi = \Phi(\Phi^T\Phi)^{-1}\Phi^T$, so we have

$$\begin{aligned}
\Phi w &= \Phi(\Phi^T\Phi)^{-1}\Phi^T(R + \gamma\Phi' w) \\
(\Phi^T\Phi)w &= \Phi^T R + \gamma\Phi^T\Phi' w \\
(\Phi^T\Phi - \gamma\Phi^T\Phi')w &= \Phi^T R \\
w &= (\Phi^T\Phi - \gamma\Phi^T\Phi')^{-1}\Phi^T R.
\end{aligned}$$

If we compare this to equation (2.7), we see that the linear fixed point is, indeed, the solution found by LSTD.

The linear fixed point is also the fixed point of linear TD. Linear TD can be viewed, somewhat loosely, as a gradient descent method minimizing the "objective" function $\frac{1}{2}\|\Phi u - T\Phi w\|^2_\sigma$. Here σ is the *stationary distribution* of the Markov process,

[1] Our name for the algorithm by Loth et al.

loosely defined as the probability distribution σ such that $P\sigma = \sigma$. If we define $\Sigma = diag(\sigma)$ (the square matrix having σ on the diagonal and zeros elswhere), then the σ-weighted norm $\|\cdot\|_\sigma$ is defined by $\|x\|_\sigma = x^T\Sigma x$.

The linear TD update is derived from the standard gradient learning rule by taking the gradient of the objective function with respect to u, together with the constraint $u = w$:

$$\begin{aligned} w_{t+1} &:= w_t - \alpha\nabla_u\left(\frac{1}{2}\|\Phi u - T\Phi w_t\|_\sigma^2\right) \\ &:= w_t + \alpha\nabla_u(\Phi u - T\Phi w_t)^T\Sigma(T\Phi w_t - \Phi u) \\ &:= w_t + \alpha\ \Phi^T\Sigma(T\Phi w_t - \Phi w_t) \end{aligned}$$

In online learning, the values of Φw and $T\Phi w$ are replaced by samples taken from the stationary distribution, yielding a stochastic estimate of the gradient (compare to equation (2.6)). It is straightforward to show that, at the fixed point, we have:

$$\begin{aligned} E[\alpha\Phi^T(\hat{T}\Phi w - \Phi w)] &= 0 \\ E[\Phi^T R + \gamma\Phi^T\Phi' w - \Phi^T\Phi w] &= 0 \\ E[\Phi^T R] &= E[(\Phi^T\Phi - \gamma\Phi^T\Phi')w] \\ w &= E[(\Phi^T\Phi - \gamma\Phi^T\Phi')^{-1}\Phi^T R]. \end{aligned}$$

A more complete analysis of the fixed point of linear TD can be found in Bertsekas and Tsitsiklis (1996) and Van Roy (1998)

The question of adding regularization to the linear fixed point is one which has been explored in depth in recent years (Farahmand et al., 2008; Taylor and Parr, 2009; Kolter and Ng, 2009; Hoffman et al., 2011). Of particular interest in the context of this dissertation is the work of Kolter and Ng (2009), who first introduced the L_1-regularized linear fixed point. The L_1-regularized linear fixed point will be discussed and expanded on in chapters 4 and 5.

3

Related Work

The problem of selecting features for approximate reinforcement learning is a long-standing one; the choice of features is crucial (Boyan and Moore, 1994; Gordon, 1995), leading many researchers to focus on this problem. One approach has been to generate features from scratch (Menache et al., 2005; Mahadevan and Maggioni, 2006; Keller et al., 2006; Mahadevan and Maggioni, 2007; Parr et al., 2007; Geramifard et al., 2011). Of particular interest is the work by Parr et al. (2007), in which successive Bellman residuals (or approximations to them) are used as features. Later work (Parr et al., 2008) suggests that this approach can be highly successful. The Parr et al. work directly inspires the greedy methods discussed in chapter 6.

Other work has tackled the feature selection problem through kernelization (Engel et al., 2003; Rasmussen and Kuss, 2003; Xu et al., 2005). Taylor and Parr (2009) demonstrated the importance of regularization in this setting and provided a unified view of kernelized RL. The use of L_2 regularization to avoid overfitting with very expressive feature spaces was also examined by Farahmand et al. (2008).

L_1 regularization is particularly useful in inducing sparsity in approximation solutions. Tibshirani (1996) first applied L_1 regularization for feature selection in

supervised learning. Tibshirani defined "the Lasso" as a quadratic program minimizing the sum of squares subject to a constraint on the sum of the absolute values of the regression coefficients. Subsequent works introduced efficient homotopy (Efron et al., 2004; Turlach et al., 2005) and interior point (Kim et al., 2007) methods for solving the Lasso. Iterative gradient descent methods for applying L_1 regularization (Daubechies et al., 2004; Hale et al., 2008; Langford et al., 2008; Carbonetto et al., 2008; Duchi and Singer, 2009) also followed. The work by Duchi and Singer also establishes convergence of the online variant, in which the gradients are replaced with stochastic estimates.

The use of L_1 regularization in reinforcement learning first appears with Loth et al. (2007), who take the simplest approach of applying the Lasso to a Bellman residual minimizing formulation. The Loth et al. work implements a homotopy approach effectively identical to that of LARS (Efron et al., 2004). Kolter and Ng (2009) first introduced the notion of an L_1 regularized linear fixed point, again following the LARS template. Our work on L_1 regularized TD learning (chapters 4 and 5) directly follows the work by Kolter and Ng. Other work in this space includes a linear complementarity problem formulation of the L_1 regularized linear fixed point (Johns et al., 2010) and a projected fixed point formulation (Ghavamzadeh et al., 2011), both of which guarantee the existence and uniqueness of the fixed point under certain conditions.

Mahadevan and Liu (2012) give an algorithm for regularized reinforcement learning, based on mirror descent (Nemirovski and Yudin, 1983), which is comparable to the gradient descent approach described in chapter 4. The Mahadevan and Liu work generalizes to norms other than L_1 and thus subsumes our work, but the derivation is correspondingly more complex.

Most work regarding L_1 regularized reinforcement learning has focused on policy evaluation, decoupled from policy improvement. Chapter 5 presents an algorithm

which solves for the greedy L_1 regularized linear fixed point; the solutions of this algorithm are approximated by the LC-MPI algorithm of Johns et al. (2010). A different approach to L_1 regularized RL is given by Petrik et al. (2010), in an extension of approximate linear programming (ALP) (de Farias and Van Roy, 2003). The Petrik et al. work is notable in that, unlike unregularized ALP, their algorithm guarantees bounded solutions. Like ALP, regularized ALP suffers from performance degradation due to sampling of transition probabilities, which is not an issue for algorithms based on the linear fixed point.

An alternative approach for sparse approximation that has received renewed attention in the supervised learning community is that of using a simple algorithm that greedily adds new features (Tropp, 2004; Zhang, 2009), such as matching pursuit (Mallat and Zhang, 1993) and orthogonal matching pursuit (Pati et al., 1993). Such algorithms have many of the good properties of L_1 regularization methods, while also being extremely efficient and, in some cases, allowing theoretical guarantees on recovery of the true form of a sparse target function from sampled data. Johns (2010) explores orthogonal matching pursuit in the context of automatic feature generation. Our work in chapter 6 builds on the work by Johns, adding a new algorithm, providing new theoretical analysis and a thorough experimental comparison with L_1 regularized methods.

4

L_1 Regularized Linear Temporal Difference Learning

In this chapter, we explore the combination of L_1-regularization and the linear fixed point in the online learning setting. Kolter and Ng (2009) introduced the notion of an L_1 *regularized linear fixed point*, and the LARS-TD algorithm for finding it; their experiments demonstrated the effectiveness of this technique at selecting good features in the presence of noise features that could otherwise cause overfitting. Johns et al. (2010) subsequently produced the algorithm LC-TD, which also finds the L_1 regularized linear fixed point, and used it in a modified policy iteration setting to find (approximate) optimal policies.

LARS-TD and LC-TD are inherently batch learning methods; regularization in online reinforcement learning is a topic heretofore unexplored. One reason for this is that regularization, particularly L_1 regularization, is most useful in the low data regime, where noise has the most impact and overfitting is most likely. A typical assumption in online learning is that data are infinite, or at least plentiful. This may not always be the case, of course, and furthermore it is occasionally desirable

to apply on "online" algorithm to batch data, in which case overfitting may indeed be a concern.

An alternative motivation for using L_1 regularization in online learning is in reducing the memory and computational requirements of the online updates. Langford et al. (2008) describe in some detail an algorithm which utilizes a class of shrinkage operators (including the L_1 regularizing soft-threshold shrinkage operator) in online supervised learning. In this work, sparsity of the iterates and sparsity in the features are leveraged to speed up the algorithm when working with large data sets. Sparsity in the iterates allows the algorithm to keep in memory only those features which are currently non-zero weighted, preventing additional costs due to swapping. Sparsity in the features directly aids computational efficiency, as only weights of non-zero features need be updated. The Langford et al. approach translates directly into the RL setting, given a problem with a large set of sparse features such that the non-zero features can be determined efficiently from the state alone, without iterating over all features. Certain types of sparse coding such as CMACs fall naturally into the category of sparse features for which the non-zero features are determinable from the state alone.

This chapter presents an L_1 regularized, online stochastic approximation algorithm, based on an iterative update equation sharing the fixed point of LARS-TD and LC-TD. We validate the ability of the online algorithm to converge to the fixed point described by theory in two sample problems, and demonstrate the tradeoff between sparsity and accuracy in a third problem.

4.1 L_1-Regularized Linear Fixed Point

In this section we define the L_1 regularized linear fixed point and review the LARS-TD (Kolter and Ng, 2009) algorithm.

Kolter and Ng (2009) introduced the problem

$$w = \arg\min_u \frac{1}{2}\|\Phi u - T\Phi w\|_\sigma^2 + \beta\|u\|_1, \tag{4.1}$$

which can be rewritten as the L_1 *regularized linear fixed point* problem

$$\Phi w = \Pi_{\{1,\beta\}}^\sigma T\Phi w.$$

where $\Pi_{\{1,\beta\}}^\sigma$ is the L_1 *regularized least-squares projection operator* for some $\beta > 0$. The above is simply the linear fixed point of equation (2.12) with the addition of L_1 regularization. Ghavamzadeh et al. (2011) analyze a generalization of the operator $\Pi_{\{1,\beta\}}^\sigma T$, which they show to be a contraction, ensuring the existence and uniqueness of the fixed point Φw (note that w by itself is not required to be unique).

Algorithms which solve for the L_1 regularized linear fixed point include LARS-TD (Kolter and Ng, 2009) and LC-TD (Johns et al., 2010). LARS-TD follows a homotopy method, in which the regularization parameter is slowly shrunken to the desired value, maintaining at all times a set of fixed point criteria with respect to the regularization parameter. LC-TD (Johns et al., 2010) restates the same fixed point criteria as a *linear complementarity problem* (LCP), which can be solved by a variety of existing solvers. As the fixed point criteria are also critical to understanding the fixed point behavior of our online algorithm, we review their derivation here.

The fixed point conditions are obtained by first taking the subdifferential of $L(u) = \frac{1}{2}\|\Phi u - T\Phi w\|_\sigma^2 + \beta\|u\|_1$:

$$\nabla L = \Phi^T\Sigma(\Phi u - T\Phi w) + \beta\,\mathrm{SGN}(u)$$

where $\mathrm{SGN}(u)$ is a set-valued function defined componentwise as

$$\mathrm{SGN}(u)_i = \begin{cases} \{+1\}, & u_i > 0 \\ [-1, 1], & u_i = 0 \\ \{-1\}, & u_i < 0, \end{cases}$$

and Σ is the diagonal matrix with $diag(\Sigma) = \sigma$, the stationary distribution.[1]

Setting the subdifferential to zero (since the subdifferential is set-valued, the actual requirement is that zero is in the subdifferential) yields the optimality conditions

$$[\Phi^T \Sigma(\Phi u - T\Phi w)]_i \begin{cases} = -\beta, & u_i > 0 \\ \in [-\beta, \beta], & u_i = 0 \\ = +\beta, & u_i < 0. \end{cases}$$

At the fixed point we require that $u = w$, thus the fixed point is characterized by the above conditions with u replaced everywhere by w:

$$[\Phi^T \Sigma(\Phi w - T\Phi w)]_i \begin{cases} = -\beta, & w_i > 0 \\ \in [-\beta, \beta], & w_i = 0 \\ = +\beta, & w_i < 0. \end{cases} \tag{4.2}$$

Any vector w satisfying these conditions minimizes the right-hand side of equation (4.1), yielding the fixed point Φw.

4.2 L_1-Regularized Linear TD

In this section we develop and analyze a new, iterative algorithm for finding the regularized linear fixed point. LARS-TD and LC-TD are inherently *batch* methods; samples are obtained prior to running the algorithm, and each iteration generates an update based on the entire sample corpus. Our new algorithm can be used with batch data but also permits *online* learning of the value function.

4.2.1 Linear TD as Gradient Descent

Our new algorithm can be motivated as a gradient descent algorithm in the spirit of linear temporal difference learning (TD) (Sutton, 1988). Linear TD can be viewed, somewhat loosely, as a gradient descent method minimizing the "objective" function

[1] Kolter and Ng (2009) and Johns et al. (2010) omit Σ. Here we explicitly weight by the stationary distribution to ensure consistency between LARS-TD and our online algorithm described in section 4.2.2. For N samples, Σ can be replaced by 1/N.

$\frac{1}{2}\|\Phi u - T\Phi w\|_\sigma^2$. The linear TD update is derived from the standard gradient learning rule by taking the gradient of the objective function with respect to u, together with the constraint $u = w$:

$$\begin{aligned} w_{t+1} &:= w_t - \alpha_t \nabla_u \left(\frac{1}{2} \|\Phi u - T\Phi w_t\|_\sigma^2 \right) \\ &:= w_t + \alpha_t \nabla_u (\Phi u - T\Phi w_t)^T \Sigma (T\Phi w_t - \Phi u) \\ &:= w_t + \alpha_t \ \Phi^T \Sigma (T\Phi w_t - \Phi w_t) \end{aligned} \tag{4.3}$$

Here α_t is a (possibly time-dependent) learning rate parameter. In online learning, the values of Φw and $T\Phi w$ are replaced by samples, yielding a stochastic estimate of the gradient; also, Σ is implicit in the online sampling and no longer appears in the update.

4.2.2 Linear TD with Soft-Thresholding

We now turn to a class of gradient descent algorithms which use thresholding to effect L_1 regularization. Introduced by Donoho and Johnstone (1994), the *soft-threshold shrinkage operator* Ψ_ν is defined as

$$\Psi_\nu(x) = \text{sign}(x) \odot \max\{|x| - \nu, 0\} \tag{4.4}$$

where $\nu > 0$. The sign and max operations are componentwise, and $\odot$ signifies componentwise multiplication. In words, the shrinkage operator reduces the magnitude of each element of x by ν, truncating to zero if the magnitude was already $\leqslant \nu$.

A number of authors (Daubechies et al., 2004; Duchi and Singer, 2009; Hale et al., 2007; Langford et al., 2008) make use of this operator composed with gradient descent in an iterative fixed point method for solving the regression problem (2.9):

$$w_{t+1} := \Psi_{\alpha\beta}(w_t - \alpha \ \Phi^T(\Phi w_t - x)), \tag{4.5}$$

where α is a (typically small) positive step size. Duchi and Singer (2009) establish convergence (under mild conditions) for the online variant, replacing the gradient $\Phi^T(\Phi w - x)$ with a stochastic estimate.

An important fact for our purposes is that the soft-threshold shrinkage operator is non-expansive in p-norm (Hale et al., 2007, Lemma 3.2).

In this section we introduce the most straightforward application of the soft-threshold shrinkage operator in the context of linear TD, and show that the fixed point of the resulting iteration satisfies the LARS-TD fixed point conditions (4.2). If we compose the soft-threshold shrinkage operator with the linear TD gradient update (4.3) we obtain

$$w_{t+1} := \Psi_{\alpha_t \beta}(w_t + \alpha_t \, \Phi^T \Sigma (T\Phi w_t - \Phi w_t)). \tag{4.6}$$

This update generates the algorithm **L1TD**, shown in algorithm 1.

We give the following lemma concerning the fixed point of (4.6); the subsequent proof is nearly identical to the proof given by Hale et al. (2007, Proposition 3.1) in the regression setting:

Lemma 1. *A vector w^* is a solution to the LARS-TD fixed point problem (4.1) with regularization parameter β if and only if, for any $\eta > 0$,*

$$w^* = \Psi_{\eta\beta}(w^* + \eta \Phi^T \Sigma (T\Phi w^* - \Phi w^*)). \tag{4.7}$$

Proof. Recall that fixed points of LARS-TD are completely characterized by the conditions given in equation (4.2). Consider the i^{th} element $w^*[i]$, and let $g[i] = [\Phi^T \Sigma (T\Phi w^* - \Phi w^*)]_i$, the i^{th} element of the "gradient". By equations (4.7) and (4.4) we have

$$w^*[i] = \operatorname{sign}(w^*[i] + \eta g[i]) \max\{|w^*[i] + \eta g[i]| - \eta\beta, 0\}. \tag{4.8}$$

Going forward, the max expression in (4.8) is nonnegative, and therefore $w^*[i] \neq 0$ implies $\text{sign}(w^*[i] + \eta g[i]) = \text{sign}(w^*[i])$. If $w^*[i] > 0$, then

$$\begin{aligned} w^*[i] &= 1(w^*[i] + \eta g[i] - \eta\beta) \\ g[i] &= \beta, \end{aligned}$$

as required by the LARS-TD fixed point conditions.

On the other hand, if $w^*[i] > 0$ and $g[i] = \beta$,

$$\begin{aligned} & \text{sign}(w^*[i] + \eta g[i]) \max\{|w^*[i] + \eta g[i]| - \eta\beta, 0\} \\ = \; & \text{sign}(w^*[i] + \eta\beta)(w^*[i] + \eta\beta - \eta\beta) \\ = \; & w^*[i], \end{aligned}$$

as required by equation (4.8).

A similar argument shows that, if $w^*[i] < 0$, equation (4.8) implies $g[i] = -\beta$, and $g[i] = -\beta$ implies equation (4.8).

Finally, if $w^*[i] = 0$, either $g[i] = 0$ or $\text{sign}(w^*[i] + \eta g[i]) = \pm 1$ and therefore equation (4.8) is satisfied if and only if

$$\begin{aligned} \max\{|w^*[i] + \eta g[i]| - \eta\beta, 0\} &= 0 \\ |0 + \eta g[i]| - \eta\beta &\leqslant 0 \\ |g[i]| &\leqslant \beta \\ g[i] &\in [-\beta, \beta]. \end{aligned}$$

□

4.2.3 A Modification

The L1TD update (4.6) is instantly recognizable as an amalgam of linear TD with soft-thresholding, and its form is convenient for proving equivalence with the L_1-regularized linear fixed point (4.1), but it turns out not to be particularly amenable

Algorithm 1 L1TD

Input: φ, α, β.
Output: Approximation weights w.
Initialize $w = 0, t = 0$.
repeat
  Obtain samples s, r, s'
  $A \leftarrow \varphi(s)(\varphi(s')^T - \varphi(s)^T))$
  $b \leftarrow \varphi(s)r$
  $w \leftarrow \Psi_{\alpha(t)\beta}(w + \alpha(t)(Aw + b))$.
until convergence.

Algorithm 2 L1TDAlt

Input: $\varphi, \alpha, \beta, \eta$.
Output: Approximation weights w.
Initialize $v = 0, w = 0, t = 0$.
repeat
  Obtain samples s, r, s'
  $A \leftarrow \varphi(s)(\varphi(s')^T - \varphi(s)^T))$
  $b \leftarrow \varphi(s)r$
  $v \leftarrow v + \alpha(t)(w + \eta(Aw + b) - v)$
  $w \leftarrow \Psi_{\eta\beta}(v)$.
until convergence.

for a proof of convergence. In this section, therefore, we introduce a slightly altered update, which is easily shown to have the same fixed point as the above, and for which we prove convergence. While we have no formal proof of convergence of (4.6), its empirical behavior (see section 4.3) closely matches that of the modified update, and its common pedigree with the modified update is a strong argument for convergence.

A first step in developing our alternate algorithm is to decompose the update into its two basic operations, and reorder them to produce a "half-phase" update. We introduce a new variable, v, with $w_t = \Psi_{\eta\beta}(v_t)$, and we fix $\eta > 0$ at some constant value. The half-phase update equation generating the sequence v is

$$\begin{aligned} v_{t+1} &:= w_t + \eta\Phi^T\Sigma(T\Phi w_t - \Phi w_t) \\ &= \Psi_{\eta\beta}(v_t) + \eta\Phi^T\Sigma(T\Phi\Psi_{\eta\beta}(v_t) - \Phi\Psi_{\eta\beta}(v_t)), \\ &= H(v_t). \end{aligned} \tag{4.9}$$

Here we take advantage of the fact that the new sequence is dependent solely on v to define a new fixed point operator H. We can recover the original sequence w by writing $w_{t+1} := \Psi_{\eta\beta}(v_{t+1})$ and expanding v_{t+1} by equation (4.9). We denote the fixed point of the new sequence v^*, and note that it is related to w^* by the mappings $w^* = \Psi_{\eta\beta}(v^*)$, and $v^* = w^* + \eta\Phi^T\Sigma(T\Phi w^* - \Phi w^*)$.

Now we introduce the new iteration, which is more "conservative" in its update, mixing the previous solution with the update due to H according to a time-dependent step size α:

$$v_{t+1} := v_t + \alpha_t(H(v_t) - v_t). \tag{4.10}$$

This update gives rise to the algorithm **L1TDAlt** given in Algorithm 2. Note that the fixed point v^* of the modified iteration is characterized by $H(v^*) - v^* = 0$, that is, at the fixed point of H. Thus fixed points of the L1TD iteration are equivalent to fixed points of the L1TDAlt iteration via a simple transform.

4.2.4 Convergence

We now turn to the question of convergence. In particular, we are interested in convergence of the sequence v in the online setting, in which the operator H is replaced by noisy samples of H. Given sample transitions (s_t, r_t, s'_t) and a set of basis functions $\{\varphi_1\ \varphi_2\ \ldots\ \varphi_k\}$, the algorithm performs the updates

$$\begin{aligned} v_{t+1} &:= v_t + \alpha_t(w_t + \eta\varphi(s_t)(r_t + \gamma\varphi(s'_t)^T w_t - \varphi(s)^T w_t) - v_t) \\ w_{t+1} &:= \Psi_{\eta\beta}(v_{t+1}). \end{aligned} \tag{4.11}$$

Our convergence result depends on the following assumptions:

Assumption 1. *The Markov reward process $M = (S, P, R, \gamma)$ is finite and mixing, with stationary distribution σ.*

Assumption 2. *The sequence (s_t, r_t, s'_t) is sampled i.i.d. from the stationary distribution, σ, of M. That is, $s_t \sim \sigma$, $r_t = R(s_t)$, and s'_t is obtained by making a*

stochastic transition in accordance with P.

Assumption 3. *The columns of the matrix Φ comprise a linearly independent set of basis functions evaluated at all states in S. In particular, this implies that Φ is full rank.*

Assumption 4. *The sequence α_t is predetermined and non-increasing, with $\sum_{t=0}^{\infty} \alpha_t = \infty$ and $\sum_{t=0}^{\infty} \alpha_t^2 < \infty$.*

We now give the following theorem regarding convergence:

Theorem 2. *Consider the sequence v_t as defined in (4.11). Then, under assumptions 1 - 4, v_t converges almost surely as $t \to \infty$.*

Proof. We first restate the sequence v using additional terms. Given sample transitions (s_t, r_t, s_t') and a set of basis functions $\{\varphi_1 \ \varphi_2 \ \dots \ \varphi_k\}$, the algorithm performs the updates

$$v_{t+1} := v_t + \alpha_t(\Psi_{\eta\beta}(v_t) + \eta(A(X_t)\Psi_{\eta\beta}(v_t) + b(X_t)) - v_t), \tag{4.12}$$

where we define

$$\begin{aligned} X_t &= \{\varphi_t, r_t, \varphi_{t+1}\}, \\ A(X_t) &= \varphi_t(\gamma\varphi_{t+1} - \varphi_t)^T, \text{ and} \\ b(X_t) &= \varphi_t r_t. \end{aligned}$$

We define

$$\begin{aligned} A &= \Phi^T\Sigma(\gamma P\Phi - \Phi); \\ b &= \Phi^T\Sigma R. \end{aligned}$$

and note that

$$E_\sigma[A(X_t)] = \Phi^T\Sigma(\gamma P\Phi - \Phi) = A, \tag{4.13}$$

$$\text{and} \qquad E_\sigma[b(X_t)] = \Phi^T\Sigma R = b. \tag{4.14}$$

Here $E_\sigma[\cdot]$ denotes the expectation with respect to the stationary distribution of the Markov process.

Our proof will follow the ODE method as outlined by Borkar (2008). For this purpose, we will rewrite our update in the form

$$v_{t+1} := v_t + \alpha_t[h(v_t) + M_{t+1}],$$

and demonstrate that h and M satisfy the assumptions needed to apply theorem 2 of Borkar (2008, p. 15).

Let:

$$\begin{aligned} h(v) &= H(v) - v \\ &= \Psi_{\eta\beta}(v) + \eta(A\Psi_{\eta\beta}(v) + b) - v, \\ M_{t+1} &= \Psi_{\eta\beta}(v) + \eta(A(X_t)\Psi_{\eta\beta}(v) + b(X_t)) - v - (H(v) - v) \\ &= \eta[(A(X_t) - A)\Psi_{\eta\beta}(v_t) + b(X_t) - b] \end{aligned}$$

We now need to satisfy assumptions A1–A3 and A5 of Borkar (2008): we must show (a) that the function h is Lipschitz, (b) that the function $h_\infty = \lim_{r\to\infty} h(rx)/r$ exists, and (c) that the origin is an asymptotically stable equilibrium for the ODE $\dot{v}(t) = h_\infty(v(t))$. We must also show (d) that $\{M_t; \mathcal{F}_t\}$ where $\mathcal{F}_t = (M_0, v_0, M_1, v_1, \ldots M_t, v_t)$ is a martingale difference sequence, that is, that the expectation of M_{t+1} given $\mathcal{F}_t$ is zero, and (e) that, for some $C_0 < \infty$ and any initial condition v_0, $E[\|M_{t+1}\|^2|\mathcal{F}_t] \leqslant C_0(1 + \|v_t\|^2)$.

For (a), to show Lipschitz continuity of h we need C such that $\|h(v) - h(v')\| \leqslant$

$C\|v - v'\|, \forall v, v'$. We have

$$\begin{aligned}
& \|h(v) - h(v')\| \\
= \; & \|\Psi_{\eta\beta}(v) + \eta(A\Psi_{\eta\beta}(v) + b) - v \\
& \qquad - (\Psi_{\eta\beta}(v') + \eta(A\Psi_{\eta\beta}(v') + b) - v')\| \\
= \; & \|(I + \eta A)(\Psi_{\eta\beta}(v) - \Psi_{\eta\beta}(v')) + (v' - v)\| \\
\leqslant \; & \|(I + \eta A)\|\|\Psi_{\eta\beta}(v) - \Psi_{\eta\beta}(v')\| + \|v' - v\| \\
\leqslant \; & \|(2I + \eta A)\|\|v - v'\|.
\end{aligned}$$

The function h is thus Lipschitz with constant $C = \|2I + \eta A\|$. Here we make use of the triangle inequality, the non-expansiveness of Ψ, and the Cauchy-Schwarz inequality as generalized to matrix norms.

For (b), we have

$$\begin{aligned}
h_\infty(v) &= \lim_{r\to\infty} h(rx)/r \\
&= \lim_{r\to\infty} [\Psi_{\eta\beta}(rv) + \eta(A\Psi_{\eta\beta}(rv) + b) - rv]/r \\
&= \lim_{r\to\infty} [(I + \eta A)(rv - \eta\beta\,\mathrm{sign}(v)) + \eta b - rv]/r \\
&= \lim_{r\to\infty} [(I + \eta A)(rv - \eta\beta\,\mathrm{sign}(v)) - rv]/r \\
&= (I + \eta A)v - v \\
&= \eta A v.
\end{aligned}$$

The third step above follows from the fact that all non-zero elements of v are shrunk by no more than $\eta\beta$; as r grows large, all non-zero elements of v are shrunk by this amount, while all zero elements are shrunk by zero. Now (c) follows directly from standard ODE theory and the fact that A is negative definite (by assumption 3 and Lemma 6.6 of Bertsekas and Tsitsiklis (1996, p. 300)).

For (d), we note that under assumption 2, our X_t are sampled according to the stationary distribution, and are thus independent of $\mathcal{F}_t$. Using the definition of M

and equations (4.13, 4.14) we obtain $E[M_{t+1}|\mathcal{F}_t] = \eta((E_\sigma[A(X_t)] - A)\Psi_{\eta\beta}(v_t) + E_\sigma[b(X_t)] - b) = 0$, as required.

Finally, for (e) we note that, given the finiteness assumption on our MRP, the second moments of the reward and our basis functions are finitely bounded; by extension, if we consider M_{t+1} as a linear function of $\Psi_{\eta\beta}(v_t)$, its coefficients are bounded, and its square is a quadratic function of $\Psi_{\eta\beta}(v_t)$ with bounded coefficients. The result follows trivially from the triangle inequality and the non-expansiveness of Ψ. □

Our theorem states convergence in the sense of Theorem 2 of Borkar (2008), under which convergence may be to a local optimum. While we do not yet have a proof that convergence is to the fixed point, our experiments show fixed point convergence in every case.

Also, assumption 2 would normally be violated in the general online setting. We make this assumption in order to fit our update to the structure of the proof template given by Borkar (2008). The machinery needed to cope with Markov noise in the online setting is somewhat involved; one approach can be found in section 4.4 of Bertsekas and Tsitsiklis (1996).

4.3 Experiments

Our proofs assume a decaying step size but we use a constant step size in our experiments[2]. We also allowed linearly dependent features in our experiments and considered episodic tasks. We found that these issues did not pose a problem, and expect that theory can be extended to these cases.

[2] Constant step sizes are commonly used to speed up improvement in the value function, possibly at the expense of some oscillation around the true solution. This practice preceded its theoretical justification by Borkar and Meyn (2000). We defer extending their approach to the regularized case for future work.

We performed experiments on two benchmark problems, Blackjack and Mountain Car, to show the convergence of the online algorithm. We also performed experiments on the Inverted Pendulum problem to illuminate the tradeoff between approximation performance and sparsity. We used both algorithm 1 (L1TD) and algorithm 2 (L1TDAlt) in our experiments; the results were essentially identical between the two algorithms in all cases, and for this reason we give results for only one algorithm in most cases.

4.3.1 Blackjack

We tested a version of the bottomless-deck blackjack problem from Sutton and Barto (1998), evaluating the policy in which the player stands on 20 or 21 and hits on anything else. In this problem, there are 200 states arising from the cross product of three variables: player total ([11...21]), dealer showing card value ([A...10]), and the presence of a usable ace in the player's hand (true/false). In our version, there are three additional states for win/lose/draw, and the player continues playing infinitely; after reaching a win/lose/draw state, the next state is drawn from the distribution of possible starting deals. With this modification, the problem is mixing with a well-defined stationary distribution. The player receives a reward of $+1$ for a win and -1 for a loss; all other rewards are zero. A discount factor of 0.95 keeps expected player losses bounded.

We provide the learner with 219 features: a bias term, an indicator for each of {win, lose}, indicator functions for all possible contiguous ranges of the player's current total times an indicator for whether the player has a usable ace, and likewise indicator functions on the dealer's show card times the ace indicator. Despite having more features than states, the rank of the feature matrix (the dimension of the basis) is only 41.

The fixed point for this problem was found via LARS-TD using full model infor-

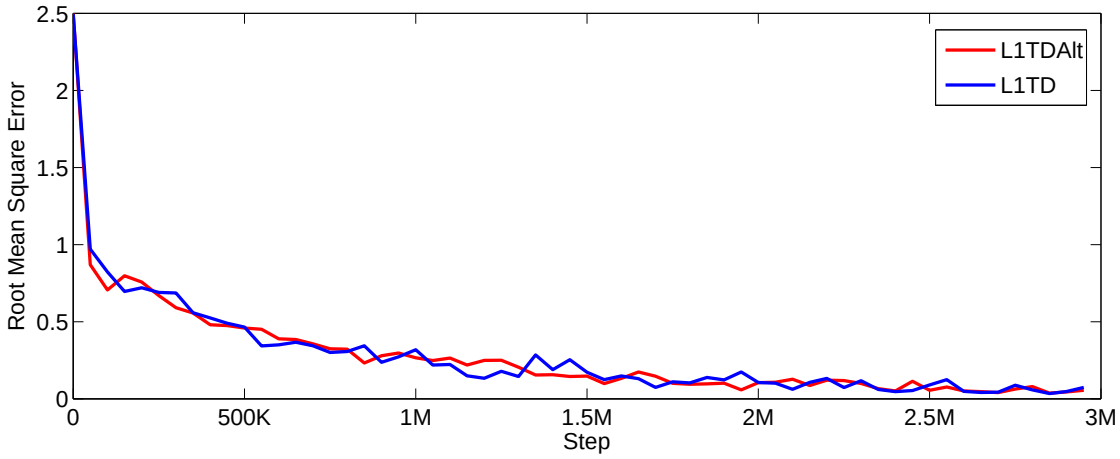


FIGURE 4.1: Blackjack: σ-weighted root mean square error between online solutions and the fixed point value function.

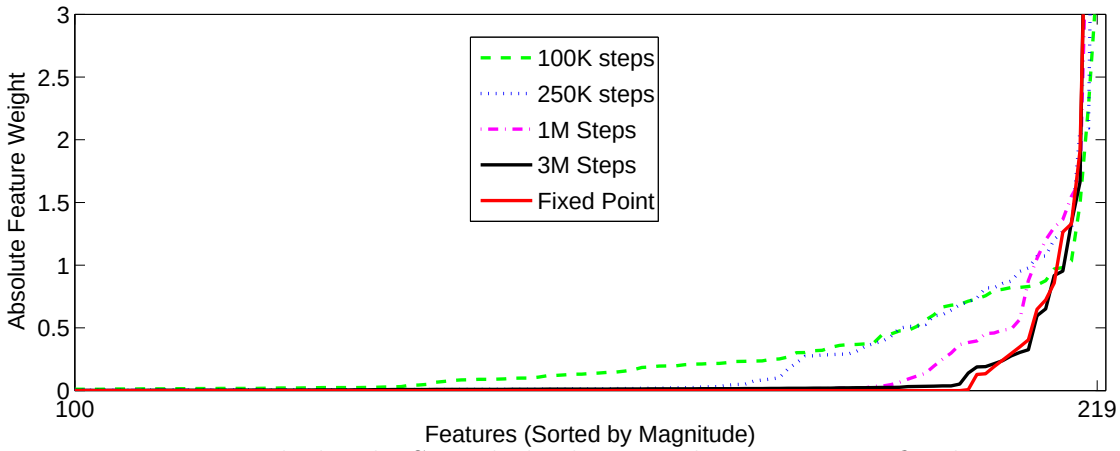


FIGURE 4.2: Blackjack: Sorted absolute weights, L1TD vs. fixed point.

mation. Figure 4.1 shows the progress of the online algorithms vs. the fixed point when the regularization coefficient is set to 0.001. The step size for L1TD was set to 0.01, while the "fixed" step size (η) and the regular step size (α) were both set to 0.1 for L1TDAlt. Each algorithm was run for 3 million iterations in one continuous trajectory. The online algorithms quickly reach solutions which are near the fixed point, and make gradual progress thereafter.

Figure 4.2 shows how the sparsity of the solution changes over time. The plot

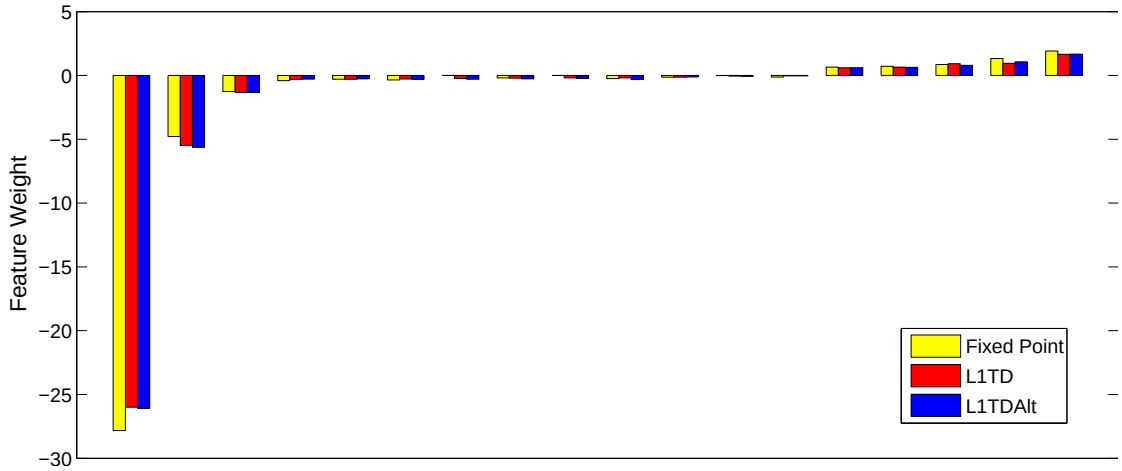


FIGURE 4.3: Blackjack: Comparison of weights assigned to the top 18 features by L1TD, L1TDAlt, and the fixed point.

shows a curve of the sorted absolute weight values after 100,000, 250,000, 1 million, and 3 million iterations of L1TD, as well as the weight curve for the fixed point solution (which has 16 non-zero weights). The figure has been scaled to emphasize the region of interest; the actual largest magnitude weights are around 28. The figure shows that, while L1TD produces few actual zeroes in this instance, over time it concentrates more and more weight into fewer and fewer features; most of the remaining non-zero features (which arise inevitably due to the stochastic updates) can be truncated without appreciably affecting the approximation.

Figure 4.3 shows the weights assigned to the top 18 features by L1TD, L1TDAlt, and LARS-TD; the strong agreement across the most significant features demonstrates the convergence of the online solutions to the fixed point qualitatively (in terms of the features it selects) as well as quantitatively (in terms of the learned value function).

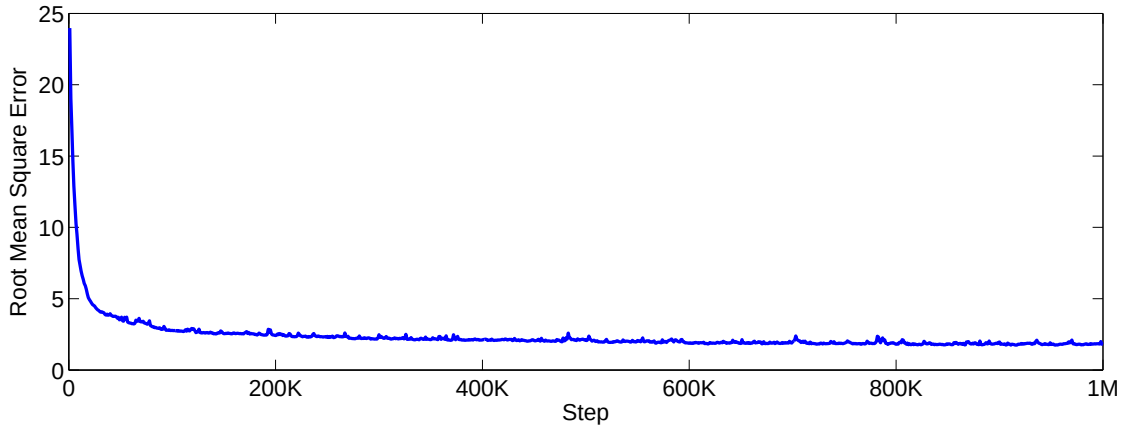


FIGURE 4.4: Mountain Car: Root mean square error between L1TD and LARS-TD average value functions.

4.3.2 Mountain Car

We also tested the familiar mountain car domain, in which an underpowered car must back up a small hill to gain enough potential energy to climb a larger one. The agent receives a -1 penalty for every step it takes until it reaches the goal region at the top of the hill. We chose to evaluate this task using a simple policy in which the car always accelerates in whichever direction it is already moving, or forward if it is at rest; while suboptimal, this policy ensures that the car will reach the goal region from any valid starting state. We used a discount factor of 0.99.

Our features for this task were composed of 1365 radial basis functions of various widths arranged in differently spaced grids over the state space together with a constant (bias) term. As movement in the mountain car domain is deterministic, we sampled trajectories by first selecting a starting state uniformly at random, and then following policy to the goal (on average, about 50 steps). A "ground truth" estimate of the fixed point was obtained by taking the average of the approximation weights returned by 100 runs of LARS-TD using independently collected samples from 2000 trajectories (approximately 100,000 samples for each run); the regularization coeffi-

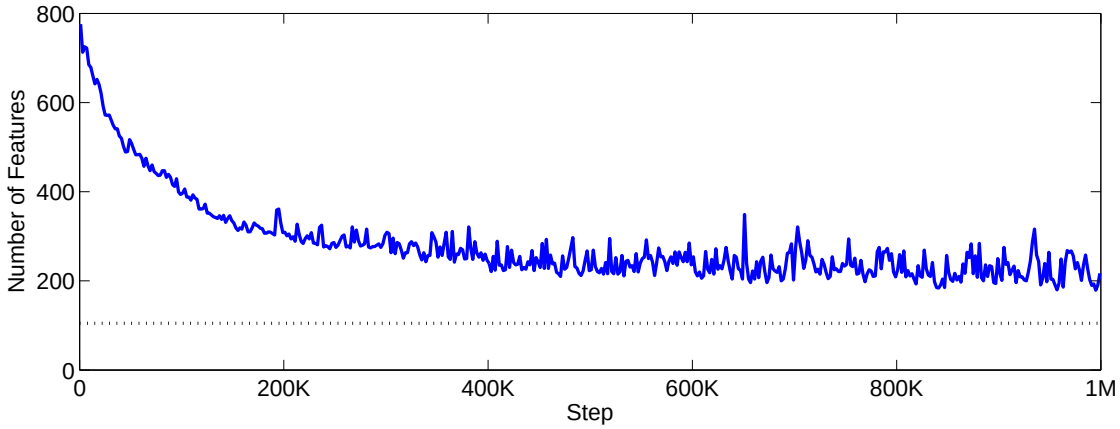


FIGURE 4.5: Mountain Car: Number of features comprising 99% of total feature weight vs. number of steps.

cient was 0.01. A separate testing set of 2000 sample trajectories was collected and used in comparing the online algorithm (L1TD) with the ground truth estimate.

For online learning, we again start with a state selected uniformly at random, and follow policy to the goal; upon reaching the goal, we select a new state at random, and follow policy to the goal, etc. The algorithm was run for 1 million iterations using a fixed step size of 0.001. Figure 4.4 shows the root mean square error of the online value estimates for our test set versus the ground truth fixed point estimate; as with blackjack, the online algorithm quickly moves near the fixed point estimate, and then makes gradual improvements thereafter. Figure 4.5 shows generally how the sparsity of the online solution compares to the sparsity attained by LARS-TD; the plot gives the count of the largest features which comprise 99% or more of the total absolute weight value vs. the number of steps taken. The dotted black line in this figure shows the average number of features retained by LARS-TD in the ground truth estimation runs.

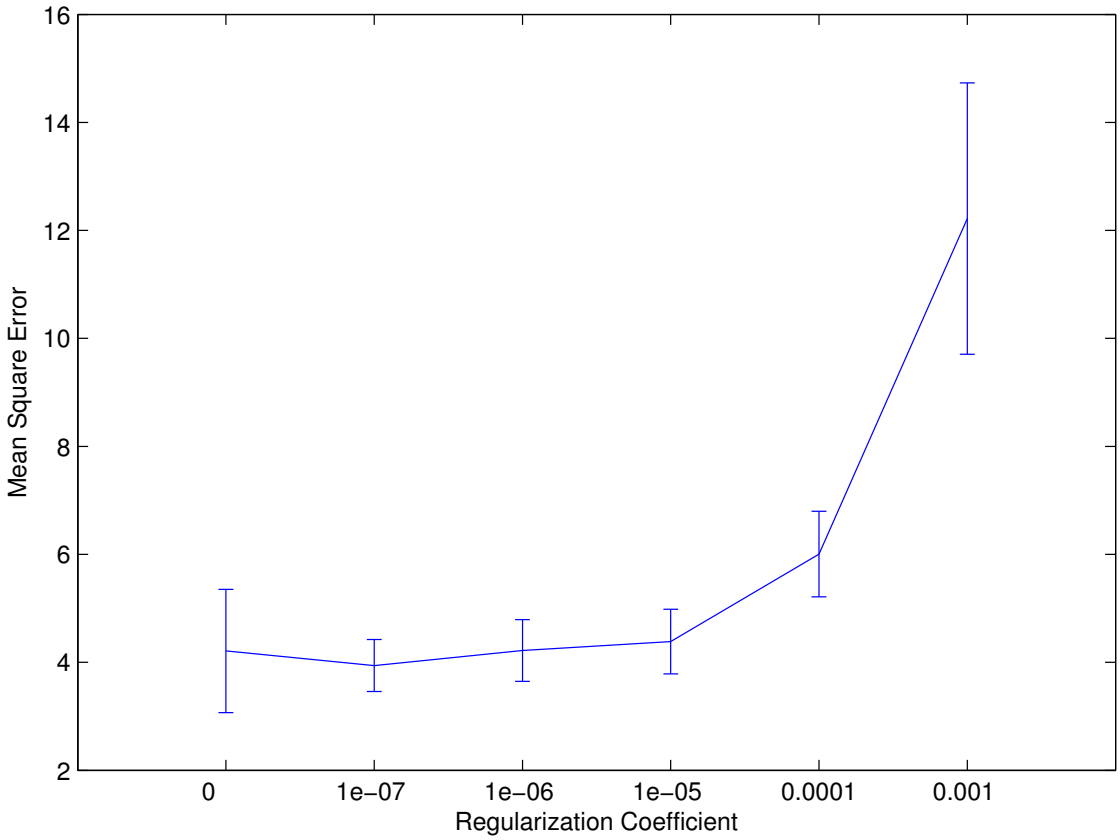


FIGURE 4.6: Pendulum: Average mean square error between L1TDAlt and ground truth.

4.3.3 Pendulum

Finally, we tested the inverted pendulum domain, in which the task is to balance an inverted pendulum by applying forces to the cart to which it is attached. There are three (noisy) actions, applying force to the left or right or no force. The agent receives a -1 penalty if and when the pendulum falls over. We evaluate this task using a simple, suboptimal policy that nevertheless balances the pendulum fairly well. We used a discount factor of 0.95.

For this task we construct a feature set by first transforming the state vector via PCA transform and a scaling matrix such that the resulting vectors live mostly in a 2π radius box around the origin. We then construct a wavelet-inspired basis

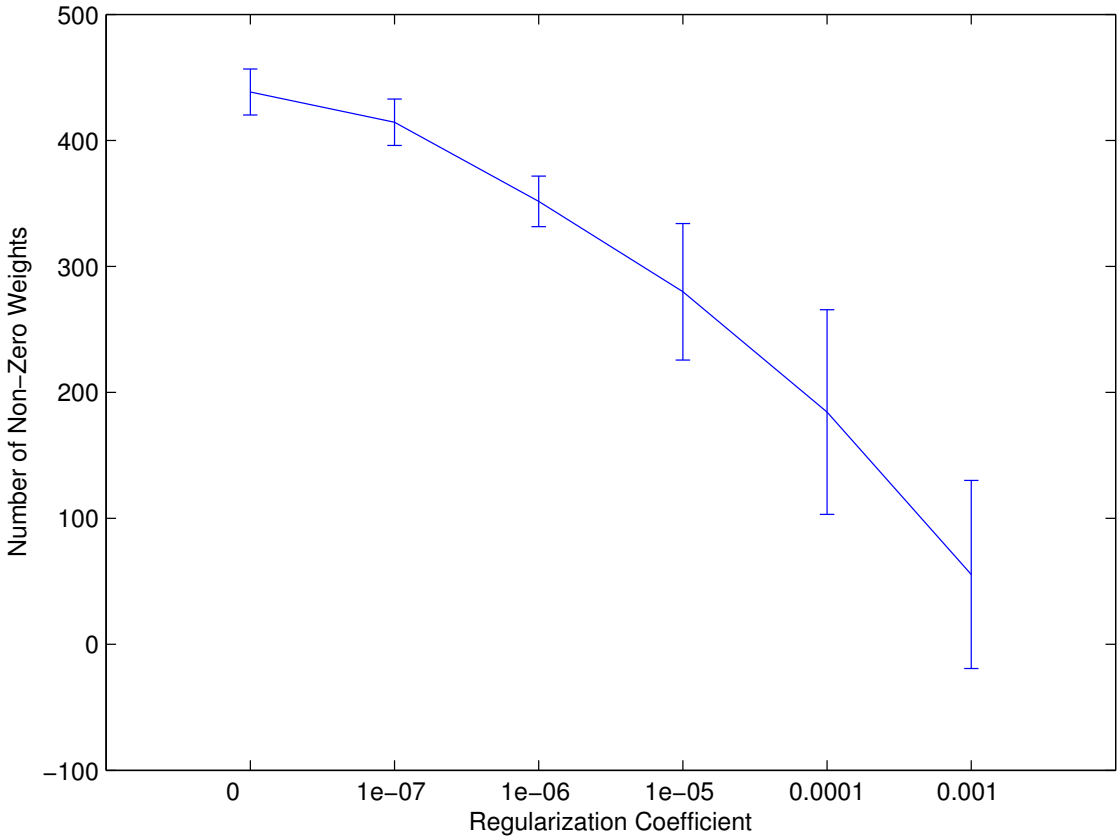


FIGURE 4.7: Pendulum: Average number of non-zero features.

using simple harmonic functions as the seed. The total number of features is 511. A set of 10,000 test samples was obtained from multiple on-policy trajectories in which the initial state was a vector randomly perturbed around the point at which the pendulum is balanced and unmoving, following the policy thereafter until the pendulum fell over. A ground truth estimate of the true value function at each sample was obtained by Monte Carlo sampling.

Our goal with this experiment was to demonstrate the performance of the algorithm for different values of regularization parameter as well as the sparsity induced by regularization. For each value of the regularization parameter, L1TDAlt was run ten times, each time for 5 million steps, starting with the state vector at the origin, thereafter following policy, resetting to the origin when the pendulum falls over. The

step size (α) used was 0.02 while the shrinkage coefficient (η) was 0.5. Figure 4.6 shows the mean squared error of the online value estimates for our test set versus the ground truth estimate for various settings of the regularization parameter. As expected, small values of the regularization parameter lead to small errors. Note that the best result is obtained when the regularization is set to 10^{-7}, where the error and variance are lower than for the result with no regularization, showing a modest benefit in average performance and substantial reduction of variance. Figure 4.7 shows how the sparsity of the online solution varies with the regularization parameter.

4.4 Discussion

In this chapter we have presented a novel regularized, online stochastic approximation algorithm, based on an iterative update equation whose fixed point corresponds to the L_1 regularized linear fixed point also shared by LARS-TD and LC-TD. We validated the ability of the online algorithm to converge to the fixed point described by theory in two sample problems, and demonstrated the tradeoff between sparsity and accuracy in a third problem.

More generally, we established a connection between iterative and online algorithms for convex approximation with regularization and temporal difference learning with regularization. The soft-threshold shrinkage method extends naturally to L_2 and other forms of regularization, which can be applied to TD and analyzed following the approach taken above. Thus we hope to provide a template for future work in regularization algorithms for online TD learning.

5

Simultaneous Feature Selection and Optimization via Least Angle Regression

In this chapter we change direction and examine the problem of policy optimization with feature selection in a batch learning setting. Most approaches to feature selection in reinforcement learning decouple the feature selection task from the optimization task. For instance, Kolter and Ng (2009) used LARS-TD within an LSPI-like (Lagoudakis and Parr, 2003) policy iteration loop. For each policy, a new, sparse set of features is discovered. Upon selecting a new policy, this feature set is discarded and another feature set is learned *ab initio*.

Here we present a modification to the LARS-TD algorithm called *LARQ* (pronounced "lark"). LARQ, as LARS-TD, builds on the LARS algorithm (Efron et al., 2004) which incrementally builds an *active* set of features. The computational complexity of LARS is, largely, driven by the size of the active set of features; costs are linear in the size of the inactive set. The run time of LARS is, therefore, fairly insensitive to the addition of irrelevant features. Since relevance is not always known *a priori*, this a very desirable property in a feature selection algorithm – one can use

large, possibly overcomplete, feature sets, and let LARS pick the relevant features.

LARQ achieves *simultaneous* feature selection and optimization by changing policies within the LARS-TD loop. This combination has the satisfying property that it does not completely discard the investment in feature selection when policies change. It may also help give deeper insight into the relationship between the optimization, prediction, and feature selection demands that we would like to place upon reinforcement learners.

In the following sections we will review the LARS-TD algorithm, then present the LARQ algorithm. We discuss the conditions under which LARQ will converge, and present some experimental results demonstrating the effectiveness of LARQ in practice.

5.1 LARS-TD

LARS-TD finds the L_1-regularized linear fixed point described in the previous chapter (4.1):

$$w = \arg\min_u \frac{1}{2}\|\Phi u - T\Phi w\|_\sigma^2 + \beta\|u\|_1.$$

While the above is not a convex optimization problem, Kolter and Ng (2009) show that it is possible to find the fixed point using an iterative procedure adapted from LARS (Efron et al., 2004), in which features are added or subtracted from a solution which is moving in steps towards the fixed point. The LARS-TD algorithm, like the LARS algorithm, uses a homotopic approach in which the regularization coefficient (β) is shrunk from some initially large value to the desired value. The algorithm maintains a set of invariant constraints which suffice to ensure the correctness of the LARS-TD solution for any intermediate value (we notate intermediate values as $\bar{\beta}$). These invariant constraints are simply the L_1-regularized linear fixed point

optimality conditions (4.2) evaluated at $\bar{\beta}$:

$$[\Phi^T \Sigma(\Phi w - T\Phi w)]_i \begin{cases} = -\bar{\beta}, & w_i > 0 \\ \in [-\bar{\beta}, \bar{\beta}], & w_i = 0 \\ = +\bar{\beta}, & w_i < 0. \end{cases}$$

If LARS-TD is run "to completion" (i.e., with $\beta = 0$), it will in fact find all fixed points for all values of β.

From here on out we will drop the use of the σ-weighted norm and omit Σ; for N samples from the stationary distribution, Σ should be set to $1/N$. As long as N is constant for any given problem, it is safe to omit Σ, but implementors should note that, without explicit weighting, it is meaningless to compare values of β for different sample sizes.

To understand how LARS-TD maintains the invariants at all times, let us first introduce some new terms and then rewrite the invariant constraints in those terms. First, the *correlation* c_i of features φ_i with the Bellman residual is given by

$$c_i = \varphi_i^T (T\hat{V} - \hat{V}) = \varphi_i^T (R + \gamma \Phi' w - \Phi w).$$

Second, the *active set*, $\mathcal{I}$, is the set of all features for which the corresponding weight is non-zero. Since the weights of all inactive features are zero, we can also write our correlations as

$$c_i = \varphi_i^T (R + \gamma \Phi'_{\mathcal{I}} w_{\mathcal{I}} - \Phi_{\mathcal{I}} w_{\mathcal{I}}).$$

The LARS-TD invariants can then be summarized as follows:

1. All features in the active set share the same absolute correlation: $\forall i \in \mathcal{I}, |c_i| = \bar{\beta}$.

2. All features in the active set have greater absolute correlation than any feature not in the active set: $\forall i \notin \mathcal{I}, |c_i| < \bar{\beta}$.

3. The signs of the weight coefficients of any feature in the active set matches the sign of its correlation: $\forall i \in \mathcal{I}, \text{sign}(c_i) = \text{sign}(w_i)$.

LARS-TD, then, proceeds in such a way as to maintain the above invariants while making progress towards the fixed point. The algorithm (as applied to state-action value function, or Q, approximation) is summarized in algorithm 3, but an explanation of the algorithm is given below.

First, we can note that all of our invariants hold at the start of the algorithm when our regression weights are all zero, and we add a single feature (the one with the greatest magnitude correlation). At each step, we must move so as to maintain the first invariant, keeping all of the correlations of features in the active set at the same magnitude. Let the change in the active set weights be $\alpha\Delta w_{\mathcal{I}}$, and let the new correlations of the active set features be

$$\begin{aligned} c'_{\mathcal{I}} &= \Phi_{\mathcal{I}}^T(R + (\gamma\Phi'_{\mathcal{I}} - \Phi_{\mathcal{I}})(w_{\mathcal{I}} + \alpha\Delta w_{\mathcal{I}})) \\ &= c_{\mathcal{I}} + \alpha\Phi_{\mathcal{I}}^T(\gamma\Phi'_{\mathcal{I}} - \Phi_{\mathcal{I}})\Delta w_{\mathcal{I}}. \end{aligned}$$

We want the correlations of the active set to shrink equally, say by some amount α, thus

$$\begin{aligned} c'_{\mathcal{I}} &= c_{\mathcal{I}} - \alpha\,\text{sign}(c_{\mathcal{I}}) \\ c_{\mathcal{I}} + \alpha\Phi_{\mathcal{I}}^T(\gamma\Phi'_{\mathcal{I}} - \Phi_{\mathcal{I}})\Delta w_{\mathcal{I}} &= c_{\mathcal{I}} - \alpha\,\text{sign}(c_{\mathcal{I}}) \\ \Delta w_{\mathcal{I}} &= (\Phi_{\mathcal{I}}^T\Phi_{\mathcal{I}} - \gamma\Phi_{\mathcal{I}}^T\Phi'_{\mathcal{I}})^{-1}\,\text{sign}(c_{\mathcal{I}}). \end{aligned}$$

Intuitively, the update direction Δw is the vector pointing from the current solution w to the LSTD fixed point solution using the active set features.

Given the above update direction, we want to take as large a step as possible without violating our constraints. To maintain invariant 2, we must add any feature φ_i not in the active set when its correlation with the residual matches the correlations

Algorithm 3 LARQ

Input: $\{s_i, a_i, r_i, s'_i\}$, γ, φ, β.
Output: Approximation weights w.

$w \leftarrow 0$
$\pi : \pi(s'_i) \leftarrow 1$
$\Phi : \Phi_{ij} \leftarrow \varphi_j(s_i, a_i)$
$c \leftarrow \Phi^T R$
$\{\bar{\beta}, i\} \leftarrow \{\max, \arg\max\}_j |c_j|$
$\mathcal{I} \leftarrow \{i\}$
while $\bar{\beta} > \beta$ **do**
  // Get features for next states following policy π:
  $\Phi'^{\pi} : \Phi'^{\pi}_{ij} \leftarrow \varphi_j(s_i, \pi(s_i))$
  // Find update direction:
  $\Delta w_{\mathcal{I}} \leftarrow (\Phi^T_{\mathcal{I}} \Phi_{\mathcal{I}} - \gamma \Phi^T_{\mathcal{I}} \Phi'^{\pi}_{\mathcal{I}})^{-1} \operatorname{sign}(c_{\mathcal{I}})$

  // Find step size for adding a feature to the active set:
  $d \leftarrow (\Phi^T (\Phi_{\mathcal{I}} - \gamma \Phi'^{\pi}_{\mathcal{I}}) \Delta w_{\mathcal{I}}$
  $\{\alpha_1, i_1\} \leftarrow \{\min, \arg\min\}^{+}_{j \notin \mathcal{I}} \left\{ \frac{c_j - \bar{\beta}}{d_j - 1}, \frac{c_j + \bar{\beta}}{d_j + 1} \right\}$
  // Find step size for removing a feature from the active set:
  $\{\alpha_2, i_2\} \leftarrow \{\min, \arg\min\}^{+,0}_{j \in \mathcal{I}} \left\{ -\frac{w_j}{\Delta w_j} \right\}$
  // Find min step size:
  $\alpha \leftarrow \min\{\alpha_1, \alpha_2, \bar{\beta} - \beta\}$

  // *** *This next section is omitted in LARS-TD* ***
  // Find step size for greedy policy update:
  $Q'^{\pi} \leftarrow \Phi'^{\pi}_{\mathcal{I}} w_{\mathcal{I}}$
  $\Delta Q'^{\pi} \leftarrow \Phi'^{\pi}_{\mathcal{I}} \Delta w_{\mathcal{I}}$
  for all actions a **do**
    $Q'^{a} : Q'^{a}_i \leftarrow \Phi(s'_i, a)_{\mathcal{I}} w_{\mathcal{I}}$
    $\Delta Q'^{a} : \Delta Q'^{a}_i \leftarrow \Phi(s'_i, a)_{\mathcal{I}} \Delta w_{\mathcal{I}}$
    $\{\alpha^a_3, i^a_3\} \leftarrow \{\min, \arg\min\}_i \left\{ -\frac{Q'^{a}_i - Q'^{\pi}_i}{\Delta Q'^{a}_i - \Delta Q'^{\pi}_i} \text{ such that } \Delta Q'^{a}_i > \Delta Q'^{\pi}_i \right\}$
  end for
  $\{\alpha_3, \pi'_3\} \leftarrow \{\min, \arg\min\}_a \{\alpha^a_3\}$
  $i_3 \leftarrow i_3^{\pi'_3}$
  $\alpha \leftarrow \min\{\alpha, \alpha_3\}$
  // *** *End of LARQ-only section* ***

  // Update weights, $\bar{\beta}$, and correlation vector:
  $w_{\mathcal{I}} \leftarrow w_{\mathcal{I}} + \alpha \Delta w_{\mathcal{I}}$
  $\bar{\beta} \leftarrow \bar{\beta} - \alpha$
  $c \leftarrow c - \alpha d$

  // Update active set or policy:
  if $\alpha = \alpha_1$ **then**
    $\mathcal{I} \leftarrow \mathcal{I} \bigcup \{i_1\}$
  else if $\alpha = \alpha_2$ **then**
    $\mathcal{I} \leftarrow \mathcal{I} - \{i_2\}$
  else if $\alpha = \alpha_3$ **then**
    $\pi(i_3) \leftarrow \pi'_3$
  end if
end while
return w

of features in the active set. If we consider an update of our solution, $w_{\mathcal{I}} = w_{\mathcal{I}} + \alpha\Delta w_{\mathcal{I}}$, setting the correlations of features in the active set equal to the correlations in the inactive set and solving for α yields the step sizes at which new features might join the active set. To maintain the invariant, however, we only consider the minimum such step. The correct step size and new feature are given by the expression

$$\alpha = \min_{j \notin \mathcal{I}}{}^{+} \left\{ \frac{c_j - \bar{\beta}}{d_j - 1}, \frac{c_j + \bar{\beta}}{d_j + 1} \right\}$$

where

$$d \equiv \Phi^T(\Phi_{\mathcal{I}} - \gamma\Phi'_{\mathcal{I}})\Delta w_{\mathcal{I}},$$

and $\min^{+}\{\cdot\}$ indicates a minimum over positive operands only.

Finally, to maintain invariant 3, we must also consider the case when moving along the update direction causes the coefficient of some active feature to change sign. If such a point is reached before a point at which we would add a feature, then the algorithm requires that the affected feature be dropped at the point at which its coefficient goes to zero. The step size at which such an event occurs can be found analytically as well:

$$\alpha = \min_{j \in \mathcal{I}}{}^{+,0} \left\{ -\frac{w_j}{\Delta w_j} \right\}.$$

Here, $\min^{+,0}\{\cdot\}$ indicates the minimum is taken over non-negative operands only.

At each step in the algorithm, then, LARS-TD moves as far as it can along the direction given by Δw, stopping at the earliest point at which one of three things occurs: 1) an inactive feature becomes active; 2) an active feature becomes inactive; or 3) we reach the target regularization coefficient (i.e., $\bar{\beta} = \beta$). Kolter and Ng prove that if $A = \Phi^T(\Phi - \gamma\Phi')$ is a P-matrix[1], then for any $\beta \geqslant 0$, LARS-TD will find a solution to equation 4.1. Johns et al. (2010) further demonstrate the uniqueness of

[1] A P-matrix is a square matrix for which every principle minor is positive.

the solution when A is a P-matrix. As Kolter and Ng note, this property is satisfied when our set of features is linearly independent and samples are taken on-policy, i.e., from the stationary distribution (A will then be positive definite). However, our usual goal is to use LARS-TD, like LSTD, inside a policy iteration algorithm such as LSPI, in which case off-policy sampling is the norm. Kolter and Ng propose adding some amount of L_2 regularization in this setting to ensure A is a P-matrix.

5.2 The LARQ Algorithm

We now introduce the LARQ algorithm, which extends LARS-TD by incorporating greedy policy improvements into the LARS-TD loop. LARQ solves the greedy L_1 regularized fixed point equation:

$$w = \arg\min_{u} \frac{1}{2}\|\Phi u - \max_{\pi}(R + \gamma\Phi'^{\pi} w)\|^2 + \beta\|u\|_1 \tag{5.1}$$

To do this, the LARQ algorithm must maintain the invariant constraints of the LARS-TD algorithm, and also a new invariant:

4. The current policy, π is greedy with respect to the current solution.

It turns out that we can change policies and avoid violating the LARS-TD constraints if we make policy changes at points where applying the Bellman operator yields the same value for both the old policy (π) and the new (π'):

$$T^{\pi}\hat{V} = T^{\pi'}\hat{V}. \tag{5.2}$$

The LARS-TD constraints all depend on the correlations of features with the residual $T\hat{V} - \hat{V}$ of the current solution. When the above equation is satisfied, the residual is equal for both policies. Thus we can change policies at this point without violating any of the LARS-TD constraints. The policy inflection points, much like the

feature addition/removal points of LARS-TD, can be found analytically; according to equation (5.2), we want to find α such that

$$\begin{aligned} R + \gamma \Phi'^{\pi}(w + \alpha \Delta w) &= R + \gamma \Phi'^{\pi'}(w + \alpha \Delta w) \\ \Phi'^{\pi} w + \alpha \Phi'^{\pi} \Delta w &= \Phi'^{\pi'} w + \alpha \Phi'^{\pi'} \Delta w \\ \alpha \Phi'^{\pi} \Delta w - \alpha \Phi'^{\pi'} \Delta w &= \Phi'^{\pi'} w - \Phi'^{\pi} w \\ \alpha &= -\frac{\Phi_i'^{\pi'} w - \Phi_i'^{\pi} w}{\Phi_i'^{\pi'} \Delta w - \Phi_i'^{\pi} \Delta w}. \end{aligned}$$

Therefore, we make policy changes at a step size given by the expression

$$\alpha = \min_{\pi', i} -\frac{\Phi_i'^{\pi'} w - \Phi_i'^{\pi} w}{\Phi_i'^{\pi'} \Delta w - \Phi_i'^{\pi} \Delta w}, \tag{5.3}$$

when this α is smaller than the distance to the next feature addition/removal, etc.

To satisfy the greedy constraint of LARQ, however, we must also ensure that the new policy we switch to will be better than the old policy as we move towards the new fixed point. In general, all that is necessary to satisfy this requirement is that the gradient of the new policy, $\Phi_i'^{\pi'} \Delta w$ with respect to the direction of travel is greater than the gradient of the old policy.

In rare cases there will not be a unique policy that is better. This will be manifest when the algorithm attempts to switch to a new policy immediately following a policy change. In this case, the algorithm can get stuck in a loop, switching back and forth between policies. We call this situation *policy deadlock*. We have implemented a simple strategy for breaking this condition, called *deadlock breaking*, by which a feature with high correlation with residual is rescaled to bring its correlation in line with the current active set. This can break deadlocks by increasing the expressiveness of the basis.

5.3 LARQ Theoretical Properties

The LARQ algorithm inherits many properties of the LARS-TD algorithm. When there is only one action possible at any state, these two algorithms are identical. The extent to which the theoretical properties of LARS-TD can generalize to LARQ depends upon the effects of policy changes.

Theorem 3. *If all* $\Phi^T(\Phi - \gamma\Phi'^{\pi})$ *are P-matrices, and LARQ never encounters a policy deadlock condition, then for any* $\beta \geqslant 0$*, LARQ finds a solution to the* L_1 *regularized fixed point condition of Equation 5.1.*

Proof. We must establish two things: (1) that LARQ never violates the LARS-TD invariants and (2) that LARQ will terminate with the greedy policy chosen by the T^* operator.

If the policy does not change, then LARQ is simply evaluating a fixed policy and the results of Kolter and Ng Kolter and Ng (2009) apply, ensuring that the LARS-TD invariants will not be violated. To establish (1), we must therefore show that policy changes cannot violate the LARS-TD invariants. Specifically, we must ensure that the correlations of the features, in both the active and inactive sets, with the residual do not change. LARQ ensures this by changing policies only at policy inflection points. Equation 5.2 ensures that changing policies cannot change the residual, thereby ensuring that the correlations cannot change upon changing policy.

To establish (2), we show that LARQ *always* follows a greedy policy by a simple, inductive argument. LARQ is initialized with a greedy policy. Policy inflection points identify the point where a new policy will become greedy. By construction, LARQ always chooses the closest policy inflection point, never missing an opportunity to switch to the greedy policy. The assumption that there are no policy

deadlocks ensures that the greedy policy is unique at each inflection point. Under these assumptions, LARQ cannot deviate from the greedy policy. □

It follows that LARQ can change policies only finitely many times under similar assumptions:

Theorem 4. *If all* $\Phi^T(\Phi - \gamma\Phi'^{\pi})$ *are P-matrices, and LARQ never encounters a policy deadlock condition, then for any* $\beta \geqslant 0$*, LARQ changes policies a finite number of times.*

Proof. First, we note that the previous theorem does not immediately guarantee this, since convergence could, in principle, be asymptotic and could involve infinitely many policy changes. We note, however, that for any policy, any active set, and any vector of correlation signs s with $s_i \in \{-1, +1\}$, the weights chosen by LARQ must satisfy:

$$\Phi_{\mathcal{I}}^T(R + \gamma\Phi_{\mathcal{I}}'^{\pi}w - \Phi_{\mathcal{I}}w) = \beta s$$

to ensure that the active set has constant absolute correlation with the residual. For any set of correlation signs, this constrains w to lie on a line with position parameterized by β. Policy inflection points are defined by the intersection of lines (equation 5.2) with this trajectory. Since lines can intersect at most once, there are finitely many lines, and β decreases monotonically, there can be only finitely many policy changes. □

As with LARS-TD, these results can be strengthened if Φ is restricted to the set of features that ever appear in the active set. We conjecture that similar theorems to these exist for the case where deadlock breaking is used. A subtlety in this case is that the feature space is changing during the LARQ run, so any guarantees must be with respect to the scaled features and not the original ones.

5.4 Experimental Results

We present two types of results in this section. The first use a known model and are designed to show performance on a known, feature selection benchmark with different regularization parameters. The second emphasize a more challenging, continuous problem and emphasize performance on the task rather than value function fit. In all of our experiments, we ran LARQ with deadlock breaking.

5.4.1 50-state Chain

The 50-state chain from Lagoudakis and Parr (2003) is a simple, discrete problem that is used to test reinforcement learning algorithms and feature selection techniques. The states are numbered from 1 to 50 and arranged in a linear chain, with two actions (right or left) that attempt to increase or decrease the state index, succeeding with probability 0.9, and resulting in movement in the opposite direction with probability 0.1. There are rewards at states 10 and 41, and the discount factor is typically 0.9. The optimal value function looks like a suspension bridge with peaks at these states.

We ran a model-based version LARQ on this problem using the true transition function, and a set of 192 radial basis functions of varying widths. The point of this evaluation is to see LARQ's ability to construct good, sparse value functions from a haphazard set of features. Algorithms that more carefully construct features (Mahadevan and Maggioni, 2007; Parr et al., 2007) typically generate about 15 basis functions per action (about 30 total) for this problem. Figure 5.1 shows the value functions produced by LARQ for varying L_1 coefficients. In general, LARQ pieces together a good, sparse value function from a haphazardly chosen set of features. Achieving a near-perfect value function requires about $2\times$ as many features as methods that construct features, but the correct policy and overall value function shape

are preserved with far fewer features.

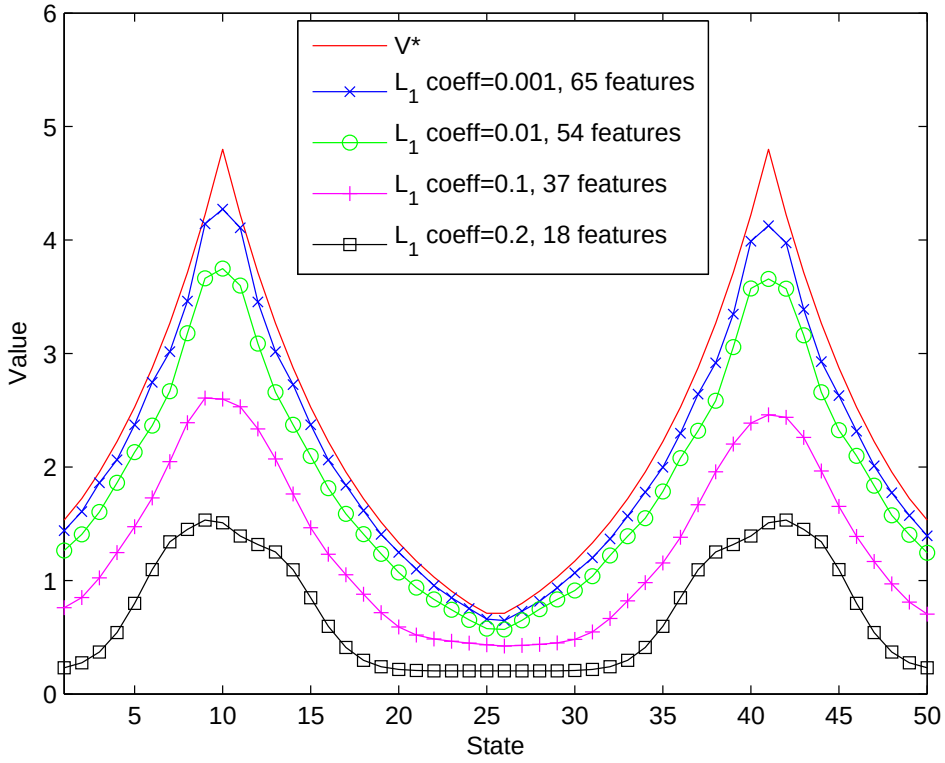


FIGURE 5.1: Value functions produced by LARQ on the 50-state chain for varying regularization coefficients. The feature count is the sum over both actions.

We also ran a model based version of the algorithm that computes the value function directly, rather than computing Q-functions. This is possible when the model is available, though it is not possible when only samples are available since there is no way of choosing actions. In this experiment (not shown in figure) the value function with 22 features was comparable to LARQ's result with 64 total features, and the value function with 15 features was comparable to LARQ's result with 54.

5.4.2 Inverted Pendulum

We ran LARQ in the inverted pendulum simulator from Lagoudakis and Parr (2003). We also implemented an LSPI-like wrapper around LARS-TD, following the implementation of Kolter and Ng.

This version of the inverted pendulum problem is a two-dimension continuous control task with three discrete actions – impulses in the two possible directions and holding still. We gave the algorithms a set of 99 basis functions, arising from a 3×3 gridding of the state space with RBFs of width 1, a 5×5 gridding of the state space with RBFs of width 0.8, and 8×8 gridding of the state space with RBFs of width 0.5.

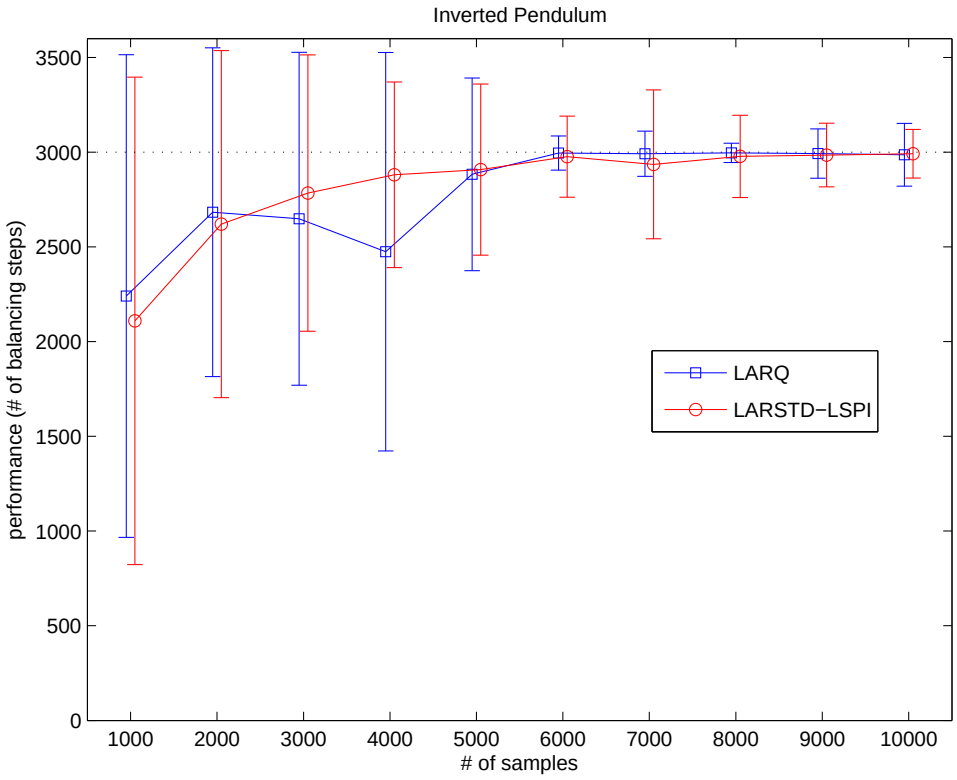


FIGURE 5.2: Performance on the inverted pendulum task for LARQ and Policy Iteration with LARS-TD as a function of the number of training samples

Figure 5.2 shows the balancing performance (up to a max of 3000 steps per trial) vs. the number training samples collected. Training samples were collected by starting the simulator in a slightly perturbed, balanced state and taking randomly selected actions until the pole falls and simulation is restarted. For both algorithms, the regularization parameter was 0.1. Runs typically used around 40 features in total. Our results show good performance for both algorithms.

We also must mention one important detail about our experiments: The discount factor for this problem is typically set to 0.95, but we used 0.9. The reason is that LARQ tended to become unstable for larger discount factors. Specifically, the algorithm would tend to add new features that would lead to ill-conditioned matrix inversion problems. This issue underscores a challenge in achieving a robust LARQ implementation: With a large number of states and large number of features, catching policy changes and feature changes at precisely the right point while preserving LARS-TD invariants and numerical stability of the solution can be non-trivial. Lowering the discount factor tends to make everything better conditioned, but such tactics are not appropriate in all situations.

5.5 Related Work

Kolter and Ng (2009) use a policy iteration iteration wrapper around LARS-TD to perform policy optimization. LARS-TD is a simpler algorithm that is likely to be more robust in implementation than LARQ. This same approach could be taken with any algorithm which finds the L_1-regularized fixed point, such as L1TD or LC-TD (Johns et al., 2010). Even so, one possible limitation is that it may be difficult to provide any kind of convergence guarantee since approximate policy iteration can oscillate (Bertsekas and Tsitsiklis, 1996), a phenomenon we observed in our experiments.

A better alternative may be the LC-MPI algorithm of Johns et al. (2010). The

Algorithm 4 LC-MPI

Input: $\{s_i, a_i, r_i, s'_i\}, \gamma, \varphi, \beta_1, \beta_2, \ldots, \beta_m$, with $\beta_i > \beta_{i+1}, \forall i < m$
Output: $\{w^i, \beta_i\}_{i=1}^m$
for $i = 1$ to $m - 1$ **do**
  // Move to next L_1 parameter using LARS-TD
  $\hat{w}^{i+1} \leftarrow$ run LARS-TD starting from (w^i, β_i) to β_{i+1} for policy $greedy(w^i)$
  // Policy iteration loop at next L_1 parameter using LC-TD with warm starts
  while not converged **do**
    $w^{i+1} \leftarrow$ run LC-TD at β_{i+1} for policy $greedy(\hat{w}^{i+1})$ with warm start $\hat{w}^{i+1}$
    if $w^{i+1} \approx \hat{w}^{i+1}$ or iterations exceeded **then**
      converged $\leftarrow$ true
    else
      $\hat{w}^{i+1} \leftarrow w^{i+1}$
    end if
  end while
end for

algorithm LC-MPI (algorithm 4) is a compromise between LARQ and LARS-TD with policy iteration. In LC-MPI, rather than continually checking for policy changes, as in LARQ, the algorithm updates the policy only at a fixed set of regularization coefficients β. At each β, the algorithm uses policy iteration with LC-TD to update the current policy and compute an approximate value function. Like LARQ, and unlike LARS-TD with policy iteration, LC-MPI keeps the previously selected features as a starting point; this gives LC-MPI an advantage because LC-TD (unlike LARS-TD) can perform a "warm start", in which the previous solution is used as a starting point to speed up convergence. (A similar approach would be possible with L1TD in batch mode instead of LC-TD, but we have not done a performance comparison of the two.) The LC-MPI algorithm is described as approximating LARQ's homotopy path, since the two algorithms agree for any β reachable by LARQ.

5.6 Discussion

We have presented a modification to the LARS-TD algorithm that incorporates policy changes within the LARS-TD weight update loop. These changes permit simultaneous optimization and feature selection in a single, unified algorithm called LARQ.

LARQ enjoys many of the theoretical properties of LARS-TD but adds policy optimization. The main strengths of LARQ are in its convergence guarantees (conditional on the avoidance of policy deadlocks) and in its ability to preserve useful features across policy changes. The main limitations of LARQ, in its current form, are the issue of policy deadlocks, and in the challenges posed in achieving a stable LARQ implementation for high discount factors. Although LARS-like algorithms are, in general, somewhat "fussy" numerically, this issue is aggravated with LARQ because the number of intersections that must be checked grows to include policy inflection points.

Our experimental results demonstrate that LARQ can find good policies and sparse solutions when presented with large, even haphazardly selected feature sets – promising developments towards the goal of easing the burden of feature engineering for reinforcement learning. Directions for further development include more robust implementations that guard against numerical instabilities, and strengthening of the theoretical guarantees.

6

Greedy Algorithms for Sparse Reinforcement Learning

So far we have been discussing work which applies L_1 regularization techniques adapted from the supervised learning literature for use with RL. Another approach that has received renewed attention in the supervised learning community is that of using a simple algorithm that greedily adds new features (Tropp, 2004; Zhang, 2009). Such algorithms have many of the good properties of the L_1 regularization methods, while also being extremely efficient and, in some cases, allowing theoretical guarantees on recovery of the true form of a sparse target function from sampled data despite the myopia associated with greediness.

The most basic greedy algorithm for feature selection for regression, matching pursuit, uses the correlation between the residual and the candidate features to decide which feature to add next (Mallat and Zhang, 1993). In this chapter we consider a variation called orthogonal matching pursuit (OMP), which recomputes the residual after each new feature is added, as applied to reinforcement learning. It is related to Bellman error basis functions (Parr et al., 2007), but it differs in that it

Algorithm 5 OMP

Input: $X \in \mathbb{R}^{n \times k}, y \in \mathbb{R}^n, \beta \in \mathbb{R}$.
Output: Approximation weights w.
$\mathcal{I} \leftarrow \{\}$
$w \leftarrow 0$
repeat
  $c \leftarrow |X^T(y - Xw)|$
  $j \leftarrow \arg\max_{i \notin \mathcal{I}} c_i$
  if $c_j > \beta$ **then**
    $\mathcal{I} \leftarrow \mathcal{I} \cup \{j\}$
  end if
  $w_{\mathcal{I}} \leftarrow X_{\mathcal{I}}^{\dagger} y$
until $c_j \leqslant \beta$ **or** $\bar{\mathcal{I}} = \{\}$

selects features from a finite dictionary. OMP for RL was explored by Johns (2010) in the context of proto-value functions (Mahadevan and Maggioni, 2007) and diffusion wavelets (Mahadevan and Maggioni, 2006), but aside from this initial exploration of the topic, we are not aware of any efforts to bring the theoretical and empirical understanding of OMP for reinforcement learning to parity with the understanding of OMP as applied to regression.

This chapter contributes to the theoretical and practical understanding of OMP for RL. Variants of OMP are analyzed and compared experimentally with existing L_1 regularized approaches. We demonstrate that perhaps the most natural scenario in which one might hope to achieve sparse recovery fails; however, one variant, OMP-BRM, provides promising theoretical guarantees under certain assumptions on the feature dictionary. Another variant, OMP-TD, lacks theoretical guarantees but empirically outperforms OMP-BRM and prior methods both in approximation accuracy and efficiency on several benchmark problems.

6.1 Prior Art

6.1.1 OMP for Regression

Algorithm 5 is the classic OMP algorithm for regression. It is greedy in that it myopically chooses the feature with the highest correlation with the residual and

never discards features. We say that target y is *m-sparse in* X if there exists an X_{opt} composed of m columns of X and corresponding w_{opt} such that $y = X_{opt}w_{opt}$, and X_{opt} is minimal in the sense there is no X' composed of fewer columns of X which can satisfy $y = X'w$. Of the many results for sparse recovery, Tropp's 2004 is perhaps the most straightforward:

Theorem 5. *If y is m-sparse in X, and*

$$\max_{i \notin opt} \|X_{opt}^{\dagger} x_i\|_1 < 1,$$

then OMP called with X, y, and $\beta = 0$ will return w_{opt} in m iterations.

In the maximization, the vectors x_i correspond to the columns of X which are not needed to reconstruct y exactly. In words, this condition requires that the projection of any suboptimal feature into the span of the optimal features must have small weights. Note that any orthogonal basis trivially satisfies this condition.

Tropp also extended this result to the cases where y is approximately sparse in X, and Zhang (2009) extended the result to the noisy case.

6.1.2 L_1 Regularization in RL

In counterpoint to the greedy methods based on OMP is the work on feature selection in RL based on least-squares methods with L_1 regularization, which we have explored in previous chapters. For regression, we recall equation (2.9), the "soft-constraint" or unconstrained version of the Lasso (Tibshirani, 1996):

$$\|y - Xw\|^2 + \beta\|w\|_1. \tag{6.1}$$

Previous chapters examined methods which apply L_1 regularization to the linear fixed point, resulting in the L_1-regularized linear fixed point (equation 4.1). An alternative approach, considered by Loth et al. (2007), is to apply the Lasso to

Bellman residual minimization (section 2.4.1). Replacing the residual $y - Xw$ in equation 6.1 with the Bellman residual, we obtain

$$\|R + \gamma\Phi' w - \Phi w\|^2 + \beta\|w\|_1.$$

Trivially, if we let $X = \Phi - \gamma\Phi'$ and $y = R$, we can substitute directly into equation 6.1 and solve as a regression problem. The regression algorithm used by Loth et al. is very similar to LARS (Efron et al., 2004), thus we refer to this approach as "LARS-BRM".

It is instructive to note that LARS bears some resemblance to OMP in that it selects as new features those with the highest correlation to the residual of its current approximation. However, where OMP is purely greedy and seeks to use all active features to the fullest, LARS is more moderate and attempts to use all active features *equally*, in the sense that all active features maintain an equal correlation with the residual. One aspect of the LARS approach that sets it quite apart from OMP is that LARS will remove features from the active set when necessary to maintain its invariants.

Intuitively, LARS and algorithms based on LARS such as LARS-TD have an advantage in minimizing the number of active features due to their ability to remove features. LC-TD also adds and removes features. These methods do suffer from some disadvantages related to this ability, however. LARS-TD can be slowed down by the repeated adding and removing of features. Worse, both LARS-TD and LC-TD involve computations which are numerically sensitive and are not guaranteed to find the desired solution in all cases since (unlike in the pure regression case) the task of finding an L_1 regularized linear fixed point is not a convex optimization problem.

6.2 OMP for RL

Algorithm 6 OMP-BRM

Input:
$\Phi \in \mathbb{R}^{n \times k} : \Phi_{ij} = \varphi_j(s_i)$,
$\Phi' \in \mathbb{R}^{n \times k} : \Phi'_{ij} = \varphi_j(s'_i)$,
$R \in \mathbb{R}^n : R_i = r_i$,
$\gamma \in [0, 1)$,
$\beta \in \mathbb{R}$
Output:
Approximation weights w.

call OMP with $X = \Phi - \gamma\Phi'$, $y = R$, $\beta = \beta$

Algorithm 7 OMP-TD

Input:
$\Phi \in \mathbb{R}^{n \times k} : \Phi_{ij} = \varphi_j(s_i)$,
$\Phi' \in \mathbb{R}^{n \times k} : \Phi'_{ij} = \varphi_j(s'_i)$,
$R \in \mathbb{R}^n : R_i = r_i$,
$\gamma \in [0, 1)$,
$\beta \in \mathbb{R}$
Output:
Approximation weights w.

$\mathcal{I} \leftarrow \{\}$
$w \leftarrow 0$
repeat
  $c \leftarrow |\Phi^T(R + \gamma\Phi' w - \Phi w)|/n$
  $j \leftarrow \arg\max_{i \notin \mathcal{I}} c_i$
  if $c_j > \beta$ **then**
    $\mathcal{I} \leftarrow \mathcal{I} \cup \{j\}$
  end if
  $w_{\mathcal{I}} \leftarrow (\Phi_{\mathcal{I}}^T \Phi_{\mathcal{I}} - \gamma\Phi_{\mathcal{I}}^T \Phi'_{\mathcal{I}})^{-1} \Phi_{\mathcal{I}}^T R$
until $c_j \leqslant \beta$ **or** $\bar{\mathcal{I}} = \{\}$

We present two algorithms for policy evaluation: OMP-BRM and OMP-TD.[1] As the names suggest, the first algorithm is based on Bellman residual minimization (BRM), while the second is based on the linear TD fixed point. Algorithm 6, OMP-BRM, is the simpler algorithm in the sense that it essentially performs OMP with features $\Phi - \gamma\Phi'$ using the reward vector as the target value. OMP-BRM is different from the OMP-BR algorithm introduced by Johns (2010), which selected basis functions from Φ.

[1] Note that both algorithms reduce to OMP when $\gamma = 0$.

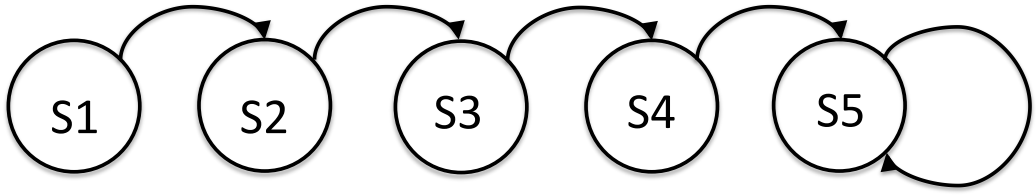


FIGURE 6.1: A Markov chain for which OMP-TD cannot achieve sparse recovery with an indicator function basis.

Algorithm 7, OMP-TD, applies the basic OMP approach to build a feature set for LSTD.[2] OMP-TD is similar in approach to the approximate BEBF algorithm of Parr et al. (2007), in which each new feature is an approximation to the current Bellman residual. In OMP-TD, rather than approximate the Bellman residual, we simply add the feature which, among features not already in use, currently has the highest correlation with the residual. After adding a feature, the new fixed point is computed using the closed form LSTD fixed point equation, and the new residual is computed. The main results of the BEBF paper apply to OMP-TD: mainly, that each new feature improves a bound on the distance between the fixed point and the true value function V^*, as long as the correlation between the feature and the residual is sufficiently large.

6.2.1 Sparse Recovery in OMP-TD

Theorem 6. *Even if V^* is m-sparse in an orthonormal basis, OMP-TD cannot guarantee exact recovery of V^* in m iterations.*

Proof. (By counterexample) Consider the Markov chain in figure 6.1. The arcs indicate deterministic transitions. Suppose $R(S2) = R(S3) = R(S4) = 1$, $R(S5) = 0$,

[2] Johns (2010) called the same algorithm OMP-FP.

and $R(S1) = -(\gamma + \gamma^2 + \gamma^3)$, then $V^* = [0, 1 + \gamma + \gamma^2, 1 + \gamma, 1, 0]$. With an orthonormal basis Φ defined by the indicator functions $\varphi_i(s) = I(s = s_i)$, V^* is 3-sparse in Φ, with $opt = \{2, 3, 4\}$. Starting from the empty set of features, the residual vector is just R. OMP-TD will pick the vector with the greatest correlation with the residual, which will be φ_1, a vector that is not in opt. □

The next feature added by OMP-TD could be $S4$, then $S3$ and $S2$. Selecting a single vector not in *opt* suffices to establish the proof, which means that we could have shortened the example by removing states $S2$ and $S3$ and connecting $S1$ directly to $S4$. However, the longer chain is useful to illustrate an important point about OMP-TD: It is possible to add a gratuitous basis function at the very first step of the algorithm and the mistake may not be evident until an arbitrary number of additional basis functions are added. This example is easily extended so that an arbitrary number of gratuitous basis functions are added before the first basis function in *opt* is added by making multiple copies of the $S1$ state (together with the corresponding indicator function features). Such constructions can defeat modifications to OMP-TD that use a window of features and discard gratuitous ones (Jain et al., 2011) for any fixed-size window.

An algorithm that chooses features from Φ based upon the Bellman residual (OMP-BR in the terminology of Johns (2010)), would suffer the same difficulties as OMP-TD in this example. The central problem is that *the Bellman error may not be a trustworthy guide* for selecting features from Φ even if Φ is orthogonal.

6.2.2 Sparse Recovery in OMP-BRM

The theoretical behavior of OMP-BRM provides an interesting counterpoint to that of OMP-TD. We begin with a simple lemma:

Lemma 7. *If V^* is m-sparse in Φ, then R is at least m-sparse in $(\Phi - \gamma P\Phi)$.*

Proof. Since $V^* = (I - \gamma P)^{-1} R$, we have

$$\begin{aligned} (I - \gamma P)^{-1} R &= \Phi_{opt} w_{opt} \\ R &= \Phi_{opt} w_{opt} - \gamma P \Phi_{opt} w_{opt} \\ &= \left[\Phi - \gamma P \Phi\right]_{opt} w_{opt}. \end{aligned}$$

□

The implication is that we can perform OMP on the basis $(\Phi - \gamma P \Phi)$ and, if there is a sparse representation for R in the basis, we will obtain a sparse representation of V^* as well. This permits a sparse recovery claim for OMP-BRM that is in stark contrast to the negative results for OMP-TD.

Theorem 8. *If V^* is m-sparse in Φ, and*

$$\max_{i \notin opt} \|X_{opt}^{\dagger} x_i\|_1 < 1, \tag{6.2}$$

for $X = \Phi - \gamma P \Phi$, then OMP-BRM called with Φ, $\Phi' = P\Phi$, R, γ, and $\beta = 0$ will return w such that $V^* = \Phi w$ in at most m iterations.

Proof. (sketch) The proof mirrors a similar proof from Tropp (2004) and is provided in detail in section 6.3. Lemma 7 implies that R is m-sparse in X. Since OMP-BRM does OMP with basis X and target R, the sparse recovery results for OMP for regression apply directly. □

Tropp's extension to the approximate recovery case also applies directly to OMP-BRM (see section 6.3). For the noisy case, we expect that the results of Zhang (2009) could be generalized to RL, but we defer that extension for future work.

6.2.3 Sparse Recovery Behavior

We set up experiments to validate the theory of exact recovery for OMP-BRM, and to investigate the behavior of OMP-TD in similar circumstances. First, we generated

a basis for the 50-state chain problem (see section 6.4) in which the first three basis functions provide an exact reconstruction of V^*, and the remaining 997 features are randomly generated, which satisfies equation (6.2). (Generating such a basis required first generating a much larger (50 x 3000) matrix, then throwing out features which violated the exact recovery condition, and finally trimming the matrix back down to 1000 features. The first three features were constructed by finding two random features which highly correlate with V^*, then adding in a third feature which was the reconstruction residual using the first two features.)

By using the resulting basis in OMP-BRM with exact data (i.e., where $\Phi' = P\Phi$), we found that, for sufficiently small threshold value, OMP-BRM uses exactly the first three features in its approximation, in accordance with theory. We also tried OMP-BRM with noisy data by sampling 200 state transitions from the 50-state chain problem. In this case, we found that OMP-BRM reliably selected the first three features before selecting any other features. Interestingly, the same basis proved to enable exact recovery for OMP-TD as well. Using the same 200 samples, OMP-TD selected the first three features before selecting any other features.

6.3 Proofs of Theorems

We proceed first with a proof of theorem 8, which we restate here:

Theorem 8. *If V^* is m-sparse in Φ, and*

$$\max_{i \notin opt} \|X_{opt}^{\dagger} x_i\|_1 < 1,$$

for $X = \Phi - \gamma P\Phi$, then OMP-BRM will return w such that $V^* = \Phi w$ in at most m iterations.

The proof below mirrors a similar proof from Tropp (2004).

Proof. Suppose that after $q < |opt|$ steps OMP-BRM has computed a solution $\hat{V} = \Phi_q w_q$ which is a linear combination of q features from the optimal set of features Φ_{opt}. The algorithm will in the next step add the feature j for which the correlation $X_j^T(R - X_q w_q)$ is largest in magnitude.

We wish for j to come from the set *opt*. Note that for j already added, the correlation with the residual is zero, thus we can safely consider the largest correlation from the set *opt*, which we can write as $\|X_{opt}^T(R - X_q w_q)\|_\infty$. To ensure that this feature is the one selected, we require that it have a higher correlation than any feature not in *opt*. The largest correlation of a feature not in *opt* is $\|X_{\overline{opt}}^T(R - X_q w_q)\|_\infty$, where $\overline{opt}$ is the set of features not in *opt*. Our requirement is then equivalent to the requirement

$$\frac{\|X_{\overline{opt}}^T(R - X_q w_q)\|_\infty}{\|X_{opt}^T(R - X_q w_q)\|_\infty} < 1. \tag{6.3}$$

We note that the residual $(R - X_q w_q)$ is necessarily in the span of X_{opt} as, by assumption, V^* is in the span of Φ_{opt} and thus by theorem 7, R is in the span of X_{opt}, and $X_q w_q$ is in the span of X_{opt} because we have only chosen features from *opt* so far. Therefore we can project the residual into the span of X_{opt} in the numerator without changing the expression:

$$\begin{aligned} & \frac{\|X_{\overline{opt}}^T(R - X_q w_q)\|_\infty}{\|X_{opt}^T(R - X_q w_q)\|_\infty} \\ = \; & \frac{\|X_{\overline{opt}}^T X_{opt}(X_{opt}^T X_{opt})^{-1} X_{opt}^T(R - X_q w_q)\|_\infty}{\|X_{opt}^T(R - X_q w_q)\|_\infty} \\ \leqslant \; & \|X_{\overline{opt}}^T X_{opt}(X_{opt}^T X_{opt})^{-1}\|_\infty, \end{aligned}$$

where in the last step we have used the definition of the matrix norm induced by the max norm:

$$\|A\|_\infty = \max_{x \neq 0} \frac{\|Ax\|_\infty}{\|x\|_\infty}.$$

As Tropp notes, $\|\cdot\|_\infty$ is equal to the maximum absolute row sum of the matrix, whereas $\|\cdot\|_1$ is equal to the maximum absolute column sum; thus we have

$$\begin{aligned} & \|X_{\overline{opt}}^T X_{opt}(X_{opt}^T X_{opt})^{-1}\|_\infty \\ = \; & \|(X_{\overline{opt}}^T X_{opt}(X_{opt}^T X_{opt})^{-1})^T\|_1 \\ = \; & \|(X_{opt}^T X_{opt})^{-1} X_{opt}^T X_{\overline{opt}})\|_1 \\ = \; & \max_{i \notin opt} \|X_{opt}^\dagger x_i\|_1. \end{aligned}$$

The proof is then complete by noting that requiring this last expression be less than 1 is sufficient to ensure that the next feature selected is from *opt*. □

6.3.1 Extension to Approximately Sparse Case

For general targets y and basis X we do not expect y to be *m-sparse*. Instead, we are interested in the best *m-sparse* approximation to y, that is, we wish to find $\hat{y}_{opt}$ such that

$$\hat{y}_{opt} = \underset{\hat{y}:\hat{y} \; is \; \text{m-sparse} \; in \; X}{\arg\min} \|\hat{y} - y\|.$$

Obviously there are $\begin{pmatrix} k \\ m \end{pmatrix}$ possible size m subsets of X for $X \in \mathbb{R}^{n \times k}$, so we do not simply want to find the least-squares approximation for every size m subset and take the one with minimum error. However, Tropp (Tropp, 2004) shows that we can still find a *nearly* optimal approximation to y under certain assumptions on X. As noted previously, these results can be applied to OMP-BRM in a straightforward fashion.

Following Tropp, we define the *cumulative coherence function* as

$$\mu_m(X) \equiv \max_{\Lambda:|\Lambda|=m} \max_{i \notin \Lambda} \|X_\Lambda^T x_i\|_1. \tag{6.4}$$

Next we give the following lemma concerning the optimal *m-sparse* BRM solution $\hat{V}_{opt}$:

Lemma 9. *If $X_{opt}w_{opt}$ is the best* m-sparse *approximation to R in $X = (\Phi - \gamma P\Phi)$, then $\hat{V}_{opt} = \Phi_{opt}w_{opt}$ is the* m-sparse *solution with the smallest Bellman residual.*

Proof. By definition,

$$w_{opt} = \arg\min_{w_\Lambda:|\Lambda|=m} \|X_\Lambda w_\Lambda - R\| \tag{6.5}$$

$$= \arg\min_{w_\Lambda:|\Lambda|=m} \|(\Phi_\Lambda - \gamma P\Phi_\Lambda)w_\Lambda - R\| \tag{6.6}$$

$$= \arg\min_{w_\Lambda:|\Lambda|=m} \|\Phi_\Lambda w_\Lambda - (R + \gamma P\Phi_\Lambda w_\Lambda)\|. \tag{6.7}$$

□

The following theorem and corollaries are direct extensions from Tropp (2004) (starting with Theorem 4.2) to OMP-BRM.

Theorem 10. *Assume that $\mu_m(X) < \frac{1}{2}$, with $X = \Phi - \gamma P\Phi$. Further assume that the columns of X are normalized. Suppose that after $q < m$ steps OMP-BRM has computed a solution $\hat{V} = \Phi_q w_q$ which is a linear combination of q features from the optimal set of features Φ_{opt}. Then at step $q+1$, OMP-BRM will add another feature from the set opt provided that*

$$\|R - X_q w_q\| > \sqrt{1 + \frac{m(1-\mu_m(X))}{(1-2\mu_m(X))^2}}\|R - X_{opt}w_{opt}\|. \tag{6.8}$$

The meaning of this theorem is that OMP-BRM recovers another optimal feature whenever the current approximation is sufficiently "bad" compared to the optimal solution.

Corollary 11. *Assume that $\mu_m(X) < \frac{1}{2}$, with $X = \Phi - \gamma P\Phi$. Then, for arbitrary R, OMP-BRM produces an m-term solution w_m that satisfies*

$$\|R - X_m w_m\| \leqslant \sqrt{1+C}\|R - X_{opt}w_{opt}\|, \tag{6.9}$$

where

$$C = \frac{m(1 - \mu_m(X))}{(1 - 2\mu_m(X))^2}.$$

Proof. Assume that at step $q + 1$ equation (6.8) is violated. Then by inspection, $\|R - X_q w_q\| \leqslant \sqrt{1 + C}\|R - X_{opt} w_{opt}\|$, and any further addition of features by OMP-BRM can only reduce the Bellman residual. □

If we take the expression for C and plug in an assumption that $\mu_m(X) \leqslant \frac{1}{3}$, we get a simpler bound on the m-term error:

Corollary 12. *Assume that $\mu_m(X) \leqslant \frac{1}{3}$. Then OMP-BRM produces an m-term solution w_m that satisfies*

$$\|R - X_m w_m\| \leqslant \sqrt{1 + 6m}\|R - X_{opt} w_{opt}\|. \tag{6.10}$$

6.3.2 Proof of Theorem 10

We will also need Lemma 2.3 from Tropp (2004), which we reproduce here:

Lemma 13. *The squared singular values of X_{opt} exceed $(1 - \mu_{m-1}(X))$.*

Proof. Consider the Gram matrix $G = (X_{opt})^T X_{opt}$. Let the columns of X_{opt} be numbered as $opt_1, opt_2, \ldots, opt_m$. The Gershgorin Disc Theorem states that every eigenvalue of G lies in one of the m discs

$$\Delta_k = \left\{ z : |G_{kk} - z| \leqslant \sum_{j \neq k} |G_{jk}| \right\}.$$

The normalization of the columns of X implies that $G_{kk} = 1$. The sum is bounded above by

$$\sum_{j \neq k} |G_{jk}| = \sum_{j \neq k} (x_{opt_k})^T x_{opt_j} \leqslant \mu_{m-1}(X).$$

Then any eigenvalue λ of G satisfies $|1-\lambda| \leqslant \mu_{m-1}(X) \Rightarrow \lambda \geqslant 1-\mu_{m-1}(X)$. The proof is completed by noting that the eigenvalues of G are the squared singular values of X_{opt}. □

Proof of theorem 10:

Proof. Suppose that after $q < m$ steps OMP-BRM has computed a solution $\hat{V} = \Phi_q w_q$ which is a linear combination of q features from the optimal set of features Φ_{opt}. As noted in the proof of theorem 8, the algorithm will in the next step add the feature j for which the correlation $X_j^T(R - X_q w_q)$ is largest in magnitude. The condition for adding another optimal feature is

$$\frac{\|X_{\overline{opt}}^T(R - X_q w_q)\|_\infty}{\|X_{opt}^T(R - X_q w_q)\|_\infty} < 1.$$

We now expand the ratio into a sum of two expressions which we will bound separately:

$$\begin{aligned}
& \frac{\|X_{\overline{opt}}^T(R - X_q w_q)\|_\infty}{\|X_{opt}^T(R - X_q w_q)\|_\infty} \\
= \; & \frac{\|X_{\overline{opt}}^T(R - X_{opt} w_{opt}) + X_{\overline{opt}}^T(X_{opt} w_{opt} - X_q w_q)\|_\infty}{\|X_{opt}^T(R - X_{opt} w_{opt}) + X_{opt}^T(X_{opt} w_{opt} - X_q w_q)\|_\infty} \\
= \; & \frac{\|X_{\overline{opt}}^T(R - X_{opt} w_{opt}) + X_{\overline{opt}}^T(X_{opt} w_{opt} - X_q w_q)\|_\infty}{\|X_{opt}^T(X_{opt} w_{opt} - X_q w_q)\|_\infty} \\
\leqslant \; & \frac{\|X_{\overline{opt}}^T(R - X_{opt} w_{opt})\|_\infty}{\|X_{opt}^T(X_{opt} w_{opt} - X_q w_q)\|_\infty} \\
& + \frac{\|X_{\overline{opt}}^T(X_{opt} w_{opt} - X_q w_q)\|_\infty}{\|X_{opt}^T(X_{opt} w_{opt} - X_q w_q)\|_\infty}.
\end{aligned} \tag{6.11}$$

The penultimate line follows from the fact that the expression $R - X_{opt} w_{opt}$ is orthogonal to the span of X_{opt}.

Starting with the right hand summand of (6.11), we note that we can project the expression $X_{opt}w_{opt} - X_q w_q$ into the span of X_{opt}, and as before, we obtain

$$\begin{aligned}
& \frac{\|X_{\overline{opt}}^T(X_{opt}w_{opt} - X_q w_q)\|_\infty}{\|X_{opt}^T(X_{opt}w_{opt} - X_q w_q)\|_\infty} \\
= & \frac{\|X_{\overline{opt}}^T X_{opt}(X_{opt}^T X_{opt})^{-1} X_{opt}^T(X_{opt}w_{opt} - X_q w_q)\|_\infty}{\|X_{opt}^T(X_{opt}w_{opt} - X_q w_q)\|_\infty} \\
\leqslant & \|X_{\overline{opt}}^T X_{opt}(X_{opt}^T X_{opt})^{-1}\|_\infty \\
= & \max_{i \notin opt} \|X_{opt}^\dagger x_i\|_1.
\end{aligned}$$

Now we expand the pseudo-inverse

$$\max_{i \notin opt} \|X_{opt}^\dagger x_i\|_1 = \max_{i \notin opt} \|(X_{opt}^T X_{opt})^{-1} X_{opt}^T x_i\|_1,$$

and bound the resulting expression by

$$\begin{aligned}
& \max_{i \notin opt} \|(X_{opt}^T X_{opt})^{-1} X_{opt}^T x_i\|_1 \\
\leqslant & \|(X_{opt}^T X_{opt})^{-1}\|_1 \max_{i \notin opt} \|X_{opt}^T x_i\|_1. \qquad (6.12)
\end{aligned}$$

Applying the definition of the cumulative coherence function (6.4), we can bound the second factor of (6.12) as

$$\max_{i \notin opt} \|X_{opt}^T x_i\|_1 \leqslant \mu_m(X).$$

Bounding the first factor of (6.12) in Tropp's words "requires more sophistication." Since by assumption the columns of X are normalized, the diagonal of $X_{opt}^T X_{opt}$ is all ones. We therefore rewrite $X_{opt}^T X_{opt} = I + A$, where I is the $m \times m$ identity matrix. A then contains only off-diagonal elements. Each column of A contains the dot products between one column of X_{opt} and the other $m-1$ columns. Then we have $\|A\|_1 = \max_i \|X_{opt\setminus i}^T x_i\|_1 \leqslant \mu_{m-1}(X)$.

Since by assumption we have $\mu_m(X) \leqslant \frac{1}{2}$ we have $\|A\|_1 \leqslant \mu_{m-1}(X) < 1$. We can use the Neumann series to write $(I+A)^{-1} = \sum_{i=0}^{\infty}(-A)^i$. Then,

$$\begin{aligned}
\|(X_{opt}^T X_{opt})^{-1}\|_1 &= \|(I+A)^{-1}\|_1 \\
&= \left\|\sum_{i=0}^{\infty}(-A)^i\right\|_1 \\
&\leqslant \sum_{i=0}^{\infty}\|A\|_1^i \\
&= \frac{1}{1-\|A\|_1} \\
&\leqslant \frac{1}{1-\mu_{m-1}(X)}.
\end{aligned}$$

Recombining the two bounds of equation 6.12, we have

$$\begin{aligned}
\max_{i \notin opt}\|X_{opt}^{\dagger}x_i\|_1 &\leqslant \frac{\mu_m(X)}{1-\mu_{m-1}(X)} \\
&\leqslant \frac{\mu_m(X)}{1-\mu_m(X)}.
\end{aligned}$$

We now return to bounding the left hand summand of (6.11). We have

$$\begin{aligned}
& \frac{\|X_{\overline{opt}}^T(R - X_{opt}w_{opt})\|_\infty}{\|X_{opt}^T(X_{opt}w_{opt} - X_q w_q)\|_\infty} \\
= & \frac{\max_{i \notin opt} |x_i^T(R - X_{opt}w_{opt})|}{\|X_{opt}^T(X_{opt}w_{opt} - X_q w_q)\|_\infty} \\
\leqslant & \frac{\max_{i \notin opt} \|x_i\| \|R - X_{opt}w_{opt}\|}{\|X_{opt}^T(X_{opt}w_{opt} - X_q w_q)\|_\infty} \\
= & \frac{1\|R - X_{opt}w_{opt}\|}{\|X_{opt}^T(X_{opt}w_{opt} - X_q w_q)\|_\infty} \\
\leqslant & \frac{\|R - X_{opt}w_{opt}\|}{(1/\sqrt{(m)})\|X_{opt}^T(X_{opt}w_{opt} - X_q w_q)\|} \\
\leqslant & \frac{\sqrt{m}\|R - X_{opt}w_{opt}\|}{\sigma_{min}(X_{opt})\|X_{opt}w_{opt} - X_q w_q\|},
\end{aligned}$$

where $\sigma_{min}(A)$ denotes the smallest singular value of A. In the penultimate step we used the fact that $\sqrt{m}\|z\|_\infty \geqslant \|z\|$ for vector z of length.[3] In the final step, we used the fact that $\|Az\| \geqslant \sigma_{min}(A)\|z\|$.[4]

Next we replace $\sigma_{min}(X_{opt})$ with $\sqrt{1 - \mu_m(X)}$ per Lemma 13, and recombine our bounds on the components of equation 6.11 to determine that we can recover another optimal feature whenever

$$\frac{\sqrt{m}\|R - X_{opt}w_{opt}\|}{\sqrt{1 - \mu_m(X)}\|X_{opt}w_{opt} - X_q w_q\|} + \frac{\mu_m(X)}{1 - \mu_m(X)} < 1.$$

Isolating the term $\|X_{opt}w_{opt} - X_q w_q\|$, we obtain

$$\|X_{opt}w_{opt} - X_q w_q\| > \frac{\sqrt{m(1 - \mu_m(X))}}{1 - 2\mu_m(X)}\|R - X_{opt}w_{opt}\|.$$

[3] Because $m\|z\|_\infty^2 = m \max_i |z_i|^2 \geqslant \sum_{i=1}^m |z_i|^2 = \|z\|_2^2$.

[4] This follows from the fact that $\sigma_{min}(A) = \min_{\|y\|=1} \|Ay\|$ (Meyer, 2000). For arbitrary z with $z = cy, c = \|z\|, \|y\| = 1$, we have $\|Az\| = \|Acy\| = c\|Ay\| \geqslant c\sigma_{min}(A) = \sigma_{min}(A)\|z\|$.

Now we note that $(R - X_{opt}w_{opt})$ is orthogonal to the span of X, while $(X_{opt}w_{opt} - X_q w_q)$ lies in the span of X. Then the Pythagorean theorem gives us

$$\begin{aligned}
\|R - X_q w_q\|^2 &= \|R - X_{opt}w_{opt}\|^2 + \|X_{opt}w_{opt} - X_q w_q\|^2 \\
&> \left(1 + \frac{m(1-\mu_m(X))}{(1-2\mu_m(X))^2}\right) \|R - X_{opt}w_{opt}\|^2 \\
\|R - X_q w_q\| &> \sqrt{1 + \frac{m(1-\mu_m(X))}{(1-2\mu_m(X))^2}} \|R - X_{opt}w_{opt}\|,
\end{aligned}$$

which is the condition stated in theorem 10. □

6.4 Experiments

While we have theoretical results for OMP-BRM that suggest we should be able to perform optimal recovery under certain conditions on the feature dictionary, practical problems contain noise, which current theory does not address. In addition, the desired conditions on the feature dictionary may not hold and it can be difficult to verify if they do hold since these conditions are a property of both the features and the transition function for OMP-BRM. For OMP-TD, we have negative worst-case results, but Section 6.2.3 gives hope that things may not be so dismal in practice. To gain some understanding of how these algorithms perform under typical conditions, we performed experiments on a number of benchmark RL problems. Figure 6.2 shows our main results, the approximation accuracy achieved by each of four algorithms on each problem. Table 6.1 summarizes the benchmark problem properties and some experimental settings. The algorithms studied included:

OMP-BRM: The OMP-BRM algorithm as described above, but with some additional machinery to improve performance in actual use. It is well known (Sutton and Barto, 1998) that BRM is biased in the presence of noise, i.e., when samples are taken from a stochastic transition function. One solution to this problem is to use double samples for each transition. In each of our experiments, we ran with

and without doubled samples, and we report the behavior of the better performing option. Table 6.1 records which experiments use doubled samples. Figure 6.3 shows how doubling samples affects performance on the 50-state chain problem. In all cases, the number of samples in the *Samples* column refers to the number of starting states.[5] In the doubled case we also add in a small amount (0.01) of L_2 regularization in order to keep the algorithm well behaved (by keeping the matrices to be inverted well conditioned).

OMP-TD: The OMP-TD algorithm as described above, but with a small amount (0.01) of L_2 regularization when computing the final solution at each threshold. This change seemed to be important for some of the harder benchmarks such as puddle-world and two-room, where without regularization, the LSTD solution at various threshold values would occasionally exhibit unstable behavior. We use regularization on all problems, as it seems to cause no issues even for benchmarks which do not need it.

LARS-BRM: This is our implementation of the algorithm of Loth et al. (2007), which effectively treats BRM as a regression problem to be solved using LARS. Note that there is no provision for sample doubling in this algorithm, which may affect its performance on some problems.

LARS-TD: This is the LARS-TD algorithm of Kolter and Ng (2009). Again we found it useful sometimes to include a small amount (0.01) of L_2 regularization, giving the "elastic net" (Zou and Hastie, 2005) solution. The additional regularization does not always benefit; however, we report in figure 6.2 the better of the two for each benchmark. Table 6.1 records which experiments use L_2 regularization in LARS-TD.

We compared performance of the various algorithms on the following benchmark

[5] This could be interpreted as giving the double-sample case an unfair advantage because there is no "penalty" for the additional samples collected. On the other hand, counting next states only could be seen as imposing too harsh of a penalty on OMP-BRM since the same number of samples would necessarily have sparser coverage of the state space.

problems:

Chain: We use the 50-state chain problem described by Lagoudakis and Parr (2003), evaluating the optimal policy for the problem. We use a discount factor of 0.8. The features used in this problem are a set of radial basis functions (RBFs) with centers located at the grid points of various discretizations of the range $[0, 51]$. The RBF centers are not necessarily located at problem states. The width of the RBFs vary depending on the grid spacing. A constant term is also included.

To ensure good coverage of the problem states, training samples are taken by first sampling initial states uniformly at random, then sampling next states according to the transition probabilities for the policy. Approximation error is evaluated with respect to the true value function with uniform weighting of the 50 states.

Pendulum: We use the inverted pendulum domain, in which the task is to balance an inverted pendulum by applying forces to the cart to which it is attached. There are three (noisy) actions, applying force to the left or right or no force. The agent receives a -1 penalty if the pendulum falls over. We evaluate this task using a simple, suboptimal policy that nevertheless balances the pendulum fairly well. We use a discount factor of 0.95. For this task we construct a feature set by placing RBFs centered on grid points for grids of various sizes. The RBF widths vary depending on the grid spacing. A constant term is also included.

Training samples are obtained from multiple on-policy trajectories in which the initial state is a vector randomly perturbed around the point at which the pendulum is balanced and unmoving, following the policy thereafter until the pendulum falls over. A set of 10,000 test samples was obtained in the same fashion. A ground truth estimate of the true value function at each sample was obtained by Monte Carlo sampling, and approximation error is measured with respect to the test samples.

Blackjack: We use a version of the bottomless-deck blackjack problem from Sutton and Barto (1998), evaluating the policy in which the player stands on 20 or 21

and hits on anything else. In this problem, there are 200 states arising from the cross product of three variables: player total ($[11 \ldots 21]$), dealer showing card value ($[A \ldots 10]$), and the presence of a usable ace in the players hand (true/false). In our version, there are three additional states for win/lose/draw, and the player continues playing infinitely; after reaching a win/lose/draw state, the next state is drawn from the distribution of possible starting deals. With this modification, the problem is mixing with a well-defined stationary distribution. The player receives a reward of $+1$ for a win and -1 for a loss; all other rewards are zero. A discount factor of 0.95 keeps expected player losses bounded.

We provide the learner with 219 features: a bias term, an indicator for each of {win, lose}, indicator functions for all possible contiguous ranges of the players current total times an indicator for whether the player has a usable ace, and likewise indicator functions on the dealer's show card times the ace indicator. Despite having more features than states, the rank of the feature matrix (the dimension of the basis) is only 41.

To ensure good coverage of the problem states, training samples were taken by first sampling initial states uniformly at random, then sampling next states according to the transition probabilities for the policy. Approximation error is evaluated with respect to the true value function with uniform weighting of the 203 states.

Mountain Car: We test the familiar mountain car domain, in which an underpowered car must back up a small hill to gain enough potential energy to climb a larger one. The agent receives a -1 penalty for every step it takes until it reaches the goal region at the top of the hill. We chose to evaluate this task using a simple policy in which the car always accelerates in whichever direction it is already moving, or forward if it is at rest; while suboptimal, this policy ensures that the car will reach the goal region from any valid starting state. We use a discount factor of 0.99. Our features for this task are composed of 1365 radial basis functions of various widths

arranged in differently spaced grids over the state space together with a constant (bias) term.

As movement in the mountain car domain is deterministic, we sample trajectories by first selecting a starting state uniformly at random, and then following policy to the goal (on average, about 50 steps). A set of 10,000 test samples was obtained in the same fashion. Approximation error is measured with respect to the true value at each test state.

Puddleworld: We report results on the Puddle World problem from Boyan and Moore (1994). Puddle World is a two-dimensional navigation problem that requires an agent to move to a corner goal state while avoiding puddles, which are regions of high cost. We evaluate this task using a simple policy in which the agent always tries to take the shortest path to the goal region, ignoring puddles. Our discount factor is 0.99. Our features for this task are composed of two sets of RBFs arranged in two differently spaced grids over the state space, with each set having an appropriately chosen RBF width. A constant term is also included.

To ensure good coverage of the problem states, training samples are taken by first sampling initial states uniformly at random, then simulating policy for one step. The test states used are the points of a 101×101 grid covering the state space. A ground truth estimate of the true value function at each state was obtained by Monte Carlo sampling, and approximation error is measured with respect to the test states.

Two Room: Here we consider an adaptation of the two room maze problem of Mahadevan and Maggioni (2007), evaluating the policy by Taylor and Parr (2009). We use a discount factor of 0.9. Our features for this task are composed of two sets of RBFs arranged in two differently spaced grids over the state space, with each set having an appropriately chosen RBF width. Indicators on {goal, not goal} are included in the features.

To ensure good coverage of the problem states, training samples are taken by first

Table 6.1: Benchmark experiment properties and experimental settings.

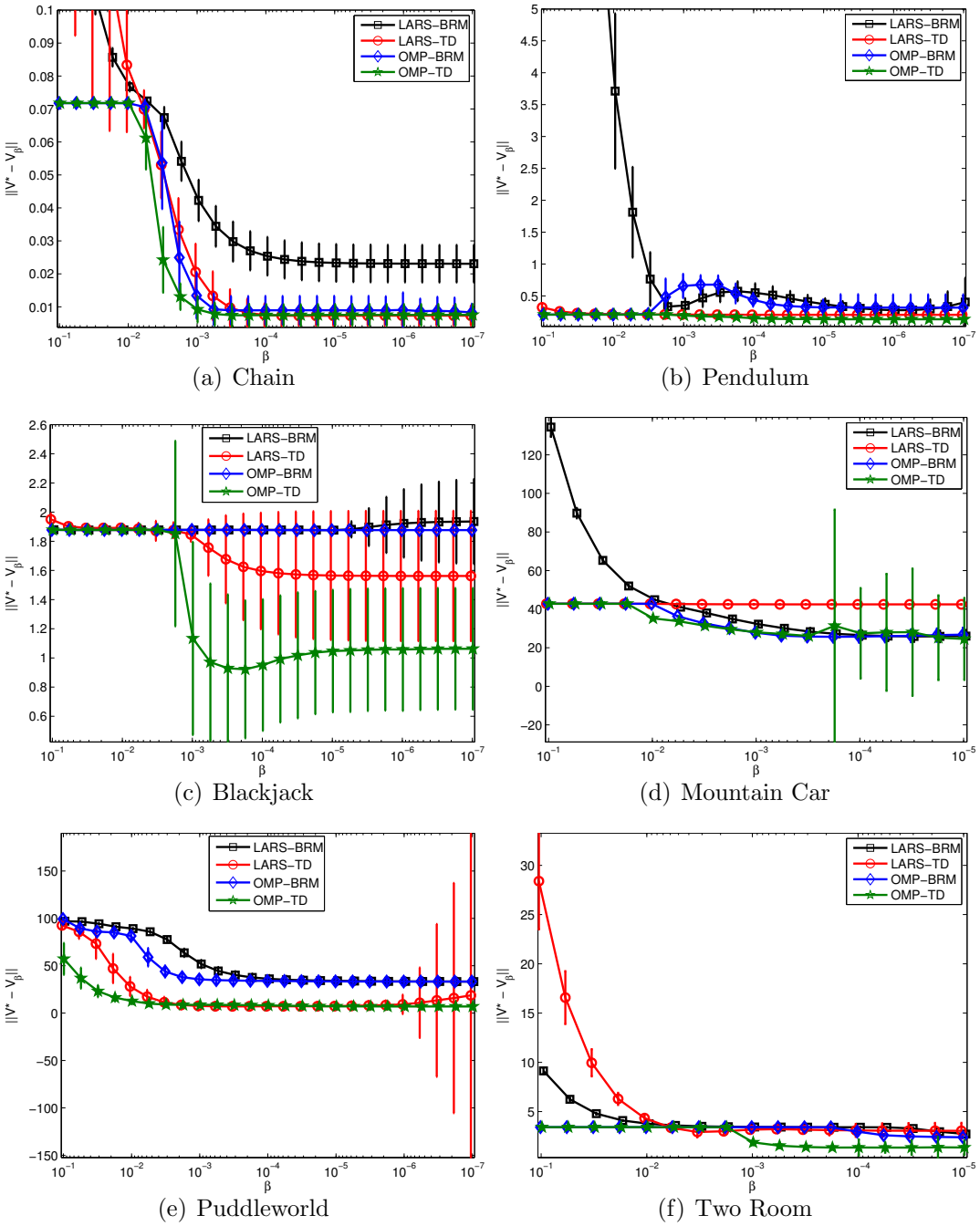
Problem	State space	Features	Samples	Trials	LARS-TD L_2?	BRM double samples?
Chain	Discrete, 50 states	208	500	1000	×	√
Pendulum	Continuous, 2d	268	200	1000	√	√
Blackjack	Discrete, 203 states	219	1600	1000	×	×
Mountain Car	Continuous, 2d	1366	5000	100	√	×
Puddleworld	Continuous, 2d	570	2000	500	×	×
Two Room	Continuous, 2d	2227	5000	1000	×	×

sampling initial states uniformly at random, then simulating policy for one step. The test states used are the points of a 101×101 grid covering the state space. A ground truth estimate of the true value function at each state was obtained by Monte Carlo sampling, and approximation error is measured with respect to the test states.

6.4.1 Results

In figure 6.2, we have plotted performance of the four algorithms on the six benchmark problems. The vertical axis is the root mean square error with respect to the true value function, V^*. For the discrete state problems, V^* is computed exactly, while for the continuous state problems V^* is computed at a large number of sampled states by Monte Carlo rollouts. The number of trials over which the values are averaged is reported in table 6.1. The horizontal axis gives the threshold/regularization coefficient value β for the value function V_β plotted.

In general, the meaning of β for the OMP algorithms is different than for the LARS algorithms. However, there is a strong similarity between the two in that, for both, a solution at value β implies that there are no further features with a correlation with the current residual of β or larger. This is explicit in the OMP algorithms, and implicit in the fixed point conditions for LARS-TD (n.b. Kolter and Ng (2009), equation 9) and LARS-BRM. Surprisingly, given that OMP is greedy and never removes features, we find that the sparsity of the solutions given by the



(a) Chain (b) Pendulum

(c) Blackjack (d) Mountain Car

(e) Puddleworld (f) Two Room

FIGURE 6.2: Error with respect to V^* versus regularization/threshold coefficient on several benchmark problems.

algorithms is similar for the same values of β; e.g., see figure 6.4.

As figure 6.2 shows, OMP-TD is generally competitive with, or better than,

LARS-BRM and LARS-TD on most of the benchmark problems. In addition, we note that the OMP-based algorithms are considerably faster than the LARS-based algorithms; see figure 6.5 for a comparison of computation time on the puddleworld problem.

While OMP-TD generally leads in our benchmarks, we should point out some caveats. With very small numbers of samples (even fewer than shown in our experiments) OMP-TD was somewhat more prone to unstable behavior than the other algorithms. This could simply mean that OMP-TD requires more L_2 regularization, but we did not explore that in our experiments. An indication that OMP-TD can require more L_2 regularization than the other algorithms is evident in our OMP-TD experiments for Mountain Car, where the high variance for low values of β arises from just two (out of 100) batches of samples. In these cases, it appears that OMP-TD is adding a feature which is essentially a delta function on a single sample. Without additional L_2 regularization, OMP-TD produces very poor value functions for these batches.

For all of the experiments except for blackjack, the features are radial basis functions (RBFs). There are multiple widths of RBFs placed in the state space in grids of various spacing. For blackjack, the basis functions are indicators on groups of states. All features are normalized, although we tried both with and without normalization, and the results were qualitatively similar.

6.5 Discussion

In this chapter we have explored the theoretical and practical applications of OMP to RL. We analyzed variants of OMP and compared them experimentally with existing L_1 regularized approaches. We showed that perhaps the most natural scenario in which one might hope to achieve sparse recovery fails; however, one variant, OMP-BRM, provides promising theoretical guarantees under certain assumptions on the

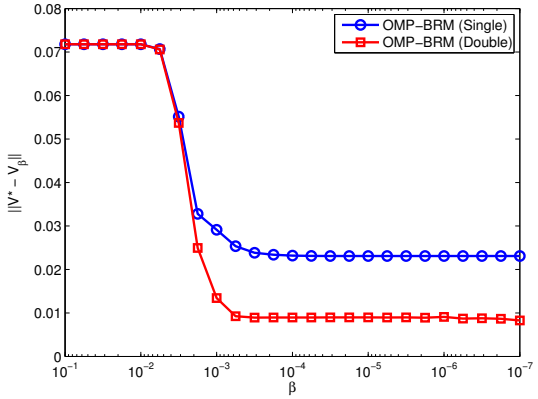


FIGURE 6.3: Comparison of error between OMP-BRM using single samples versus doubled samples on chain.

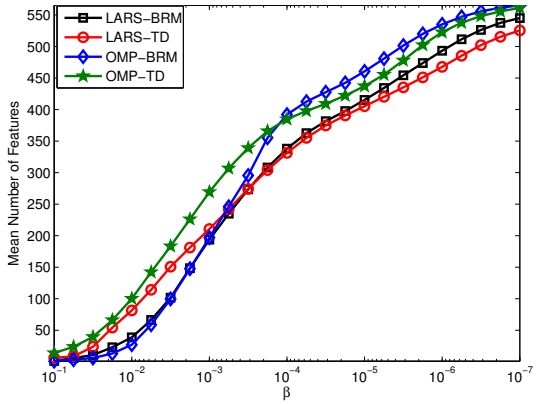


FIGURE 6.4: Sparsity (mean number of features) versus regularization/threshold coefficient on puddleworld.

feature dictionary. Another variant, OMP-TD, empirically outperforms prior methods both in approximation accuracy and efficiency on several benchmark problems.

There are two natural directions for further development of this work. Our theoretical results for OMP-BRM built upon the simplest results for sparse recovery in regression and do not apply directly to more realistic scenarios that involve noise. Stronger results may be possible, building upon the work of Zhang (2009). A more interesting, but also more challenging, future direction would be the theoretical development to explain the extremely strong performance of OMP-TD in practice despite

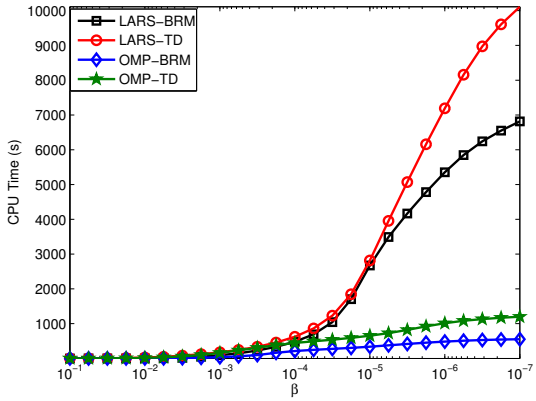


FIGURE 6.5: Execution time versus regularization/threshold coefficient on puddle-world.

negative theoretical results on sparse recovery. For example, there could be an alternate set of conditions on the features that frequently hold in practice and that can be shown theoretically to have good sparse recovery guarantees.

7

Summary and Conclusions

This dissertation describes work from a time period during which feature selection and regularization were major topics of discussion in the RL community. The problem of selecting features for RL is a longstanding one. Despite decades of research, binary features (e.g. tile coding, coarse coding) and radial basis functions are still go-to features for linear value function approximation; while these can be very effective, they do not scale well in general. Some of the more notable efforts to break free of this state of affairs include kernelized methods (Taylor and Parr, 2009) and methods which automate feature generation based on the topology of the problem space (Mahadevan and Maggioni, 2007) or the Bellman error (Parr et al., 2007).

With the introduction of highly expressive feature spaces, such as with kernelization, the topic of regularization in RL came to the fore (Farahmand et al., 2008; Taylor and Parr, 2009). Meanwhile, the sparsity-inducing L_1 regularization techniques current in the supervised learning community (Efron et al., 2004; Kim et al., 2007) began to be applied by the signal processing community to reconstruct signals with small amounts of data (Candès and Wakin, 2008). The combination of these two trends culminated in the work by Kolter and Ng (2009), who introduced the L_1

regularized linear fixed point which has been a topic throughout this dissertation. Along with L_1 regularization, the supervised learning and signal processing communities have recently returned to methods which greedily add relevant features one at a time (Tropp, 2004), which produce results comparable to those of L_1 regularized methods, but which are simpler in description and implementation. Johns et al. (2010) first applied these to RL, a study which we continue here.

The benefits of applying sparsity-inducing techniques to RL are many. First, these techniques are excellent at preventing overfitting, allowing the use of highly expressive feature spaces. Second, these techniques naturally select sparse feature sets from large dictionaries of features, easing the feature selection problem greatly. Finally, sparse feature selection distinguishes relevant features from irrelevant ones, aiding in interpretation of the model and identifying important parts of the problem.

Our major contributions have regarded the use of sparsity inducing methods in effecting feature selection for reinforcement learning. First, we created the first online algorithm for finding the L_1 regularized linear fixed point. Second, we extended the L_1 regularized linear fixed point to define a *greedy* L_1 regularized linear fixed point, also extending the policy evaluation algorithm of Kolter and Ng to perform greedy policy optimization within the homotopy loop. Finally, we examine the application of greedy selection methods to linear value function approximation and compare the resulting methods with L_1 regularized methods. These results are summarized below.

7.1 L_1 Regularized Linear Temporal Difference Learning

We presented the algorithm L1TD, an online algorithm for L_1 regularized TD learning. We show that the fixed point of L1TD is the L_1 regularized linear fixed point shared by LARS-TD (Kolter and Ng, 2009) and LC-TD (Johns et al., 2010). We define a closely related, somewhat more "conservative" algorithm, L1TDAlt, which shares the same fixed point as L1TD, and show that L1TDAlt converges under the

usual conditions for convergence of linear TD.

While convergence may, in theory, be local, we demonstrate experimental convergence toward the fixed point on two benchmark problems. Furthermore we validate the ability of L1TD and L1TDAlt to produce sparse solutions. On a third benchmark problem we show that small amounts of regularization result in a modest improvement in solution quality over unregularized linear TD, despite having a wealth of data.

This contribution is significant in two particulars. First, it is the first online algorithm for L_1 regularized TD learning (easily modified to perform L_2 regularization, if desired), and lays the groundwork for additional efforts in this direction. Second, the algorithm uses an efficient iterative update procedure which may be preferred over direct (least-squares) approaches in some applications. If it is the case that features are themselves sparse, i.e., only a few features are non-zero for any given state, and if the non-zero features can be efficiently computed, then the online update can be extremely efficient, involving only the non-zero features for the state being updated and the features which are currently active in the algorithm (the ones with non-zero weights). This same property has been leveraged in a supervised learning context by Langford et al. (2008), and could see application on certain RL problems such as computer Go (Silver et al., 2012).

7.2 Simultaneous Feature Selection and Optimization via Least Angle Regression

We extended the L_1 regularized linear fixed point and defined the greedy L_1 regularized linear fixed point; in the greedy fixed point, the current policy is maintained so as to be greedy with respect to the current solution. We show that maintaining the greedy property is possible within the homotopy loop of LARS-TD because the point at which one policy needs to make way for another policy is exactly the point

at which the two policies agree on the Bellman residual; this agreement makes it possible to maintain all of the LARS-TD invariants while also choosing the greedy policy. We provide the algorithm LARQ which extends LARS-TD to maintain the greedy invariant and thus yield simultaneous feature selection and optimization.

Unfortunately it is not possible to guarantee that LARQ will run to completion for a given problem, due to the issue of policy deadlock conditions in which the algorithm wishes to switch to a new policy (or even the previous old policy) immediately upon switching policies. While we do provide a simple mechanism for breaking the deadlock, it complicates our analysis of the algorithm. Theoretically, then, we show that if there are no deadlocks and if the conditions under which LARS-TD converges apply, then LARQ will converge to the greedy fixed point in a finite number of steps.

Experimentally, we demonstrate that LARQ is similar to LARS-TD with policy iteration in terms of solution quality. LARQ unfortunately inherits the numerical stability issues of LARS-TD, and then amplifies them by taking many more steps due to making policy changes in addition to changes in the feature set. Nevertheless, LARQ was an important inspiration for the modified policy iteration algorithm LC-MPI (Johns et al., 2010), which approximates the LARQ solution for a fixed set of regularization coefficients.

7.3 Greedy Algorithms for Sparse Reinforcement Learning

We consider applications of orthogonal matching pursuit (OMP) (Pati et al., 1993) to reinforcement learning, and compare them with corresponding applications of L_1 regularization. An analysis of OMP by Tropp (2004) shows that this greedy forward selection scheme can provide some of the same guarantees as L_1 regularized regression in regards to the exact reconstruction of a signal from samples. At the same time, OMP is a much simpler algorithm than LARS or other implementations of the Lasso, making it a desirable technique to apply to reinforcement learning.

Johns (2010) previously implemented the algorithm which we call OMP-TD as a scheme for selecting features from a dictionary of automatically generated features. To the best of our knowledge, the algorithm OMP-BRM is original to us. OMP-TD uses a greedy forward selection scheme within a loop that computes the linear fixed point at each step. OMP-BRM, on the other hand, applies OMP to Bellman residual minimization treated as a simple regression problem, although we make provision for double sampling to avoid bias in the BRM solutions.

The corresponding L_1 regularized methods are LARS-TD (Kolter and Ng, 2009) and what we call LARS-BRM (Loth et al., 2007). Like OMP-TD, LARS-TD adds features and computes residuals based on the linear fixed point, but using a LARS-like homotopy path instead of greedy forward selection. Like OMP-BRM, LARS-BRM treats Bellman residual minimization as a regression problem, to which it applies the Lasso (via LARS).

Theoretically we show that OMP-BRM, unsurprisingly, inherits the reconstruction guarantees of OMP, albeit on a different basis set ($X = \Phi - \gamma\Phi'$ rather than Φ). We show that OMP-TD can fail to recover the minimal sparse representation of a target even using an orthogonal basis, which is the best case for OMP. Intriguingly, in a simple experiment on a standard benchmark problem, OMP-TD performs as well as OMP-BRM in an exact recovery task.

Our main contribution regarding these four algorithms is a careful comparison of their performance on six benchmark problems to demonstrate the strengths and weaknesses of each algorithm. In general we found that the BRM algorithms did not perform as well as the TD algorithms, which is a somewhat expected result based on past literature comparing these two approaches. More significantly, however, we found that OMP-TD generally performed as well as or better than both BRM algorithms and LARS-TD. This is a highly useful result for two reasons; for one, OMP-TD is a far simpler and more robust algorithm than LARS-TD, and two,

OMP-TD is significantly faster than LARS-TD for small values of the regularization coefficient.

7.4 Conclusion

The work in this dissertation contributes to the efforts of the RL community to find effective and efficient feature selection methods for linear value function approximation. Sparsity inducing methods, including L_1 regularization and greedy methods, are highly effective in identifying relevant features from large dictionaries of features. Both approaches have found application in value function approximation.

Despite these advances, feature selection is still an area where more work needs to be done. One area that is easily identified is a continuing need for adequate dictionaries of features from which to select. Very large dictionaries of generic features (e.g., as a result of kernelization) are appealing, but do not scale well. For large problems, the task of identifying candidate features arguably still requires domain expertise.

However, identifying features that will be relevant over a wide range of policies (expecting that our real goal will require some form of policy improvement) is difficult even for an expert. This suggests a kind of semi-supervised interaction between the feature designer and the feature selection algorithms; the feature designer can expand a dictionary based on domain knowledge and observations of the current policy (noting, for instance, where the policy seems to lack information with which to make proper decisions is a good way of discovering new features), while the feature selection algorithms keep the growth of the function representation in check.

The development of new automated feature generation techniques may be the most helpful advance in future efforts to scale RL to real-world-size problems. With effective feature selection approaches available, the focus of feature generation becomes that of creating dictionaries of features which generalize well across policies,

with perhaps the ability to interact with and be informed by feature selection methods. It is unclear whether we can ever take people out of the feature generation loop entirely, but every advance in that direction opens up new horizons for reinforcement learning.

Bibliography

Baird, L. (1995), "Residual Algorithms: Reinforcement Learning with Function Approximation," in *In Proceedings of the Twelfth International Conference on Machine Learning*.

Bertsekas, D. P. and Tsitsiklis, J. N. (1996), *Neuro-Dynamic Programming*, Belmont, Mass: Athena Scientific.

Borkar, V. and Meyn, S. (2000), "The ODE method for convergence of stochastic approximation and reinforcement learning," *SIAM Journal on Control and Optimization*, 38, 447–469.

Borkar, V. S. (2008), *Stochastic Approximation: A Dynamical Systems Viewpoint*, New York: Cambridge University Press.

Boyan, J. A. (2002), "Technical Update: Least-Squares Temporal Difference Learning," *Machine Learning*, 49, 233–246.

Boyan, J. A. and Moore, A. W. (1994), "Generalization in Reinforcement Learning: Safely Approximating the Value Function," in *NIPS*, eds. G. Tesauro, D. S. Touretzky, and T. K. Leen, pp. 369–376, MIT Press.

Bradtke, S. J. and Barto, A. G. (1996), "Linear Least-Squares Algorithms for Temporal Difference Learning," *Machine Learning*, 22, 33–57.

Candès, E. J. and Wakin, M. B. (2008), "An Introduction To Compressive Sampling," *IEEE Signal Processing Magazine*, 21.

Carbonetto, P., Schmidt, M., and de Freitas, N. (2008), "An interior-point stochastic approximation method and an L1-regularized delta rule," in *NIPS*, eds. D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, pp. 233–240, MIT Press.

Chen, S. S., Donoho, D. L., and Saunders, M. A. (2001), "Atomic Decomposition by Basis Pursuit," *SIAM Review*, 43, 129–159.

Daubechies, I., Defrise, M., and Mol, C. D. (2004), "An iterative thresholding algorithm for linear inverse problems with a sparsity constraint," *Communications on Pure and Applied Mathematics*, 57, 1413–1457.

de Farias, D. P. and Van Roy, B. (2003), "The Linear Programming Approach to Approximate Dynamic Programming," *Operations Research*, 51, 850–865.

Donoho, D. L. and Johnstone, J. M. (1994), "Ideal spatial adaptation by wavelet shrinkage," *Biometrika*, 81, 425–455.

Duchi, J. and Singer, Y. (2009), "Efficient Online and Batch Learning using Forward Backward Splitting," *Journal of Machine Learning Research*, 10, 2899–2934.

Efron, B., Hastie, T., Johnstone, I., and Tibshirani, R. (2004), "Least Angle Regression," *The Annals of Statistics*, 32, 407–451.

Engel, Y., Mannor, S., and Meir, R. (2003), "Bayes Meets Bellman: The Gaussian Process Approach to Temporal Difference Learning," in *ICML*, eds. T. Fawcett and N. Mishra, pp. 154–161, AAAI Press.

Ernst, D., Geurts, P., and Wehenkel, L. (2005), "Tree-Based Batch Mode Reinforcement Learning," *Journal of Machine Learning Research*, 6, 503–556.

Farahmand, A. M., Ghavamzadeh, M., Szepesvári, C., and Mannor, S. (2008), "Regularized Policy Iteration," in *NIPS*, eds. D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, pp. 441–448, MIT Press.

Geramifard, A., Doshi, F., Redding, J., Roy, N., and How, J. P. (2011), "Online Discovery of Feature Dependencies," in *ICML*, pp. 881–888.

Ghavamzadeh, M., Lazaric, A., Munos, R., and Auer, P. (2011), "Finite-Sample Analysis of Lasso-TD," in *Proceedings of the Twenty-Eighth International Conference on Machine Learning (ICML-2011)*.

Gordon, G. J. (1995), "Stable Function Approximation in Dynamic Programming," Tech. Rep. CMU-CS-95-103, School of Computer Science, Carnegie Mellon University.

Gordon, G. J. (2001), "Reinforcement learning with function approximation converges to a region," in *Advances in neural information processing systems*.

Hale, E. T., Yin, W., and Zhang, Y. (2007), "A Fixed-Point Continuation Method for ℓ_1-Regularized Minimization with Applications to Compressed Sensing," Tech. Rep. CAAM TR07-07, Department of Computational and Applied Mathematics, Rice University, Houston, Texas.

Hale, E. T., Yin, W., and Zhang, Y. (2008), "Fixed-Point Continuation for ℓ_1-Minimization: Methodology and Convergence," *SIAM Journal on Optimization*, 19, 1107–1130.

Hoffman, M. W., Lazaric, A., Ghavamzadeh, M., and Munos, R. (2011), "Regularized Least Squares Temporal Difference learning with nested l2 and l1 penalization," in *European Workshop on Reinforcement Learning*.

Jain, P., Tewari, A., and Dhillon, I. (2011), "Orthogonal Matching Pursuit with Replacement," Tech. Rep. arXiv:1106.2774, Arxiv preprint.

Johns, J. (2010), "Basis Construction and Utilization for Markov Decision Processes using Graphs," Ph.D. thesis, University of Massachusetts Amherst.

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 *ICML*, eds. W. W. Cohen and A. Moore, vol. 148 of *ACM International Conference Proceeding Series*, pp. 449–456, ACM.

Kim, S., Koh, K., Lustig, M., Boyd, S., and Gorinevsky, D. (2007), "An interior-point method for large-scale l1-regularized least squares," *IEEE Journal on Selected Topics in Signal Processing*, 1, 606–617.

Kolter, J. Z. and Ng, A. Y. (2009), "Regularization and feature selection in least-squares temporal difference learning," in *ICML*, eds. A. P. Danyluk, L. Bottou, and M. L. Littman, vol. 382 of *ACM International Conference Proceeding Series*, p. 66, ACM.

Lagoudakis, M. G. and Parr, R. (2003), "Least-Squares Policy Iteration," *Journal of Machine Learning Research*, 4, 1107–1149.

Langford, J., Li, L., and Zhang, T. (2008), "Sparse Online Learning via Truncated Gradient," in *NIPS*, eds. D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, pp. 905–912, MIT Press.

Loth, M., Davy, M., and Preux, P. (2007), "Sparse temporal difference learning using LASSO," in *ADPRL 2007*.

Mahadevan, S. and Liu, B. (2012), "Sparse Q-learning with Mirror Descent," in *UAI*, eds. N. de Freitas and K. P. Murphy, pp. 564–573, AUAI Press.

Mahadevan, S. and Maggioni, M. (2006), "Value function approximation with diffusion wavelets and Laplacian eigenfunctions," *Advances in Neural Information Processing Systems*, 18, 843.

Mahadevan, S. and Maggioni, M. (2007), "Proto-value Functions: A Laplacian Framework for Learning Representation and Control in Markov Decision Processes," *Journal of Machine Learning Research*, 8, 2169–2231.

Mallat, S. G. and Zhang, Z. (1993), "Matching Pursuits With Time-Frequency Dictionaries," *IEEE Transactions on Signal Processing*, 41, 3397–3415.

Menache, I., Mannor, S., and Shimkin, N. (2005), "Basis Function Adaptation in Temporal Difference Reinforcement Learning," *Annals OR*, 134, 215–238.

Meyer, C. D. (2000), *Matrix Analysis and Applied Linear Algebra*, SIAM.

Nemirovski, A. and Yudin, D. (1983), *Problem Complexity and Method Efficiency in Optimization*, John Wiley Press.

Osborne, M. R., Presnell, B., and Turlach, B. A. (2000), "A new approach to variable selection in least-squares problems," *IMA Journal of Numerical Analysis*, 20, 389–403.

Painter-Wakefield, C. and Parr, R. (2012a), "Greedy Algorithms for Sparse Reinforcement Learning," in *ICML*, icml.cc / Omnipress.

Painter-Wakefield, C. and Parr, R. (2012b), "L_1 Regularized Linear Temporal Difference Learning," Tech. Rep. CS-2012-01, Duke University.

Parr, R., Painter-Wakefield, C., Li, L., and Littman, M. L. (2007), "Analyzing feature generation for value-function approximation," in *ICML*, ed. Z. Ghahramani, vol. 227 of *ACM International Conference Proceeding Series*, pp. 737–744, ACM.

Parr, R., Li, L., Taylor, G., Painter-Wakefield, C., and Littman, M. L. (2008), "An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning," in *ICML*, eds. W. W. Cohen, A. McCallum, and S. T. Roweis, vol. 307 of *ACM International Conference Proceeding Series*, pp. 752–759, ACM.

Pati, Y., Rezaiifar, R., and Krishnaprasad, P. (1993), "Orthogonal matching pursuit: recursive function approximation with applications to wavelet decomposition," in *Signals, Systems and Computers, 1993. 1993 Conference Record of The Twenty-Seventh Asilomar Conference on*, pp. 40–44 vol.1.

Petrik, M., Taylor, G., Parr, R., and Zilberstein, S. (2010), "Feature Selection Using Regularization in Approximate Linear Programs for Markov Decision Processes," in *ICML*, eds. J. Fürnkranz and T. Joachims, pp. 871–878, Omnipress.

Rasmussen, C. E. and Kuss, M. (2003), "Gaussian Processes in Reinforcement Learning," in *NIPS*, eds. S. Thrun, L. K. Saul, and B. Schölkopf, MIT Press.

Richardson, L. F. (1910), "The approximate arithmetical solution by finite differences of physical problems involving differential equations, with an application to the stresses in a masonry dam," *Philos. Trans. Roy. Soc. London Ser. A*, 210, 307–357.

Robbins, H. and Monro, S. (1951), "A Stochastic Approximation Method," *The Annals of Mathematical Statistics*, 22, 400.

Rummery, G. A. and Niranjan, M. (1994), "On-Line Q-Learning Using Connectionist Systems," Tech. Rep. CUED/F-INFENG/TR 166, Cambridge University Engineering Department.

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

Sutton, R. S. (1988), "Learning to Predict by the Methods of Temporal Differences," *Machine Learning*, 3, 9–44.

Sutton, R. S. and Barto, A. G. (1998), *Reinforcement Learning: An Introduction*, Cambridge, Mass.: MIT Press.

Taylor, G. and Parr, R. (2009), "Kernelized value function approximation for reinforcement learning," in *ICML*, eds. A. P. Danyluk, L. Bottou, and M. L. Littman, vol. 382 of *ACM International Conference Proceeding Series*, p. 128, ACM.

Tibshirani, R. (1996), "Regression Shrinkage and Selection via the Lasso," *Journal of the Royal Statistical Society. Series B (Methodological)*, 58, 267–288.

Tropp, J. A. (2004), "Greed is Good: Algorithmic Results for Sparse Approximation," *IEEE Transactions on Information Theory*, 50, 2231–2242.

Turlach, B. A., Venables, W. N., and Wright, S. J. (2005), "Simultaneous Variable Selection," *Technometrics*, 47, 349–363.

Van Roy, B. (1998), "Learning and Value Function Approximation in Complex Decision Processes," Ph.D. thesis, Massachusetts Institute of Technology.

Watkins, C. J. C. H. (1989), "Learning from Delayed Rewards," Ph.D. thesis, Cambridge University, Cambridge, England.

Xu, X., Xie, T., Hu, D., and Lu, X. (2005), "Kernel Least-Squares Temporal Difference Learning," *International Journal of Information Technology*, p. 5463.

Zhang, T. (2009), "On the Consistency of Feature Selection using Greedy Least Squares Regression," *Journal of Machine Learning Research*, 10, 555–568.

Zou, H. and Hastie, T. (2005), "Regularization and variable selection via the elastic net," *J. R. Statist. Soc. B*, 67, 301–320.

Biography

Christopher Robert Painter-Wakefield was born Christopher Robert Wakefield in 1968 in Manila, The Philippines. Most of his later childhood and youth was spent in Richmond, Virginia. He met Patty Painter, his eventual wife, while in college at Wake Forest University. He obtained a B.S. in Physics from Wake Forest in 1990, after which he moved to Oklahoma for his first try at graduate school (in Physics). After leaving graduate school and working various jobs, he eventually found his way into computer programming, pursuing that career for over a decade before heading back to graduate school in Computer Science. He obtained his Ph.D. in Computer Science from Duke University in May 2013.