Model-based Reinforcement Learning with State and Action Abstractions

by

Hengshuai Yao

A thesis submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Computing Science

University of Alberta

Abstract

In model-based reinforcement learning a model is learned which is then used to find good actions. What model to learn? We investigate these questions in the context of two different approaches to model-based reinforcement learning. We also investigate how one should learn and plan when the reward function may change or may not be specified during learning.

We propose an off-line API algorithm that uses linear action models to find an approximate policy. We show that the new algorithm performs comparably to LSPI, and often converges much quicker. We propose a so-called pseudo-MDPs framework. In this framework, we learn an optimal policy in the pseudo-MDP and then pull it back to the original MDP. We give a performance error bound for the approach. Surprisingly, the error bound shows that the quality of the policy derived from an optimal policy of the pseudo-MDP is governed only by the policy evaluation errors of an optimal policy in the original MDP and the "pull-back" policy of an optimal policy in the pseudo-MDP. The performance error bound of the recent kernel embedding AVI can be derived using our error bound. The pseudo-MDP framework is interesting because it not only includes the kernel embedding model but also opens the door to new models.

We introduce a so-called universal option model. The problem we address is temporal abstract planning in an environment where there are multiple reward functions. A traditional approach for this setting requires a significant amount of computation of option return for each reward function. The new model we propose enables a very efficient and simple generation of option returns. We provide algorithms of learning this model as well as planning algorithms for generating returns and value functions. We also prove the convergence of these algorithms.

Preface

Most of the work included in this thesis is a collaborative effort of the author with his supervisor Csaba Szepesvari, Professor at Department of Computing Science, University of Alberta. For Chapter 2, the author's work include ideas and experiments. For Chapter 3, the author's work include the ideas, proof and experiments. Csaba Szepesvari provided feedbacks and improved all these work.

The ADMM algorithm in Chapter 4 for computing the constrained optimization models is developed by Xinhua Zhang (now Assistant Professor at Department of Computer Science, University of Illinois at Chicago) when he was working at the Machine Learning Research Group of National ICT Australia (NICTA). Bernardo A. Pires, Ph.D student Department of Computing Science, University of Alberta, contributed to speeding up the ADMM algorithm.

Joseph Modayil (now at Google DeepMind in London, UK) contributed to developing the ideas and experiment design in Chapter 5 when he was working at Department of Computing Science, University of Alberta. Rich Sutton, Professor at Department of Computing Science, University of Alberta, contributed to developing the ideas of Chapter 5.

To Dongcui, Alexander and YingYing

For your love.

Do the difficult things while they are easy and do the great things while they are small.

– Laozi.

Acknowledgements

Csaba Szepesvári has been a wonderful support on many topics. I couldn't overstate how much I learned from him. His critical thinking, hard working and team working spirits have inspired and enriched me. Rich Sutton's work on temporal difference learning aroused my interests in reinforcement learning in 2002, and his work on planning has inspired a main part of this thesis. Dale has supported me on using reinforcement learning for ranking webpages. I appreciate his open-mindedness and encouragement. Davood has been an invaluable support on web search and link analysis. His advice for me is, *How do you evaluate it?* I'd like to thank Joseph Modayil for guiding me to the road of scientific and rigorous writing with his good example. Xinhua Zhang has been a wonderful support of optimization problems. Bernardo Avila Pires has been a great team mate. Thanks to David Szepesvári for careful reading and providing many feedback that improves this thesis.

The following document uses the first person plural to indicate the collaborative nature of much of the work.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Reinforcement Learning

Sequential decision making means a setting where a computer agent or a human makes a decision on choosing some action in order to achieve a certain goal; the decision is made continuously, usually updated when a new observation of the environment arrives. The problem is universal to cover fundamental problems in Artificial Intelligence. Examples of sequetial decision making include large board games (Tesauro, 1994; Silver et al., 2007), helicopter piloting (Coates et al., 2008), elevator scheduling (Crites and Barto, 1995), investment strategy modeling (Choi et al., 2007), and complex power system management (Powell et al., 2012).

Reinforcement learning is the learning framework for sequential decision making. The key that leads to these success is that reinforcement learning is grounded in maximizing long-term rewards through observations in a highly scalable way. Dynamic Programming has been successful in learning and planning, but it is limited to small problems because it requires an accurate model of the environment (Sutton and Barto, 1998). By the use of state abstraction and function approximation, reinforcement learning is scalable to problems with large state spaces. By the use of action abstraction, reinforcement learning is scalable to problems with large action space. We will review these two topics in the literature and present our contribution in this thesis.

We first review necessary concepts in reinforcement learning briefly, including state and reward, policy, value function, state abstraction, transition, action abstraction and Bellman equation.

Reinforcement learning problems can be categorized into two fundamental problems: policy evaluation (learning) and control (planning).

1.1.1 States and Rewards

Reinforcement learning uses the “agent-environment” framework to describe sequential decision making. The *agent* is the party that makes decisions and takes actions. The *environment* is the party that collects the status of the other objects and reports the information to the agent. The agent takes actions. In reponse, the environment updates its status report, reflecting the agent’s influence on the other objects. The environment can contain other agents as well; so the framework is general enough to cover multi-agent scenarios.

The status of the environment is called the *state*, summarizing all the information of the environment. The state is temporal, which is updated by some (usually unknown) function in the environment every time step. In addition, the environment also gives a *reward* (a scalar signal) that can be observed by the agent. In this framework, the agent follows some policy to take an action at the current state, and then receives the reward and observes the new state; the iteration continues. Reinforcement learning is all about evaluating policies and choose the best policy to maximize future rewards. Normally we assume there is a reward (utility) function that generates the rewards after taking actions. The reward function is goal dependent; in practice, it is usually provided by a system designer in order to achieve some goal.

1.1.2 Decision Making and Policies

A policy is a way of choosing actions. In this thesis, we assume that all policies are deterministic. A *deterministic policy* takes in a state and maps to an action. Policy gives us a device to talk about both low-level and high-level decision making. Low-level and high-level decisions really mean the time scale at which decisions are made. A low-level decision is an action to take at a time step, while a high level decision is a sequence of actions to take over multiple time steps. The following is an everyday example to illustrate low- and high-level decision making.

Going-to-library example: Alex decides to go to library when sitting at the chair in an office at the Computer Science building. With this goal, he stands up, walks out of the office, makes a turn to the hallway, steps out of the building and takes a walk or drives to the library. Each step can be decomposed into millions of tiny scales where the human body is making decisions on taking actions.

In this process, we see different levels of decision making.With a specified goal, high level decision of choosing a way of going to the library, *e.g.*, by car or on foot. This process chooses a policy, which is called *planning* or *control* in reinforcement learning terminology. Third, once the policy is decided, it is then followed during the process. For example, walking his steps to the office door, making a turn, locking the door, and turning to the hallway are all traces of following the policy.

Like this example, we may face a reinforcement learning problem every moment in real life. A baby may learn that if she smiles then her mother will kiss her. So she may figure out that, "when I see mother, I will smile but not cry." Here she is making a decision to follow a policy that will give her a reward. The policy will be triggered at the special states—when she sees her mother. Reinforcement learning is the science that generalizes these daily life examples, with a goal to help people build computer programs that help people make decisions.

1.1.3 Value Function and Recursive Temporal Dependence

There are "good" and "bad" states when making a decision. In reinforcement learning, good/bad states are those that lead to large/small future rewards starting from there. A state has a value. The value of a state is associated with policies. A state has multiple values—each associated with a policy from the state. The same state can have a very large value under a policy, but a small value under another.

In the going-to-library example, the beginning state is sitting at the chair, one intermediate state can be standing at the entrance of the Computing Science department building; and the library is the *goal state*. Suppose the state of arriving the library has a reward 1.0, and all the other states have reward 0.0 if he chooses to walk; otherwise -1.0 because of fuel cost if he chooses to drive. Suppose he chooses to walk, then all the states have a value 1.0. It helps to work backward from the goal state. The intermediate state leads to the library, so it is natural for it to have a value 1.0. From the office, he walks to the entrance of the building, so the office which is the initial state should have a value 1.0. Suppose he chooses to drive, then he suffers negative rewards when reaching the library. Thus driving is an inferior policy to walking in this example. We can see that the all the states except the goal(library) state have different values under the two policies.

The importance of states is defined in a forward way. The *value* of a state is defined by the future states that can be reached from the policy from the state, or more precisely, by the future rewards of following the policy from this state. The definition of state values has a temporal nature, which looks forward in time to search for rewards at future states to define the value of the current state. Indeed this backward propagation is an important feature of reinforcement learning. In early days, people used "delayed reward/credit assignment" to describe the nature of reinforcement learning since in many early studied problems there comes a nonzero reward only when reaching the goal state (Sutton, 1984). An intuive way to think of reinforcement learning is, if the goal state is reached, what are the values for the previous states in the best policy? Correctly figuring out the answer will help future decision making in similar situations. Backward propagation is a ubiquitous pattern seen in many reinforcement learning algorithms.

In defining the value of a state under a given policy, it suffices to align it to the immediate next states. This results in a recursive definition—every state respects the immediate next states that can be reached in one step. This recursion is known as *Bellman equation* or *Bellman optimality*, which is a fundamental fact in reinforcement learning and dynamic programming.

1.1.4 State Abstraction

In some problems, states are not observable. Sometimes states are observable but we need to generalize to new states that are not previously observed. State abstraction deals with state representation, often called features. State abstraction is ubiquitous in reinforcement learning and machine learning. In this thesis we will consider the parametric approach to the state representation which involves a number of parameters to be learned by algorithms. State abstraction enables algorithms to deal with large discrete-state problems and continuous-state problems. Algorithms without state abstraction are usually only applicable to small problems. Representation learning and parameter learning are important problems in reinforcement learning.

1.1.5 Transition

In reinforcement learning, sample comes in the form of a tuple $\langle x_i, a_i, x_{i+1}, r_i \rangle, i = 1, 2, \ldots$, where at state x_i the agent takes action a_i and then observes the next state x_{i+1} and a reward r_i. This sample is called a transition of action a_i. If a_i is selected according to a policy, it is also called a sample of following the policy. If these samples are presented to algorithms in a real-time fashion, *i.e.*, algorithms can access a sample immediately after it happens, then this is called *online learning*. If samples are given as a batch to algorithms, then this is called *offline learning*. Samples are interaction experience of the agent with the environment, and provide statistics of taking actions. From samples, an algorithm can figure out the value function directly, which is called a *direct* algorithm or a *model-free* approach. An algorithm can also first build some model (*e.g.*, the probabilities of reaching a state after taking an action from some other state) from samples and then use the model to derive the value function, which is called a *model-based* approach.

1.1.6 Action Abstraction

The motivation of action abstraction is to facilitate high-level planning. Instead of reasoning with single-step transitions, with action abstraction one can reason about the consequence after taking a sequence of actions from given states. *Option*, a data structure comprising of a policy, a termination function and an initiation set is convenient for this purpose. With options, one can ask questions like the following examples.

"From a state, if one takes actions according to a policy, what will happen after five time steps?" In this case, the termination function is specified by the number of time steps.

"What will happen if one reaches the library in the going-to-library example?" The termination condition is specified by states.

The consequence of executing an option often include, *e.g.*, what states the agent will be at, and how many rewards can one achieve when an option stops? In order to answer these questions, certain models of the world need to be built. Indeed, option is a good example of model-based planning.

1.1.7 Policy Evaluation

Policy evaluation deals with questions like "if I follow this policy (from any given state), what are the expected rewards in the future"? In otherwords, *policy evaluation* is the problem of estimating the value function given a policy. Depending on the means of obtaining this estimate, there are model-based and model-free approaches. Model-based approach first estimates some model from following the policy and then solves the model to have the value function estimate. Model-free approach estimates the value function directly from samples. Some policy evaluation algorithms are based on states without abstraction. Other algorithms are based on state abstraction.

1.1.8 Control

The *planning* or *control* problem in reinforcement learning is to find a policy that brings the largest future rewards in the future. One can think of the control problem as to look for the largest values for states in the best situation—"I am going to be optimal. My value cannot be bigger than following the optimal policy." A planning algorithm often have a counterpart policy evaluation algorithm, the value of a state under the optimal policy also recursively looks forward the future states from the state. Similar to policy evaluation, there are also model-based and model-free approaches.

1.2 Problems Studied and Contributions

In this thesis, we study two problems. One is finding a near-optimal policy using value function approximation, i.e., control learning. The other is evaluating a high-level policy in temporal abstract planning. There are three contributions in this thesis:

- In Chapter 3, we propose an off-line API algorithm that uses linear action models to find an approximate policy. We show that the new algorithm performs comparably to LSPI, and often converges much quicker (Yao and Szepesvári, 2012).

- In Chapter 4, we propose a so-called pseudo-MDPs framework. In this framework, we learn an optimal policy in the pseudo-MDP and then pull it back to the original MDP. We give a performance error bound for the approach. Surprisingly, the error bound shows that the quality of the policy derived from an optimal policy of the pseudo-MDP is governed only by the policy evaluation errors of an optimal policy in the original MDP and the "pull-back" policy of an optimal policy in the pseudo-MDP. This result is interesting because performance error bound of the recent kernel embedding AVI (Grünewälder et al., 2012) can be derived using our error bound (Yao et al., 2014a). The pseudo-MDP framework is interesting because it not only includes the kernel embedding model (Grünewälder et al., 2012) but also opens the door to new models. We propose a least-squares approach and a constrained optimization approach of learning factored linear models, which can be used for AVI. We studied the performance of the new AVI algorithms and explored feature construction based on them.

- In Chapter 5, we introduce a so-called universal option model. The problem we address in this chapter is temporal abstract planning in an environment where there are multiple reward functions. A traditional approach for this setting requires a significant amount of computation of option return for each reward function. The new model we propose enables a very efficient and simple generation of option returns. We provide algorithms of learning this model as well as planning algorithms for generating returns and value functions. We also prove the convergence of these algorithms (Yao et al., 2014b).

When working on this thesis, we proposed a linear Dyna-style planning that uses multi-step predictions. We showed empirically that this multi-step prediction Dyna gives better results for both policy evaluation and control. We studied issues in implementing linear Dyna-style planning, including step-sizes for learning, modeling, and planning. With careful implementation, we demonstrated that linear Dyna as a model-based algorithm learns faster than model-free TD learning and Q-learning (Yao et al., 2009b).

We also proposed a new two-time scale gradient-descent algorithm called LMS-2 that is useful to the literature of machine learning and RL (Yao et al., 2009a). LMS-2 has the same order of complexity as Least-Mean-Square (LMS) but it improves the learning speed to near that of recursive least-squares. These two works are not included in this thesis.

Chapter 2

Background

In this chapter we provide necessary background on reinforcement learning (RL). All results presented in this section can be found in (Puterman, 1994; Bertsekas and Tsitsiklis, 1996; Szepesvári, 2010).

2.1 Markov Decision Processes

We use Markov Decision Processes (MDPs) as a theoretical framework to study RL. Thus we define an MDP by a 5-tuple,

$$(\mathcal{X}, \mathcal{A}, (\mathcal{P}^a)_{a\in\mathcal{A}}, (f^a)_{a\in\mathcal{A}}, \gamma),$$

where $\mathcal{X}$ is the state space which we will assume to be finite, [1] $\mathcal{A}$ is the finite action space, $\mathcal{P}^a$ is a transition model with $\mathcal{P}^a(x, x')$ being the probability of transitioning to state x' after taking action a at state x, f^a is a reward model with $f^a(x, x')$ being the immediate reward of the state transitioning, which is a real number, and $\gamma \in [0, 1)$ called a discount factor. An MDP gives rise to a sequential decision process, where at each step of the process an action has to be chosen based on the past observations, leading to a next observed state X' sampled from the conditional distribution $\mathcal{P}^a(\cdot|x)$, where x is the current state and a is the action chosen. While transitioning to X', a reward $f^a(x, x')$ is incurred and also observed.

The question then is how to choose actions so that the expected total discounted reward incurred is maximized regardless of the initial state for the process. A standard result is that this can be achieved by choosing an action A_t at time step t

[1]Finiteness assumption of the state spaces is not essential but it simplifies presentation.

from some distribution $\alpha(X_t, \cdot)$. Note α can be viewed as $\alpha : \mathcal{X} \times \mathcal{A} \to [0,1]$ with $\sum_{a\in\mathcal{A}} \alpha(x,a) = 1$ for all $x \in \mathcal{X}$. Such functions will be called stationary Markov policies and the above describes how a given stationary Markov policy can be "followed" in an MDP. We denote the set of stationary Markov policies by Π.

Value functions. We will denote by $V^\alpha(x)$ the expected total discounted reward incurred while following α from some state $x \in \mathcal{X}$:

$$\begin{aligned} &V^\alpha(x) \\ &= \mathbb{E}\Big\{ \textstyle\sum_{t=0}^{\infty} \gamma^t f^{A_t}(X_t, X_{t+1}) \,\Big|\, \\ &\qquad\qquad X_0 = x, X_{t+1} \sim \mathcal{P}^{A_t}(\cdot|X_t), A_t \sim \alpha(X_t, \cdot), t = 0, 1, \ldots \Big\}. \end{aligned}$$

One can show that V^α is well defined. V^α is called the *state value function* or *value function* for short. The problem of evaluating some policy α is to find a "good" approximation to its value function V^α. Since the value function is often estimated from data, the process in which the approximation is constructed is called *learning*.

The *optimal value function* is $V^* : \mathcal{X} \to \mathbb{R}$,

$$V^*(x) = \sup_{\alpha\in\Pi} V^\alpha(x),$$

for all $x \in \mathcal{X}$. That is, the optimal value function is the value function that satisfies $V^*(x) \geq V^\alpha(x)$, for all $x \in \mathcal{X}$ and $\alpha \in \Pi$. Note V^* can be shown to exist and must be well defined. A policy whose value function is V^* is called an *optimal policy*. Note, there can be more than one optimal policy for an MDP, but there is one and only one optimal value function.

The state-action value functions which we define next facilitate control by extending the concept of the value function to include also actions. In particular, for a state-action pair (x, a), its value under policy α is defined by the expected total discounted reward of taking a at state x and behaving henceafter according to α:

$$\begin{aligned} Q^\alpha(x,a) = \mathbb{E}\Big\{ &\sum_{t=0}^{\infty} \gamma^t f^{A_t}(X_t, X_{t+1}) \,| \\ &X_0 = x, A_0 = a, A_t \sim \alpha(X_t, \cdot), X_{t+1} \sim \mathcal{P}^{A_t}(\cdot|X_t), t = 1, 2, \ldots \Big\} \end{aligned}$$

It follows from the definition that

$$Q^\alpha(x,a) = \mathbb{E}\{ f^a(x, X') + \gamma V^\alpha(X') |\, X' \sim \mathcal{P}^a(\cdot|x)\}, \tag{2.1}$$

Let Q^* be the state-action value function of an optimal policy. We also call Q^* the *optimal state-action value function*. Note Q^* can be used to derive V^*:

$$V^*(x) = \max_{a \in \mathcal{A}} Q^*(x, a), \tag{2.2}$$

for all $x \in \mathcal{X}$ and $a \in \mathcal{A}$ (Bertsekas and Tsitsiklis, 1996).

Equation 2.1 and 2.2 imply that the optimal state value function satisfies the following so-called *Bellman equation*:

$$V^*(x) = \max_{a \in \mathcal{A}} \sum_{x' \in \mathcal{X}} \mathcal{P}^a(x, x')[f^a(x, x') + \gamma V^*(x')]. \tag{2.3}$$

Any policy that selects actions at state x from the set of those that give the maximum on the right-hand side (RHS) of either (2.2) or (2.3) is an optimal policy. Those maximizing actions are called "greedy" with respect to V^* (if equation 2.3 is used) or Q^* (if equation 2.2 is used).

Dynamic programming iteratively updates an estimation of V^*:

$$V(x) \leftarrow \max_{a \in \mathcal{A}} \sum_{x' \in \mathcal{X}} \mathcal{P}^a(x, x')[f^a(x, x') + \gamma V(x')].$$

This algorithm is called *value iteration*. Value iteration is guaranteed to converge to the solution of Bellman equation given that all states are updated infinitely many times (Bertsekas and Tsitsiklis, 1996).

Value iteration applies only one sweep of policy evaluation. The effect is that in value iteration the value function is not solved until convergence for the most recent policy. Another way of generating an estimate of V^* is *policy iteration*, in which the value function of the most recent policy is solved accurately before updating the policy.

2.2 Value Function Approximation

Value function approximation provides generalization among states for problems with a large state space. Linear function approximation is widely used in practice, because of its elegance, simplicity and ease of interpretation. Given d ($d < |\mathcal{X}|$)

feature functions $\varphi_j(\cdot) : \mathcal{X} \mapsto \mathbb{R}$ ($\mathbb{R}$ is the set of real numbers), $j = 1, \ldots, d$, the feature vector of state x is [2]

$$\phi(x) = [\varphi_1(x), \varphi_2(x), \ldots, \varphi_d(x)]^\top.$$

The value of a state x (under some policy) is then approximated by $\hat{V}(x) = \phi(x)^\top \theta$, where θ is a d-dimensional weight vector to be "learned".

To approximate the state-action value function, we often use state-action features. Similar to previous passage, let $\psi(x, a) \in \mathbb{R}^{d_0}$ be the feature vector of the state-action pair, (x, a). The value of (x, a) is then approximated by $\hat{Q}(x, a) = \psi(x, a)^\top \theta$, where θ is a weight vector. In this thesis we adopt the most widely used practice for the state-action features, which is, first using d features for the states and then using a lookup table for the actions. Thus $\psi(x, a)$ and θ have $|\mathcal{A}| \times d$ components.

With function approximation introduced, we will seek approximate solutions to Bellman equation.

2.3 Policy Evaluation Algorithms

In the section, we briefly review some algorithms for learning the state value function of some fixed policy, including Temporal Difference (TD), least-squares TD (LSTD), Dyna and linear Dyna for policy evaluation. They all use linear state value function approximation. These algorithms learn from experience, i.e., sample transitions of following the policy to be evaluated). For other relevant algorithms, please refer to (Nedič and Bertsekas, 2003; Bertsekas et al., 2004; Geramifard et al., 2007; Yao and qiang Liu, 2008). In the end of this section, we also review off-policy learning, which is the problem of policy evaluation when the transition in the data may come from a policy different from the one being evaluated.

Both TD and LSTD are incremental, in the sense that some data structures are updated after every transition and can be implemented conveniently online. TD

[2] We use "feature" short for "feature vector" when there is no risk of causing confusion in the remainder of this thesis.

updates a weight vector (Sutton and Barto, 1998); LSTD updates a matrix and a vector in addition to a weight vector (Bradtke and Barto, 1996; Boyan, 2002).

2.3.1 TD

TD learns according to the difference in the predictions of the values of the current and next states. In the tth transition sample, given state X_t, we choose action A_t according to the fixed policy α, and observe a reward $R_t = f^{A_t}(X_t, X_{t+1})$ where the next state is X_{t+1}. TD adjusts the weight vector by

$$\theta_{t+1} = \theta_t + \eta_t(R_t + \gamma\theta_t^\top\phi_{t+1} - \theta_t^\top\phi_t)\phi_t,$$

where $\phi_t = \phi(X_t)$ and η_t is a positive step-size that is chosen by the user of the algorithm. TD is guaranteed to converge with a diminishing step-size (Tsitsiklis and Van Roy, 1997) to the fixed point which is the solution to the following linear system, $\mathbb{A}\theta + b = 0$, with

$$\mathbb{A} = \mathbb{E}[\phi_t(\gamma\phi_{t+1} - \phi_t)^\top], \quad b = \mathbb{E}[\phi_t R_t].$$

2.3.2 LSTD

LSTD removes the step-size in TD learning. It builds a matrix $\mathbb{A} \in \mathbb{R}^{d\times d}$ and vector $b \in \mathbb{R}^d$ from the transition experience:

$$\mathbb{A}_{t+1} = \mathbb{A}_t + \phi_t(\gamma\phi_{t+1} - \phi_t)^\top, \quad b_{t+1} = b_t + \phi_t R_t.$$

Then one solves a linear system, $\mathbb{A}_{t+1}\theta_{t+1} + b_{t+1} = 0$. In the case that $\mathbb{A}_{t+1}$ is not invertible, one often adds to the matrix a diagonal matrix of uniform negative entries. LSTD has been shown to learn more accurate solutions for policy evaluation than TD given the same amount of data in some domains (Boyan, 2002). LSTD converges to the same fixed point as TD.

2.3.3 Dyna and Linear Dyna

Dyna is an integrated architecture that simultaneously learns a value function, builds a model, and uses the model to speed up learning (Sutton, 1990). It uses TD methods to update the value function from samples. It also records the samples and use the samples to build a model in order to predict the outcome given a state. In the inner loop, first a state is sampled according to some distribution (not necessarily the distribution of the samples), and then the model is applied to project into the expected next reward and expected next state. This projected experience is then treated as a real-time sample and TD learning is re-applied. Dyna is successful in that its model is grounded in experience of the world; and the model can be used to speed up learning.

Linear Dyna is built on the idea of Dyna and extends to linear function approximation. Linear Dyna does not have to record the samples to build a model as Dyna does. Instead, the model is a compressed data structure, and is updated incrementally from samples. The linear Dyna for policy evaluation builds a model—a matrix and a vector pair, (F, e), where $F \in \mathbb{R}^{d \times d}$, $e \in \mathbb{R}^d$. The model is sample dependent, and is updated on the tth transition by using

$$e_{t+1} = e_t + \eta_t (R_t - \phi_t^\top e_t), \tag{2.4}$$

and

$$F_{t+1} = F_t + \eta_t (\phi_{t+1} - F_t \phi_t) \phi_t. \tag{2.5}$$

Given a feature vector ϕ, $\phi^\top e$ predicts the expected one-step reward and $F\phi$ predicts the expected next feature vector following the policy. In the inner loop we sample a feature vector ϕ and apply the model to generate projected experience. Then TD method is re-applied:

$$\theta_{t+1} = \theta_t + \eta_t (e_{t+1}^\top \phi + \gamma \theta_t^\top F_{t+1} \phi - \theta_t^\top \phi) \phi.$$

Linear Dyna for policy evaluation converges to the same solution as TD method under mild conditions (Sutton et al., 2008).

The accuracy of the model is important to the performance of linear Dyna. The model can be updated using recursive least-squares to speed up model learning and have better performance than model-free TD learning for online policy evaluation and control (Yao et al., 2009b).

2.3.4 On-policy vs. Off-policy Learning

Evaluating a policy using samples collected from following itself is called *on-policy learning*. This is what we considered so far. Evaluating a policy using samples collected from another policy is called *off-policy learning*. In an off-policy learning problem, the policy to evaluate is called the *target policy*, and the policy that collects samples is called the *behavior policy*. Off-policy learning is very useful since sample collection costs time and money and we would like to learn as many policies as possible from data. More importantly, control learning often needs to evaluate off-policy learning. However, off-policy learning is much more challenging than on-policy learning because the sample distribution follows the behavior policy and is usually not the same as that of the target policy. In particular, it may fail to be representative of the target policy. TD method is not guaranteed to converge for off-policy learning. Recently, fast gradient methods with a complexity of linear in the number of features were proposed and are guaranteed to converge (Sutton et al., 2009b,a).

2.4 Control Algorithms

In this section, we introduce control algorithms including approximate value iteration, Q-learning, Least-Squares Policy Iteration (LSPI), and linear Dyna for control. They all aim to approximate the optimal state-action value function Q^* and all of them use state-action value function approximation.

2.4.1 Q-learning

Let (X_t, A_t, R_t, X_{t+1}) be the transition sample at time step t, where A_t can be chosen by any mechanism based on past observations. Let $\psi_t = \psi(X_t, A_t)$, $A^*_{t+1} = \arg\max_{a\in\mathcal{A}} \psi(X_{t+1}, a)^\top \theta_t$. The update rule of Q-learning is,

$$\theta_{t+1} = \theta_t + \eta_t \left[R_t + \gamma \psi(X_{t+1}, A^*_{t+1})^\top \theta_t - \psi_t^\top \theta_t \right] \psi_t.$$

In Q-learning, usually A_t is selected to be the maximizing action of $\psi(X_t, a)^\top \theta$ for $a \in \mathcal{A}$. To collect good samples, one often selects A_t to be some exploration action, such as in the epsilon-greedy policy (Sutton and Barto, 1998).

Q-learning is usually used as an AVI algorithm. However, it can be also used as an API algorithm as follows. There are two weight vectors. One can keep the first weight vector fixed for some time steps and use it for action selection. The second weight vector is updated every time step. Then the first weight vector is set to the second weight vector. The procedure repeats until the first weight vector change is smaller than some threshold.

2.4.2 LSPI

LSPI is an API algorithm. Given the tth sample, LSPI performs greedification according to its state-action value function estimate $\hat{Q}$ to obtain A^*_{t+1} at X_{t+1} in the same way as in Q-learning. A variant of LSTD (LSTDQ) updates a matrix $\mathbb{A} \in \mathbb{R}^{(d\cdot|\mathcal{A}|)\times(d\cdot|\mathcal{A}|)}$ and vector $b \in \mathbb{R}^{d\cdot|\mathcal{A}|}$ by

$$\mathbb{A}_{t+1} = \mathbb{A}_t + \psi_t \left[\gamma\psi(X_{t+1}, A^*_{t+1}) - \psi_t\right]^\top, \quad b_{t+1} = b_t + \psi_t R_t.$$

This accumulation is performed for all the samples where the greedy action is selected using the weight vector from last iteration. After the accumulation, θ is solved for equation $\mathbb{A}\theta + b = 0$. Then one updates the policy to be the greedy policy with respect to the previous one (Lagoudakis and Parr, 2003).

Q-learning and LSTDQ are both *off-policy*, since they both approximate the optimal policy in the long run irrespective of what policies they follow (Q-learning) or sample from (LSTDQ).

Given sufficient samples and appropriate features, LSPI will find an optimal or near-optimal policy if they converge. LSPI is considered as one of the most powerful control algorithms in the RL literature (Li et al., 2009). However, LSPI does not have convergence guarantees, unfortunately. If the state-action value function produced during iteration stays bounded, then LSPI converges (Lagoudakis and Parr, 2003). Finite-sample performance bounds for LSPI are given by Munos and Szepesvári (2008); Antos et al. (2008).

2.4.3 Linear Dyna for Control

In linear Dyna for policy evaluation the model is a single matrix-vector pair that is learned from past observations. On the other hand, the model built in linear Dyna control is a set of matrix-vector pairs, (F^a, e^a) for each action $a \in \mathcal{A}$, where $F^a \in \mathbb{R}^{d\times d}$ and $e^a \in \mathbb{R}^d$. We call this collection of pairs a *linear action model* (*LAM*). At time step t, only the model for action $a = A_t$ is updated as in 2.4 and 2.5. Note in linear Dyna for policy evaluation, the reward model e is used to predict the expected one-step reward following the policy. In linear Dyna for control, a reward model e^a is used to predict the expected one-step reward of the action. Similarly, in linear Dyna for policy evaluation, the matrix F is used to predict the expected next feature vector following the policy. In linear Dyna for control, a matrix F^a is to predict the expected next feature vector of an action a.

Given a feature vector ϕ, the expected immediate reward of action a is predicted to be $\phi^\top e^a$. The expected next feature vector is predicted to be $F^a\phi$. The state-action value function is approximated by combining these two predictions:

$$\hat{Q}(x,a) = \phi(x)^\top e^a + \gamma\theta^\top(F^a\phi(x)),\ x \in \mathcal{X}, a \in \mathcal{A}.$$

To build an approximation to Q^*, the model is used as follows. In a process that runs in parallel to the model updates, a feature vector ϕ is sampled according to some mechanism, and the greedy action

$$A^* = \text{argmax}_{a\in\mathcal{A}}[\phi^\top e^a + \gamma\theta^\top(F^a\phi)]$$

is computed. TD learning is then applied to update the weight vector θ using the data (ϕ, A^*, R, ϕ'), where $R = \phi^\top e^{A^*}$ and $\phi' = F^{A^*}\phi$.

2.4.4 Fitted Q-iteration

Fitted Q-iteration is a family of control algorithms. First define $T^* : \mathbb{R}^{\mathcal{X}\times\mathcal{A}} \to \mathbb{R}^{\mathcal{X}\times\mathcal{A}}$:

$$(T^*Q)(x,a) = \sum_{x'\in\mathcal{X}} \mathcal{P}^a(x,x')\left(f^a(x,x') + \gamma \max_{a'\in\mathcal{A}} Q(x',a')\right), \quad (x,a) \in \mathcal{X}\times\mathcal{A}.$$

Given a state-action value function Q from last iteration, one first computes Monte-Carlo approximation to $(T^*Q)(x,a)$ at selected state-action pairs, and then performs regression on the resulting points. There are counter-examples showing that fitted Q-iteration can diverge unless a special regressor is used (Baird, 1995; Boyan and Moore, 1995; Tsitsiklis and Van Roy, 1996). Special regressors that can guarantee convergence include kernel averaging (Ormoneit and Sen, 2002) and tree-based regressors (Ernst et al., 2005).

2.5 Temporal Abstraction and Options

The terminology, ideas and results in this section are based on the work of (Sutton et al., 1999) unless otherwise stated. An option, $o = (\alpha, \beta)$, has two components: [3] a policy α, and a *continuation function* $\beta : \mathcal{X} \to [0,1]$. The latter maps a state to the probability of continuing the option from the state. We assume that β is state-dependent and stationary. An option o is executed as follows. At time step t, when visiting state X_t, the next action A_t is selected according to $\alpha(X_t, \cdot)$. The environment then transitions to the next state X_{t+1}, and a reward $R_{t+1} = f^{A_t}(X_t, X_{t+1})$ is observed. The option terminates at the new state X_{t+1} with probability $1-\beta(X_{t+1})$. If it does not terminate, it continues: a new action is chosen from the policy of the option, etc. When one option terminates, another option can start.

The model of option o is a pair (R^o, p^o), where R^o is the so-called *option return* and p^o is the so-called *(discounted) terminal distribution* of option o. In particular, $R^o : \mathcal{X} \to \mathbb{R}$ is a mapping such that for any state x, $R^o(x)$ gives the expected total discounted return starting from x until the option terminates. More precisely,

$$R^o(x) = \mathbb{E}[R_1 + \gamma R_2 + \cdots + \gamma^{T-1} R_T], \tag{2.6}$$

where T is the random termination time of the option, assuming that the process $(X_0, R_1, X_1, R_2, \ldots)$ starts at time 0 at state $X_0 = x$ (*initialization*), and every time step the policy underlying o is followed to get the reward and the next state until

[3]Notice the definition of an option can also include a third component, an initialization set, indicating if the option is available at a state. In this thesis, we assume all options are available at each state. That is, the initialization set is the state space for each option.

termination. The mapping $p^o : \mathcal{X} \times \mathcal{X} \to [0, \infty)$ is a function that, for any given $x, x' \in \mathcal{X}$, gives the discounted probability of terminating at state x' provided that the option is followed from the initial state x:

$$p^o(x, x') = \mathbb{E}[\, \gamma^T \mathbb{I}_{\{X_T = x'\}} \,] = \sum_{k=1}^{\infty} \gamma^k \, \mathbb{P}\{X_T = x', T = k\} \,. \tag{2.7}$$

Here, $\mathbb{I}_{\{\cdot\}}$ is the indicator function, and $\mathbb{P}\{X_T = x', T = k\}$ is the probability of terminating the option at x' after k steps away from x.

A semi-MDP (SMDP) is like an MDP, except that it allows multi-step transitions between the states. An MDP with a fixed set of options gives rise to an SMDP, because the execution of options lasts multiple time steps. Given a set of options $\mathcal{O}$, an *option policy* is then a mapping $h : \mathcal{X} \times \mathcal{O} \to [0, 1]$ such that $h(x, o)$ is the probability of selecting option o at state x (provided the previous option has terminated). We shall also call these policies *high-level* policies. Note that a high-level policy selects options which in turn select actions. Thus a high-level policy gives rise to a standard MDP policy (albeit one that needs to remember which option was selected the last time, i.e., a history dependent policy). Let $\mathrm{flat}(h)$ denote the standard MDP policy of a high-level policy h. The value function underlying h is defined as that of $\mathrm{flat}(h)$: $V^h(x) = V^{\mathrm{flat}(h)}(x), x \in \mathcal{X}$. The process of constructing $\mathrm{flat}(h)$ given h is the flattening operation. The model of an option is constructed in such a way that if we think of the option return as the immediate reward obtained when following the option and if we think of the terminal distribution as transition probabilities, then Bellman's equations will formally hold for the tuple $\langle \mathcal{X}, \mathcal{O}, (R^o)_{o \in \mathcal{O}}, (p^o)_{o \in \mathcal{O}}, \gamma = 1 \rangle$.

Chapter 3

Linear Action Models for Approximate Policy Iteration

3.1 Introduction

LSPI is an API algorithm that takes advantage of a variant of LSTD for a high data efficiency (Lagoudakis and Parr, 2003; Szepesvári, 2010). LSPI does not build any model of the environment. [1]

In this chapter, we introduce a model-based API framework that is built upon linear action models (LAMs). LAMs were previously used in linear Dyna for control (Sutton et al., 2008), which describe transitions among state feature vectors. Compared to the standard MDP model, a LAM is compressed and abstract, which enables efficient storage and computation. We separate LAMs from linear Dyna and prioritized sweeping. The concrete advancement of this separation is that LAM can be used for off-line policy iteration and off-policy learning (Yao and Szepesvári, 2012).

3.2 Linear Action Models

In this section, we define LAMs and projection operations. We also show how to perform approximate policy evaluation and policy improvement with LAMs. For the remainder of this section, we fix an MDP $\mathcal{M} = (\mathcal{X}, \mathcal{A}, (\mathcal{P}^a)_{a\in\mathcal{A}}, (f^a)_{a\in\mathcal{A}}, \gamma)$, and a feature extractor $\varphi : \mathcal{X} \to \mathbb{R}^d$, *e.g.*, as specified in Section 2.2.

Definition 1. *A* linear model *of the MDP $\mathcal{M}$ is a list of individual linear models for the actions, $(F^a, e^a)_{a\in\mathcal{A}}$, where $F^a \in \mathbb{R}^{d\times d}$ is a matrix and $e^a \in \mathbb{R}^d$ is a vector.*

The key operations of LAM are the reward and feature projections. The projection operations of LAM enable predicting what will happen after taking an action at a state with a given feature vector $\phi \in \mathbb{R}^d$. In particular, the *reward projection* predicts the expected immediate reward after taking an action a via: $\tilde{r}^a = \phi^\top e^a$. The *feature projection* predicts the expected next feature vector via: $\tilde{\phi}^a = F^a\phi$. Given these two projection operations of LAM, we are able to perform approximate policy evaluation and policy improvement.

[1]The matrix and the vector that LSPI builds both depend on policies.

Algorithm 1 The least-squares algorithm of learning LAM.

```
Input: a data set, $\mathcal{D} = \{(\phi_i, a_i, \phi_{i+1}, r_i)\}$
Output: a set of LAM, $\{\langle F^a, e^a \rangle\}$.
Initialize $H^a$, $E^a$ and $h^a$ for all $a$, where $H^a \in \mathbb{R}^{d\times d}, E^a \in \mathbb{R}^{d\times d}, h^a \in \mathbb{R}^d$
for $i = 1, 2, \ldots, d$ do
    $a = a_i$
    Update LAM structures of $a$:
      $H^a = H^a + \phi_i \phi_i^\top$
      $E^a = E^a + \phi_{i+1} \phi_i^\top$
      $h^a = h^a + \phi_i r_i$
end
for all $a$, compute
  $F^a = E^a (H^a)^{-1}$
  $e^a = (H^a)^{-1} h^a$
```

3.3 Approximate Policy Iteration with a LAM

In this section, we show how to learn a LAM and use it for API. We give an algorithm of learning LAM. We show how to use a LAM for policy evaluation. We then provide a sample-based policy evaluation method with a LAM. Lastly, we show how to perform policy improvement and API with a LAM.

3.3.1 Learning a Least-squares LAM

In this section, we give an algorithm of learning a LAM.

Without loss of generality, we assume there is a set of sample transitions:

$$\mathcal{D} = \{(\phi_i, a_i, \phi_{i+1}, r_i)\},\ i = 1, 2, \ldots$$

The least-squares approach of learning LAM is to learn $\langle F^a, e^a \rangle$ for all action $a \in \mathcal{A}$, such that

$$F^a = \arg\min_{\mathbb{F}\in\mathbb{R}^{d\times d}} \sum_{i=1}^{|\mathcal{D}|} \mathbb{I}_{a_i=a} \cdot \|\phi_{i+1} - \mathbb{F}\phi_i\|_2^2, \tag{3.1}$$

and

$$e^a = \arg\min_{\mathbf{e}\in\mathbb{R}^d} \sum_{i=1}^{|\mathcal{D}|} \mathbb{I}_{a_i=a} \cdot (r_i - \phi_i^\top \mathbf{e})^2, \tag{3.2}$$

where $\|x\|_2$ denotes the 2-norm $\sqrt{\sum_i x_i^2}$ of vector x, and $\mathbb{I}_{a_i=a} = 1$ if $a_i = a$; otherwise $\mathbb{I}_{a_i=a} = 0$. That is, we classify the transitions by their action "label" and perform model learning for each individual class.

Algorithm 1 shows a straightforward way of computing the above least-squares solution. In the algorithm, for an action a, matrix H^a accumulates the auto-correlation of the features, matrix E^a accumulates the correlation of the feature vectors and their consecutive next feature vectors, and vector h^a accumulates the correlation of the feature vectors and the rewards. Given H^a, E^a and h^a, F^a and e^a are obtained by inverting H^a. The method does not require tuning a step-size or repeated training.

3.3.2 Policy Evaluation

In this section, we show how to evaluate a policy α using a LAM. Note although we focus on the least-squares learned LAM in this thesis, there is generally no restriction on the type of LAM that can be used in this section.

Note that α can be the optimal policy or any other policy. The transition dynamics P^α and the immediate reward r^α are defined by

$$P^\alpha(x, x') = \sum_{a \in \mathcal{A}} \alpha(x, a) \mathcal{P}^a(x, x'), \quad \forall x, x' \in \mathcal{X}$$

and

$$r^\alpha(x) = \sum_{a \in \mathcal{A}} \alpha(x, a) r^a(x), \quad \forall x \in \mathcal{X},$$

where $r^a(x) = \sum_{x' \in \mathcal{X}} \mathcal{P}^a(x, x') f^a(x, x')$. For any $V \in \mathbb{R}^{|\mathcal{X}|}$, we define

$$T^\alpha V = \gamma P^\alpha V + r^\alpha, \tag{3.3}$$

and

$$T^a V = \gamma \mathcal{P}^a V + r^a, \; a \in \mathcal{A}, \tag{3.4}$$

where P^α, $\mathcal{P}^a$, r^α and r^a are viewed as $|\mathcal{X}| \times |\mathcal{X}|$ matrices and $|\mathcal{X}|$-dimensional vectors, respectively, obtained by fixing an arbitrary ordering of the states in $\mathcal{X}$, or equivalently identifying $\mathcal{X}$ with $\{1, 2, \ldots, |\mathcal{X}|\}$.

The equations 3.3 and 3.4 define the operators, $T^\alpha, T^a : \mathbb{R}^{|\mathcal{X}|} \to \mathbb{R}^{|\mathcal{X}|}$, both of which are affine linear. The relationship between the operators T^α and T^a is

$$(T^\alpha V)(x) = \sum_{a \in \mathcal{A}} \alpha(x, a)(T^a V)(x), \quad \forall x \in \mathcal{X}. \tag{3.5}$$

Note that V^α is the fixed point of $V = T^\alpha V$ (which is Bellman equation for policy evaluation). Because of (3.5),

$$V^\alpha(x) = \sum_{a\in\mathcal{A}} \alpha(x,a)(T^a V^\alpha)(x), \quad \forall x \in \mathcal{X}. \tag{3.6}$$

With linear function approximation, $V^\alpha(x) \approx \phi(x)^\top\theta$. In this case, we want to retain (3.6). Thus

$$\phi(x)^\top\theta \approx \sum_{a\in\mathcal{A}} \alpha(x,a)(T^a(\Phi\theta))(x), \quad \forall x \in \mathcal{X},$$

where $\Phi \in \mathbb{R}^{|\mathcal{X}|\times|\mathcal{X}|}$ is the so-called feature matrix whose xth row is $\phi(x)^\top$. Expanding the definition of T^a, this rewrites into

$$\phi(x)^\top\theta \approx \sum_{a\in\mathcal{A}} \alpha(x,a)\left[\gamma(\bar{\phi}^a(x))^\top\theta + r^a(x)\right],$$

where the term, $\bar{\phi}^a(x) = \sum_{x'\in\mathcal{X}} \mathcal{P}^a(x,x')\phi(x')$, is the expected next feature vector under taking a at state x.

Since they are unknown, we replace $\bar{\phi}^a(x)$ by $\tilde{\phi}^a(x) = F^a\phi(x)$, and $r^a(x)$ by $\tilde{r}^a(x) = \phi(x)^\top e^a$. With all these, we arrive at a linear system of equations:

$$\phi(x)^\top\theta = \sum_{a\in\mathcal{A}} \alpha(x,a)\left[\gamma(\tilde{\phi}^a(x))^\top\theta + \tilde{r}^a(x)\right], \quad \forall x \in \mathcal{X}. \tag{3.7}$$

The linear system can then be solved iteratively or directly.

The solution of the linear system 3.7 is generally not guaranteed to exist for least-squares LAM, because the underlying matrix in the linear system is not necessarily invertible. Nonetheless in our experiments we found that good performance can often achieved. In case that the matrix is not invertible, one can apply the same perturbation trick as in LSTD. In Chapter 4, we propose method that can learn a LAM which guarantees the contraction property in the approximate transition model, for which the solution of 3.7 must exist.

To perform policy evaluation with a LAM, we generate a set of sample transitions, $\{(\phi, \tilde{\phi}, \tilde{r})\}$, from a set of feature vectors $\{\phi\}$ using LAM. Note that the policy to evaluate needs to select actions for feature vectors. For example, the greedy policy defined in section 3.3.3 is able to do so. We then feed these sample transitions

Algorithm 2 LAM-API with LSTD (LAM-LSTD for short).

```
Input: a set of feature vectors, $\{\phi\}$, and a LAM, $\{\langle F^a, e^a \rangle\}_{a \in \mathcal{A}}$, where $F^a \in \mathbb{R}^{d \times d}$, $e^a \in \mathbb{R}^d$
Output: a weight vector $\theta \in \mathbb{R}^d$.
Initialize $\theta$
repeat until $\theta$ no longer changes
  for $\phi$ in the data set do
      Project a sample using the greedy action:
        $a^* = \arg\max_a \{\phi^\top e^a + \gamma \theta^\top F^a \phi\}$
        $\tilde{\phi} = F^* \phi$        //$F^* = F^{a^*}$
        $\tilde{r} = \phi^\top e^*$        //$e^* = e^{a^*}$
      Accumulate LSTD structures:
        $A = A + \phi(\gamma\tilde{\phi} - \phi)^\top$
        $b = b + \phi\tilde{r}$
  end
  $\theta = -A^{-1}b$
end
```

to LSTD or any other off-policy learning algorithm. Note that this is an off-policy learning task. The distribution of ϕ is determined by the underlying state distribution, which can be an arbitrary distribution. However, the transition sample is projected according to a target policy. Hence there is a mismatch between where the source states of the transitions are from and what policy the transitions follow. Thus we need convergent off-policy learning algorithms. LSTD is suitable for this task: First it is guaranteed to converge for off-policy learning if the TD solution exists. Second, LSTD is efficient for batch data.

3.3.3 Policy Improvement

The projection power of LAM lends itself to policy improvement. In particular, given the latest policy parameter θ, policy improvement can be performed by taking

$$a^* = \arg\max_a \left[\phi^\top e^a + \gamma (F^a \phi)^\top \theta \right], \tag{3.8}$$

for any feature vector $\phi \in \mathbb{R}^d$.

Algorithm 2 shows an API algorithm with LAM for projection and LSTD for evaluation.

3.4 Empirical Results

In this section, we provide empirical results of LAM-LSTD on chain-walk examples and a cart-pole balancing problem. We compare with LSPI on the learned policy and number of iterations required for convergence. On chain-walk problems, LAM-LSTD found the optimal policy in the same number of iterations as LSPI (4-state chain), and much fewer number of iterations than LSPI for 50-state chain (LAM-LSTD: 2 iterations; LSPI: 14). On pendulum, LAM-LSTD balanced the pendulum for 2930 steps on average and LSPI balanced for 2700 steps on average both with 400 episodes of training data. The variance of LAM-LSTD is also much smaller than LSPI. LAM-LSTD spent 4.3 iterations to converge in general while LSPI spent 8.2 iterations on average for the same number of training episodes.

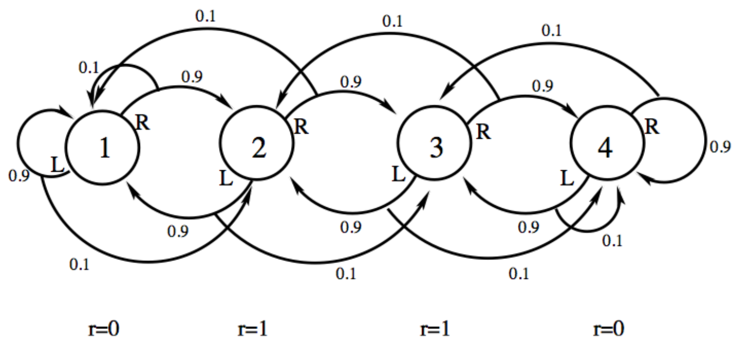

Figure 3.1: The 4-state chain-walk (figure is from (Lagoudakis and Parr, 2003)). States 2 and 3 are rewarding only.

3.4.1 The 4-state Chain

The problem is the chain-walk example originally used by Koller and Parr (2000). The problem is shown in Figure 3.1. There are two actions, "Left" (L) and "Right" (R). With probability 0.9, an action leads to a state in the intended direction; with probability 0.1, it leads to a state in the opposite direction. The discount factor is 0.9. The nonzero reward (one) is given exclusively at the middle states. Features are, $\phi(s) = [1, s, s^2]^\top$, $s = 1, 2, 3, 4$. A data set of ten counts of each state-action pair was used (notice this training data has no randomness involved, so the following experiments can be reproduced almost exactly). θ was initialized to 0 for all algorithms. The algorithms were stopped if the L_2-norm of the change of θ in two successive iterations was smaller than 0.001.

Figure 3.2 shows the performance of LAM-LSTD. In the figure, the x-axis shows the states, and the y-axis shows the following value functions:

$$\hat{Q}^*_L(s) = \phi(s)^\top e^L + \gamma(F^L\phi(s))^\top\theta,$$

represented by *blue dash-dot line, marker* '●'; and

$$\hat{Q}^*_R(s) = \phi(s)^\top e^R + \gamma(F^R\phi(s))^\top\theta,$$

represented by *red dashed line, marker* '+'. LAM-LSTD found the optimal policy in 2 iterations. Notice that the success of LAM-LSTD can be somehow predicted by checking the quality of LAM in advance. In fact, because the weight vector θ was

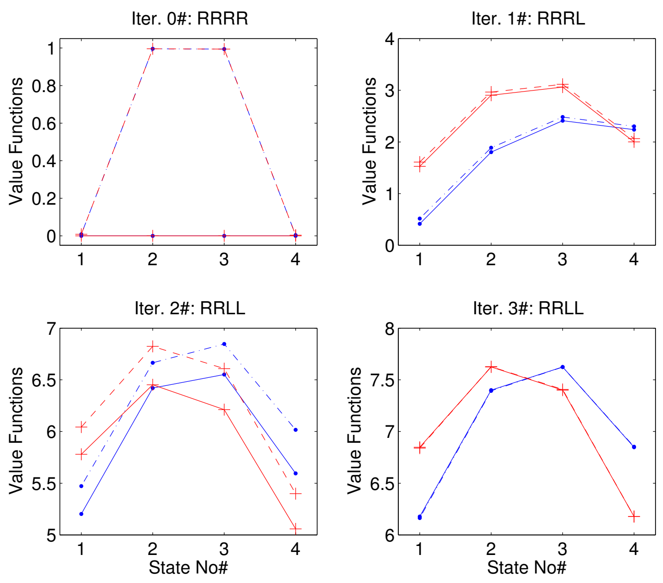

Figure 3.2: LAM-LSTD (*dashed*) and LSPI (*solid*) for the 4-state chain. The blue line shows estimate of the optimal value of going left, and the red line shows estimate of the optimal value of going right. The two algorithms finds the same policy at all the iterations. For example, at iteration 3, both algorithms find the optimal policy "RRLL" which means the optimal actions at states 1, 2, 3, 4 are going right, going right, going left, and going left, respectively.

initialized to 0, the plot at iteration 0 actually shows the reward approximations. In Figure 3.2, the first plot not only shows the approximated rewards of states 2 and 3 are bigger, but also shows that they are close to each other. For this experiment, the learned linear reward model was $e^L = e^R = [-1.9675, 2.4702, -0.4943]^\top$, [2] and reward predictions are $\phi(1)^\top e^L = 0.0084 \approx 0, \phi(2)^\top e^L = 0.9956 \approx 1, \phi(3)^\top e^L = 0.9941 \approx 1, \phi(4)^\top e^L = 0.0039 \approx 0$. The quality of F can be checked before iteration in a similar way.

Though LSPI found exactly the same policy as LAM-LSTD, the learned (state-value) value functions were different. In particular, LAM-LSTD converged faster than LSPI in the first two iterations. One reason might be, though both algorithms initialized their weight vector to 0, the initial policies were actually different. In particular, LAM-LSTD took advantage of the linear reward model, and the initial state-action value function was initialized to the one-step approximated reward. The

[2]The two reward models are the same, because the rewards only depend on the states.

initial state value function was 0 for LAM-LSTD. while for LSPI it was the initial state-action value function that was 0. This explains why at iteration 1 LAM-LSTD was a bit better, but cannot explain why at iteration 2 LAM-LSTD became much better. This difference is caused by that *LAM-LSTD takes advantage of using a learned model, while LSPI learns on the real samples*. The projected samples used by LAM-LSTD summarize across the real samples all possible results for any state through the use of LAM; the real samples used by LSPI only contain pieces of single-step transitions. Therefore policy improvement at one projected sample by LAM-LSTD actually considers many transitions in real samples. In this example, we saw that LSPI caught up in iteration 3. We expect the advantage of LAM-LSTD increases with problem size with a good model.

3.4.2 The 50-state Chain

For the 50-state chain, the reward one is now given exclusively at states 10 and 41. We also used exactly the same radial basis features as Lagoudakis and Parr (2003), giving 22-dimensional state-action features (used by LSPI), and 11-dimensional state features (used by LAM-LSTD). We used a sample set comprising 100 pairs for each state-action (this gives $100 \times 50 \times 2 = 10,000$ samples).

In total, LSPI spent 14 iterations to converge, and found the optimal policy at the last iteration. The initialization plus two early iterations are shown in Figure 3.3. Three later iterations in Figure 3.4. There is a noticeable policy degradation from iteration 7 to iteration 8. At iteration 8, the shape of the state-action value function looks much worse than the previous iteration.

LAM-LSTD spent 4 iterations to converge, and found the optimal policy at iteration 2 as shown in Figure 3.5. After one iteration, LAM-LSTD already found a nearly optimal policy. In the figures, both algorithms used the same initialization for policy evaluation. In particular, the matrix and vector in both LSTDQ and LSTD were initialized to 0. H^a used by the least-squares algorithm of learning LAM was initialized to $100I$ (I is the identity matrix) for both actions. We also tried extensively to optimize the initialization of this matrix for both algorithms, As the experiment with the 4-state chain, the weight vector was initialized to 0 for both

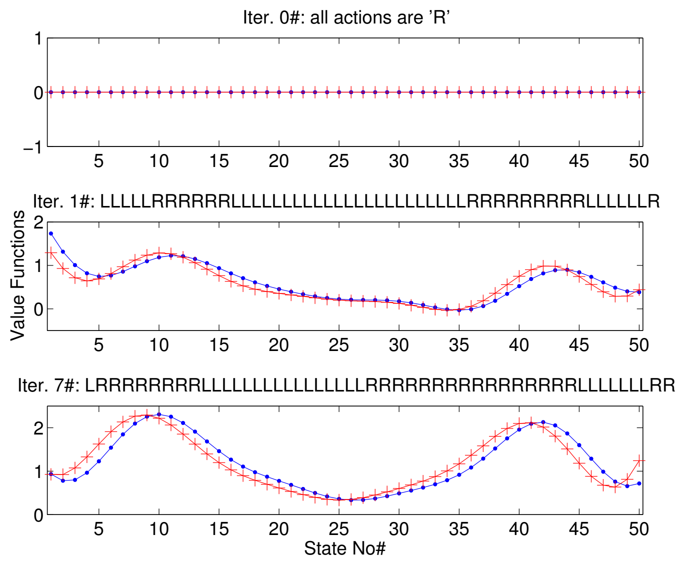

Figure 3.3: LSPI on the 50-state chain-walk, shown are iteration 0 (initialization) plus iteration 1 and 7. Like the 4-state chain, the actions for the states are shown for the iterations. Note that only states 10 and 41 are rewarding. The two bumps at iteration 7 shows the state-action value function of the policy has a good shape (but not optimal yet; in fact LSPI found the optimal policy at iteration 14).

algorithms. The first plot of LAM-LSTD, showing the reward approximations by the features, indicates that the positions of the largest rewards are modeled correctly. With five polynomial bases, however, we found the positions of the largest reward are modeled incorrectly, and hence LAM-LSTD only gave a sub-optimal policy, which is consistent to LSPI's results (Lagoudakis and Parr, 2003).

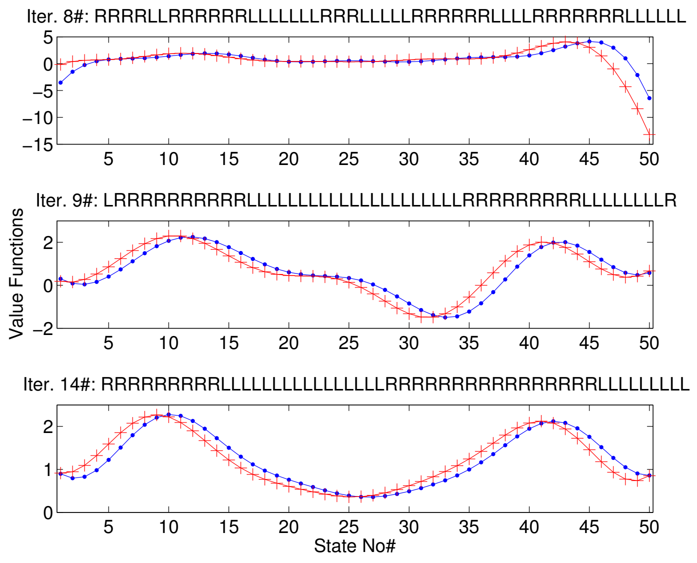

Figure 3.4: LSPI on the 50-state chain-walk *continued*: iteration 8, 9, 14. At iteration 14, LSPI converges.

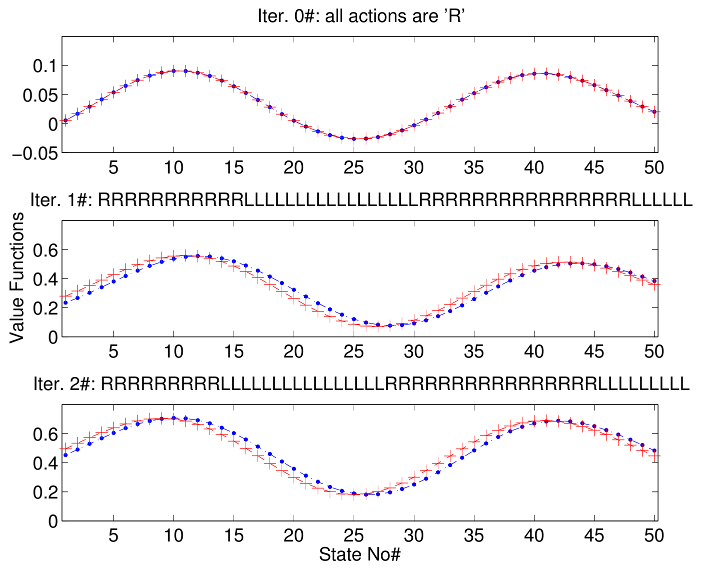

Figure 3.5: LAM-LSTD on the 50-state chain-walk. At iteration 2, the policy is already optimal. (LAM-LSTD converges in two more iterations.) The policy actions for the last two iterations remain unchanged and are optimal. We run to many iterations and it is still stable.

3.4.3 Cart-Pole System

We used exactly the same simulator as Lagoudakis and Parr (2003). The state is $(\vartheta, \dot{\vartheta})$, where ϑ is the angle between the pendulum and the vertical line, and $\dot{\vartheta}$ is the velocity of the pendulum. The goal is to keep a pendulum above the horizontal line ($|\vartheta| \leq \pi/2$) for a maximum of 3000 steps. There are three actions: applying a left or right force of 50 Newton or not applying any force. The problem is shown in figure 3.6 The discount factor is 0.95. The features are state-based RBFs, 9 regular Gaussians plus a constant basis over the 2D state space. The features of a state x are $\phi_i(x) = \exp(-||x - u_{i-1}||^2/2)$, $i = 1, 2, \ldots, 10$, where $u_0 = s$, and the other u_i are the points from the grid $\{-\pi/4, 0, \pi/4\} \times \{-1, 0, 1\}$. We collected 100 data sets, each comprising 1000 episodes of data. In each episode, the pendulum was started from a uniformly random perturbation from the state $(0, 0)$, and applied a uniformly random policy. An episode finished when the $|\vartheta| > \pi/2$. The lengths of these episodes were around 6 to 9. Each data set was then split into subsets, giving 20 tsets with $50, 100, \ldots, 1000$ episodes, respectively.

The results are shown in Figure 3.7 and Figure 3.8, which show the balancing steps and number of iterations of the algorithms respectively, given various sizes of training data. Figure 3.7 shows that LAM-LSTD policies had a significantly better control performance (in terms of both mean and variance of the balanced steps) over LSPI policies across all the sizes of training data. Figure 3.8 shows that LAM-LSTD spends much fewer iterations than LSPI. In generating the figures, a LAM-LSTD policy and an LSPI policy were learned from the same training data of a particular size, and were both evaluated 100 times to give the average performance of the policies. The experiment was then repeated on the 100 sets of training data of the identical size, to further give the final, average balancing steps for a policy. Finally, twenty such experiments were performed for all the sizes of training data. Both algorithms initialized the matrix to $-0.01I$. This leads to better performance and fewer iterations for LSPI than initializing the matrix to 0. For learning all LAM, $H^a = 0$.

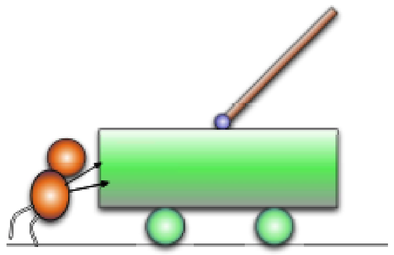

Figure 3.6: Cart-pole balancing problem.

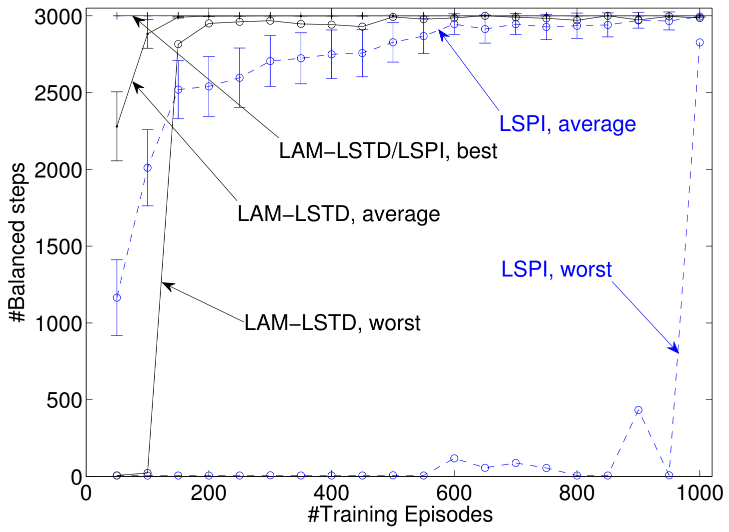

Figure 3.7: Pendulum: balancing steps of LAM-LSTD and LSPI.

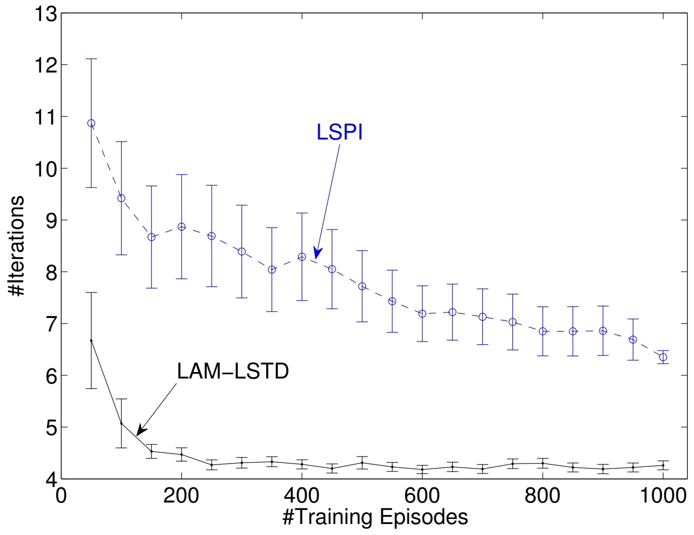

Figure 3.8: Pendulum: number of iterations of LAM-LSTD and LSPI.

3.5 Conclusion

For evaluating fixed policies, it is known that building linear models of policies is advantageous in improving the sample efficiency (Boyan, 2002; Sutton et al., 2008; Parr et al., 2008). However, these models do not generalize across policies. In this chapter, we have introduced a model-based API framework that uses linear action models learned from data. Linear action models are shared across policies, and hence can be used to evaluate multiple policies. As opposed to model-free methods, our approximate policy iteration method can use state value functions since the model can be used to predict action values. This might reduce the variance of the value- function estimates. Our empirical results show that LAM-LSTD gives better policies than LSPI, often in significantly fewer iterations. Our work suggests actions models are effective and efficient for off-policy learning and control. It would be interesting to build other (such as nonlinear) action models, from which we may have more accurate projections and hence better performance.

Chapter 4

Pseudo-Markov Decision Processes

4.1 Introduction

This chapter deals with a model-based approach for large-scale learning and planning. We want to learn a near-optimal policy for an MDP with a generic model. The desired properties of the model are as follows.

1. The optimal value function of the model can be efficiently computed.
2. It leads to a policy in the original MDP whose sub-optimality can be controlled in terms of how well the model approximates the MDP.
3. The model can be learned from data, efficiently and effectively.
4. The model is flexible.

Item 4 is not really well-defined, but we wish to use models similar to those of the previous chapter, while items 1 and 2 answer the deficiency of the approach in the previous chapter.

This chapter is based on (Yao et al., 2014a). It is organized as follows.

4.2 Preliminaries

In this chapter, we will allow continuous state space. As such, we need to change the notation slightly and introduce some more notations. In particular, now $\mathcal{P}^a$ will be a probability kernel: for all $(x, a) \in \mathcal{X} \times \mathcal{A}$, $\mathcal{P}^a(\cdot|x)$ is a probability measure over the state space, which is assumed to be equipped with a measurability structure. We also simplify (without loss of generality) the reward function f^a, which in this chapter will take only one argument: for $x \in \mathcal{X}$, $a \in \mathcal{A}$, $f^a(x)$ is the immediate reward received when action a is taken in at state x. In this chapter, we use lower case variables to denote states and actions.

For a measure μ over some measurable set W, let $L^1(\mu)$ denote the space of μ-integrable real-valued functions with domain W. Further, for a kernel $\mathcal{P}^a$ let $L^1(\mathcal{P}^a) = \cap_{x\in\mathcal{X}} L^1(\mathcal{P}^a(\cdot|x))$. We also let

$$L^1(\mathcal{P}) = \cap_{a\in\mathcal{A}} L^1(\mathcal{P}^a) = \cap_{a\in\mathcal{A}, x\in\mathcal{X}} L^1(\mathcal{P}^a(\cdot|x)).$$

We require that for any $a \in \mathcal{A}$, $f^a \in L^1(\mathcal{P}^a)$ and further that for any measurable set $U \subset \mathcal{X}$, $a \in \mathcal{A}$, $\mathcal{P}^a(U|\cdot) \in L^1(\mathcal{P})$ (in particular, $x \mapsto \mathcal{P}^a(U|\cdot)$ must be measurable). These conditions ensure that the integrals below are well-defined. Note that $L^1(\mathcal{P}^a)$ and $L^1(\mathcal{P})$ are vector-spaces.

Under some extra mild conditions, the optimal value function V^* still satisfies the "Bellman optimality equation",

$$V^*(x) = \max_{a\in\mathcal{A}} f^a(x) + \gamma \int \mathcal{P}^a(dx'|x) V^*(x'), \quad \forall x \in \mathcal{X}.$$

One simple sufficient condition is that the rewards are bounded.

For a normed vector space $\mathcal{V} = (\mathcal{V}, \|\cdot\|)$, the (induced) norm of an operator $T : \mathcal{V} \to \mathcal{V}$ is defined by $\|T\| = \sup_{V\in\mathcal{V}, V\neq 0} \|TV\|/\|V\|$. An operator is called a contraction if $\|T\| < 1$. The difference of two operators $T, \hat{T} : \mathcal{V} \to \mathcal{V}$ is defined via $(T - \hat{T})V = TV - \hat{T}V$. The supremum norm $\|\cdot\|_\infty$ of a (real-valued) function f over some set W is defined by $\|f\|_\infty = \sup_{w\in W} |f(w)|$. We will denote by $\delta_{x_0}(dx)$ the Dirac measure concentrated on x_0: $\int f(x)\delta_{x_0}(dx) = f(x_0)$ for any measurable f.

4.3 Pseudo-MDPs

We shall consider abstracting MDPs into what we call "pseudo-MDPs", which we define as follows.

Let $\mathcal{S}$ be a measurable space. Recall that a signed measure μ over $\mathcal{S}$ maps measurable subsets of $\mathcal{S}$ to reals and satisfies $\mu(\cup_i S_i) = \sum_i \mu(S_i)$ for any countable family $(S_i)_i$ of disjoint measurable sets of $\mathcal{S}$. We call the tuple

$$\mathcal{N} = (\mathcal{S}, \mathcal{A}, (\mathcal{Q}^a)_{a\in\mathcal{A}}, (g^a)_{a\in\mathcal{A}}, \gamma)$$

a *pseudo-MDP* if $\mathcal{Q}^a$ maps elements of $\mathcal{S}$ to signed measures over $\mathcal{S}$ ($\mathcal{Q}^a(\cdot|s) \doteq \mathcal{Q}^a(s,\cdot)$ is a signed measure over $\mathcal{S}$) and $g^a : \mathcal{S} \to \mathbb{R}$ is a measurable function. As for MDPs, we assume that $g^a \in L^1(\mathcal{Q})$ and for any measurable $U \subset \mathcal{S}$ and action $a \in \mathcal{A}$, $\mathcal{Q}^a(U|\cdot) \in L^1(\mathcal{Q})$, and $\gamma \in [0, 1)$.

The difference between a pseudo- and a "real" MDP is that in a pseudo-MDP $\mathcal{Q}^a(\cdot|s)$ does not need to be a probability measure. This can be useful when constructing "abstractions" of an MDP as dropping the requirement that the transition kernel must be a probability measure increases the power of what can be represented. The concepts of policies and value functions extend to pseudo-MDPs with almost no change. In particular, the concept of policies does not change. To define value functions, first notice that in an MDP, $(\mathcal{X}, \mathcal{A}, (\mathcal{P}^a)_{a\in\mathcal{A}}, (f^a)_{a\in\mathcal{A}}, \gamma)$, the value function of a stationary Markov policy is equivalently defined by defining the probability measure $\mu_{x,\alpha}$ as the distribution of the trajectories $(x_0, a_0, x_1, a_1, \ldots)$, where $x_0 = x$, and for $t \geq 1$, $a_{t-1} \sim \alpha(x_{t-1}, \cdot)$, $x_t \sim \mathcal{P}^{a_{t-1}}(\cdot|x_{t-1})$. Then

$$V^\alpha(x) = \int \sum_{t=0}^{\infty} \gamma^t f^{a_t}(x_t) d\mu_{x,\alpha}(x_0, a_0, x_1, a_1, \ldots).$$

Copying this definition, in a pseudo-MDP $\mathcal{N}$, we define the value function of a policy β, by considering the signed measures $\mu_{s,\beta}$ induced by $(\mathcal{Q}^a)_a$ and β over the set $(\{s\}\times\mathcal{A})\times(\mathcal{S}\times\mathcal{A})^{\mathbb{N}}$ of trajectories starting at some state $s \in \mathcal{S}$ as before. Then, the value function of β, v^β is defined by $v^\beta(s) = \int \sum_{t=0}^{\infty} \gamma^t g^{a_t}(s_t) d\mu_{s,\beta}(s_0, a_0, s_1, a_1, \ldots)$. We assume that v^β is finite-valued for any policy β of $\mathcal{N}$. In pseudo-MDPs, we generalize the Bellman-optimality to

$$v^* = \max_{a\in\mathcal{A}} \tag{4.1}$$

We assume for now that this has a solution. The approach followed will be to compute v^* and then a polity that is greedy with respect to v^* in $\mathcal{N}$. As before such a policy chooses actions that maximizes the RHS of 4.1. For brevity we call these policies *Bellman optimal*, and the actions they take $\mathcal{N}$*-greedy*.

The purpose of constructing pseudo-MDPs is to create abstractions that facilitate efficient computation. However, for an abstraction to be of any use, we need to be able to use it to come up with good (near-optimal) policies in the source MDP. Denote the abstracted, or *source MDP* by

$$\mathcal{M} = (\mathcal{X}, \mathcal{A}, (\mathcal{P}^a)_{a\in\mathcal{A}}, (f^a)_{a\in\mathcal{A}}, \gamma).$$

The connection of $\mathcal{M}$ and $\mathcal{N}$ will be provided by a measurable map $\phi : \mathcal{X} \to \mathcal{S}$, which must be chosen at the time of choosing $\mathcal{N}$. In what follows we fix the mapping ϕ.

We let $\Pi_{\mathcal{M}}, \Pi_{\mathcal{N}}$ be the space of policies in the original MDP and the pseudo-MDP, respectively. The map ϕ can be used to pull any policy of the pseudo-MDP back to a policy of the source MDP:

Definition 1 (Pullback Policy)**.** *Let $\mathcal{N}$ be a ϕ-abstraction of $\mathcal{M}$. The* pullback of policy *$\beta \in \Pi_{\mathcal{N}}$ is the policy $\alpha \in \Pi_{\mathcal{M}}$ that satisfies $\alpha(x, a) = \beta(\phi(x), a)$. The map that assigns α to β will be denoted by L and we will call it the pullback map. In particular, $L : \Pi_{\mathcal{N}} \to \Pi_{\mathcal{M}}$ and $L(\beta)(x, a) = \beta(\phi(x), a)$, for any $x \in \mathcal{X}$, $a \in \mathcal{A}$.*

The power of pseudo-MDPs is that it provides a common framework for many MDP-abstractions that were considered previously in the literature: Some examples are as follows.

Example 1 (Finite Models)**.** *Let $\mathcal{S}$ be finite set, for $s \in \mathcal{S}$, $a \in \mathcal{A}$, $\mathcal{Q}^a(\cdot|s)$ be a distribution over $\mathcal{S}$, $g^a : \mathcal{S} \to \mathbb{R}$ be an arbitrary function.*

Example 2 (Linear Action Models)**.** *Assume that $\mathcal{S} = \mathbb{R}^d$ where measurability is meant in the Borel sense. For each $a \in \mathcal{A}$, let $F^a \in \mathbb{R}^{d\times d}$, $f^a \in \mathbb{R}^d$. Then, for each $s \in \mathcal{S}$, $a \in \mathcal{A}$, U Borel measurable, $\mathcal{Q}^a(U|s) = \mathbb{I}_{\{F^a s \in U\}}$ and $g^a(s) = (f^a)^\top s$.*

Example 3 (Factored Linear Action Models). *Let $\mathcal{S} = \mathcal{X}$, $\psi : \mathcal{X} \times \mathcal{A} \to \mathbb{R}^d$, $\xi : \mathcal{B} \to \mathbb{R}^d$, where $\mathcal{B}$ is the collection of measurable sets of $\mathcal{X}$. Then, for $x \in \mathcal{X}$, $a \in \mathcal{A}$, $U \in \mathcal{B}$, $\mathcal{Q}^a(U|x) = \xi(U)^\top \psi(x, a)$, while g^a is arbitrary.*[1]

In the first two examples $(\mathcal{Q}^a)_a$ are probability kernels. Discrete models are typically obtained in a process known as *state aggregation* (Van Roy, 2006) in which case $\phi : \mathcal{X} \to \mathcal{S}$ is assumed to be surjective and is known as the state-aggregation function. Given ϕ, one further chooses for each $s \in \mathcal{S}$ a distribution μ_s supported on $\phi^{-1}(s) = \{x \in \mathcal{X} \,|\, \phi(x) = s\}$. Then, $\mathcal{Q}^a$ is given by $\mathcal{Q}^a(U|s) = \int \mu_s(dx)\mathcal{P}^a(dx'|x)\mathbb{I}_{\{\phi(x') \in U\}}$ and $g^a(s) = \int f^a(x)\mu_s(dx)$. Linear action models arise when the transition dynamics underlying each action is approximated via a linear model such as in the previous chapter. In this case ϕ is known as the "feature-map". Note that one can represent any finite MDP with linear models: Given a finite model $\mathcal{N}$ with state space $\mathcal{S} = \{1, \ldots, d\}$, define $\tilde{\mathcal{S}}$, the state space of the linear model, as the simplex of $\mathbb{R}^d$, $(F^a)_{j,i} = \mathcal{Q}^a(j|i)$, $f^a_i = g^a(i)$, $1 \leq i, j \leq d$.

Motivation examples. While abstractly it is clear that pseudo-MDPs are more powerful as tools to abstract MDPs, we also provide two specific examples below to illustrate this. Often, the pseudo-MDP differs from an MDP only in that the transition kernel is unnormalized (*i.e.*, for some $s \in \mathcal{S}$, $a \in \mathcal{A}$, $Q^a(\mathcal{S}|s) \neq 1$). It is temping then to normalize $(Q^a)_{a \in \mathcal{A}}$.

The first example shows that this may ruin how well $\hat{V}^*$ approximates V^*. The second example shows that normalization may ruin how well the policy obtained from $\hat{V}^*$ performs when pulled back to $\mathcal{M}$.

Example 4. *The MDP $\mathcal{M}$ is $\mathcal{A} = \{a\}, \mathcal{S} = \{1, 2\}, \mathcal{P}^a = [0.01, 0.99; 0, 1], g^a = [1, 0]$. $V^*(1) \approx 0.01, V^*(2) = 0$, $\gamma = 0.9$. The pseudo-MDP is identical to $\mathcal{M}$ except that $\mathcal{Q}^a = [0.01, 0; 0, 1]$. It is easy to see that $\hat{V}^* = V^*$. If normalizing the model (*i.e., *normalizing each row of $\mathcal{Q}^a$ by its ℓ^1 norm), we get, $\bar{\mathcal{Q}}^a = [1, 0; 0, 1]$, and $\hat{V}^*(1) = 10, \hat{V}^*(2) = 0$, a much worse approximation that we had before.*

Example 5. *The MDP $\mathcal{M}$ is, $\mathcal{A} = \{a_1, a_2\}, \mathcal{S} = \{1, 2\}, \mathcal{P}^{a_1} = [0.01, 0.99; 0, 1], \mathcal{P}^{a_2} = [0, 1; 0, 1]$, $g^{a_1}(1, 1) = 100, g^{a_1}(1, 2) = 0$, $g^{a_1}(2, 2) = g^{a_2}(2, 2) = 10, g^{a_2}(1, 2) =$*

[1]A natural restriction on g^a would be to assume $g^a(x) = (f^a)^\top \psi(x, a)$.

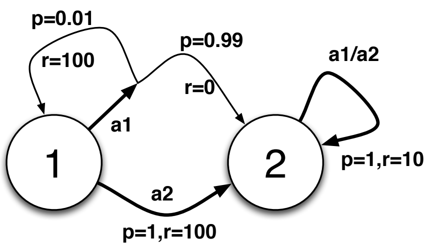

Figure 4.1: A small MDP used in Ex. 2.

100*, and the discount factor is 0.9. The optimal policy is* $\alpha^*(1) = \alpha^*(2) = a_2$*. The MDP is is shown in Figure 4.1.*

The pseudo-MDP again differs from $\mathcal{M}$ *only in the choice of* (Q^a)*. In particular,* $\mathcal{Q}^{a_1} = [0.01, 0; 0, 1], \mathcal{Q}^{a_2} = [0, 1; 0, 1]$*. The normalized model has the kernels,* $\bar{\mathcal{Q}}^{a_1} = [1, 0; 0, 1], \bar{\mathcal{Q}}^{a_2} = [0, 1; 0, 1]$*. A small calculation shows that greedy policy with respect to* $\hat{V}^*$ *is the optimal policy of* $\mathcal{M}$*, but the same does not hold for the normalized model. In particular, the optimal policy according to the normalized model selects action* a_1 *at state* 1*.*

Let us now return to discussing versions of linear action models. Although linear action models are powerful, it may be difficult to compute a Bellman-optimal policy in a linear action model. The idea of factored linear models is similar except that here the state space is unchanged; the "abstraction" happens because the transition kernel is written in a factored form: The map ψ extracts the features of state-action pairs, while the "features" of the sets one may arrive at are extracted by ξ. An interesting special case is when ξ takes the form

$$\xi(U) = \int_U f(x')\mu(dx'), \tag{4.2}$$

where μ is a signed measure over $\mathcal{X}$ and $f : \mathcal{X} \to \mathbb{R}^d$ is some fixed measurable function. A concrete example is when

$$\xi(U) = \sum_{i=1}^{n} f(x'_i)\mathbb{I}_{x'_i \in U}. \tag{4.3}$$

for some $x'_i \in \mathcal{X}$, $1 \le i \le n$. Then $\mathcal{Q}^a(U|x) = \sum_{i=1}^n f(x'_i)^\top \psi(x,a)\mathbb{I}_{x'_i \in U}$, and for $v \in L^1(Q^a)$,

$$\int v(y)Q^a(dy|x) = \sum_{i=1}^{n} f(x'_i)^\top \psi(x,a)v(x'_i).$$

In this case, under some additional conditions the optimal policy can be computed efficiently. Indeed, if $\hat{V}^*$ denotes the optimal value function for the pseudo-MDP, from the Bellman optimality equation, we get

$$\begin{aligned}\hat{V}^*(x) &= \max_{a\in\mathcal{A}} g^a(x) + \gamma \left(\sum_{i=1}^{n} \hat{V}^*(x_i')f(x_i')\right)^\top \psi(x,a)\\ &= \max_{a\in\mathcal{A}} (\hat{T}^a\hat{V}^*)(x),\end{aligned}$$

where the last equation defines the operators $\hat{T}^a$. By this equation, knowing $\hat{V}^*$ at states in $\mathcal{X}' = x_1', x_2, \ldots, x_n'$ suffices to compute an optimal action of $\mathcal{N}$ at any state $x \in \mathcal{X}$. The Bellman optimality equation will be guaranteed to have a solution if $\hat{T}^a$ is a contraction in the $\|\cdot\|_\infty$-norm, which holds if $|\sum_{i=1}^n f(x')^\top\psi(x,a)| \le 1$ for any $(x,a) \in \mathcal{X}\times\mathcal{A}$. Using the Bellman optimality equation again, we see that $\hat{V}^*|_{X'}$ is the optimal value function of the *finite* pseudo-MDP

$$(\mathcal{X}', (\mathcal{Q}^a|_{\mathcal{X}'\times 2^{\mathcal{X}'}})_{a\in\mathcal{A}}, (g^a|_{\mathcal{X}'})_{a\in\mathcal{A}}) \tag{4.4}$$

and, as such, it can be found, e.g., by either value iteration or linear programming.

Factored linear models which satisfy equation 4.3 will be called *finitely supported factored linear models* (FSFLMs).

We now show that FSFLMs generalize existing models. First, we show that finite models are also FSFLMs. Indeed, given a finite model

$$\mathcal{N} = (\mathcal{S}, \mathcal{A}, (\mathcal{Q}^a)_{a\in\mathcal{A}}, (g^a)_{a\in\mathcal{A}})$$

with $\mathcal{S} = \{1,\ldots,d\}$ and a surjective map $\phi : \mathcal{X} \to \mathcal{S}$, pick $\mu(\cdot|i)$ so that for each $i \in \mathcal{S}$, $\mu(\cdot|i)$ is a probability distribution supported on $\phi^{-1}(i) = \{x \in \mathcal{X} \mid \phi(x) = i\}$. Define the probability kernels $(\hat{\mathcal{P}}^a)_{a\in\mathcal{A}}$ by $\hat{\mathcal{P}}^a(dx'|x) = \sum_{j\in\mathcal{S}} \mu(dx'|j)\mathcal{Q}^a(j|\phi(x))$. By choosing $\xi_i(dx) = \mu(dx|i)$, $\psi_i(x,a) = \mathcal{Q}^a(i|\phi(x))$, $1 \le i \le d$, we see that $\hat{\mathcal{P}}^a(U|x) = \xi(U)^\top\psi(x,a)$, thus a finite model gives rise to a factored model. If we choose $\mu(\cdot|j)$ to be atomic, say, supported at x_j', we can write $\xi_i(U) = \mathbb{I}_{x_i'\in U}$ and $\xi(U) = \sum_{i=1}^n e_i \mathbb{I}_{x_i'\in U}$, thus we get $\mathcal{P}^a$ defines a FSFLM.

Let us now argue that the kernel-based model of Ormoneit and Sen (2002) also gives rise to an FSFLM. Let $\mathcal{Z} = \mathcal{X}\times\mathcal{A}$, $(z_1, x_1'), \ldots, (z_n, x_n') \in \mathcal{Z}\times\mathcal{X}$,

$\nu(\cdot|x)$ a probability kernel over $\mathcal{X}$, and $k : \mathcal{Z} \times \mathcal{Z} \to [0, \infty)$ a function such that $\sum_{j=1}^{n} k(\,(x,a), z_j) > 0$ for any $(x,a) \in \mathcal{Z}$. Define $\xi_i(dx') = \nu(dx'|x'_i)$ and $\psi_i(x,a) = k(\,(x,a), z_i)/\sum_{j=1}^{n} k(\,(x,a), z_j)$. The resulting factored linear model generalizes the *kernel-based model* of Ormoneit and Sen (2002) who chooses $\nu(dx'|x'_i) = \delta_{x'_i}(dx')$. In this case, ξ can be put in the form (4.3) with $f_i(x') = \mathbb{I}_{\{x'=x'_i\}}$ and $\mu(dx') = \sum_{j=1}^{n} \delta_{x'_j}(dx')$, and we get a FSFLM. The model of Grünewälder et al. (2012) that embeds the transition kernels into reproducing kernel Hilbert spaces can also be seen to be an FSFLM, though to allow this we need to replace the range of ξ and ψ with some Hilbert space.

4.4 A Generic Error Bound

From Section 4.2, we know that any policy in a pseudo-MDP can be used for action selection in the source MDP through the pull-back operation. In this section, we derive a bound on how well the pullback of a near-Bellman optimal policy of a pseudo-MDP $\mathcal{N}$ will perform in the source MDP $\mathcal{M}$. Note that we will not assume contraction for Bellman operator of the pseudo-MDP. Before stating this result, we need some definitions. Given any measurable function v over $\mathcal{S}$, we let V_v denote the function over $\mathcal{X}$ defined by

$$V_v(x) = v(\phi(x)),$$

and V_v is called the *pullback* of v. We also introduce a left inverse $l : \Pi_{\mathcal{M}} \to \Pi_{\mathcal{N}}$ to L. We call l a *pushforward map*. Thus, $l(L(\beta)) = \beta$ holds for any $\beta \in \Pi_{\mathcal{N}}$. Note that to ensure that L has a left inverse, ϕ must be surjective:

Assumption A1 ϕ is surjective.

When ϕ is surjective, it is easy to see that a left inverse of L indeed exists because for any $s \in \mathcal{S}$ all policies act uniformly for any state of $\phi^{-1}(s)$. Figure 4.2 illustrates the mappings, ϕ, l, and L.

The push-forward map is a theoretical construction in the sense that it is only used in characterizing the "power" of abstractions (it is not used algorithmically). This allows one to choose the best push-forward map that gives the tightest error bounds.

A push-forward and a feature map together give rise to the concept of approximate value functions:

Definition 2 (Approximate Value Function)**.** *Fix a push-forward map l and a feature map ϕ. Given a policy $\alpha \in \Pi_{\mathcal{M}}$, we call $v^{l(\alpha)}$ the value-function of α under l in $\mathcal{N}$. Further, we let $V_v^{\alpha} \doteq V_{v^{l(\alpha)}}$, i.e., the pullback of $v^{l(\alpha)}$, be the* $\mathcal{N}$-induced approximate value function *underlying policy α.*[2]

[2]In fact, in addition to $\mathcal{N}$, both l and ϕ influence V_v^{α}.

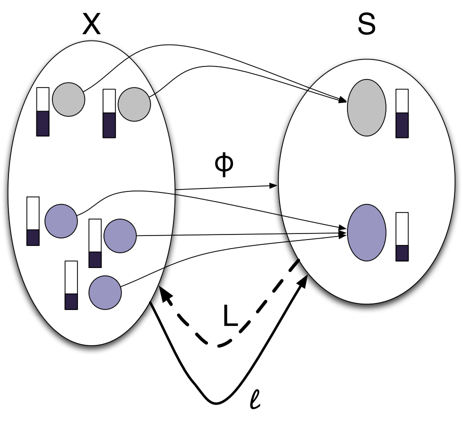

Figure 4.2: Illustration of ϕ, l, L using a MDP $\mathcal{M}$ with two actions. The coloring of the states in $\mathcal{X}$ and $\mathcal{S}$ identifies ϕ: ϕ keeps the colors of states. The bars next to states in $\mathcal{S}$ define a policy in $\Pi_{\mathcal{N}}$. This is pulled back to a policy of $\mathcal{M}$, by L. Note that identically colored states of $\mathcal{X}$ will be assigned the same probability distribution over the two actions. The mapping l takes a policy in $L(\Pi_{\mathcal{N}})$ and maps it to a policy of $\mathcal{N}$. As long as l mixes action-distribution of identically coloured states, l will be a left inverse of L.

Let $\mathcal{B}(\mathcal{X}) = (\mathcal{B}(\mathcal{X}), \|\cdot\|)$ be a normed subspace of $L^1(\mathcal{P})$: $\mathcal{B}(\mathcal{X}) = \{V : \mathcal{X} \to \mathbb{R} \mid V \in L^1(\mathcal{P}), \|V\| < \infty\}$. We use the norm $\|\cdot\|$ associated with $\mathcal{B}(\mathcal{X})$ to measure the magnitude of the errors introduced by $\mathcal{N}$: We call

$$\epsilon(\alpha) = \|V^\alpha - V_v^\alpha\| \tag{4.5}$$

the evaluation error of policy α induced by $\mathcal{N}$.

To compare policies we will use the expected total discounted reward where the initial state is selected from some fixed distribution, ρ. Given any $V : \mathcal{X} \to \mathbb{R}$, define $V_\rho = \int_{x\in\mathcal{X}} V(x)\rho(dx)$. Then $V_\rho^\alpha = \int_{x\in\mathcal{X}} V^\alpha(x)\rho(dx)$ gives the expected total discounted reward collected while following α assuming that the initial state is selected from ρ. Further, for a function $V \in L^1(\mathcal{P})$, define its $L^1(\rho)$-norm by $\|V\|_{L^1(\rho)} = \int |V(x)|\rho(dx)$ and let $K_\rho = \sup_{V\in\mathcal{B}(\mathcal{X})} \|V\|_{L^1(\rho)}/\|V\|$. We will denote by $\phi_*(\rho)$ the *push-forward of ρ under ϕ*. Thus, $\phi_*(\rho)$ is a probability measure on $\mathcal{S}$: it is the distribution of $\phi(X)$ where $X \sim \rho$.

4.4.1 An Error Bound

With this, we can present our first main result which bounds the suboptimality of the pullback of the pseudo-MDP's optimal policy:

Theorem 1. *Let* $\alpha^* \in \arg\max_{\alpha\in\Pi_{\mathcal{M}}} V^\alpha_\rho$, $\beta^* \in \arg\max_{\beta\in\Pi_{\mathcal{N}}} v^\beta_{\phi_*(\rho)}$ *and let* $\alpha^*_L = L(\beta^*)$ *be the pullback of* β^**. Then, under A1,*

$$V^{\alpha^*}_\rho - K_\rho(\epsilon(\alpha^*) + \epsilon(\alpha^*_L)) \le V^{\alpha^*_L}_\rho \le V^{\alpha^*}_\rho.$$

The theorem shows that the quality of the policy derived from an optimal policy of the pseudo-MDP is governed by the error induced by $\mathcal{N}$ on the value functions of policies α^*, α^*_L alone. Thus, it suggests that when considering the construction of $\mathcal{N}$, one should concentrate on the evaluation error of these two policies. *The result is remarkable because it suggests that the common objection against model learning—a good model must capture all the details of the world—has not theoretical basis.* Of course, the difficulty is that while β^* may be accessible (given $\mathcal{N}$), α^* is hardly available. Nevertheless, the result suggests an iterative approach towards constructing $\mathcal{N}$, which we will explore later.

The policy evaluation error defined in (4.5) depends on the norm chosen for the functions over $\mathcal{X}$. If one chooses the supremum norm, Theorem 1 immediately gives the following result:

Corollary 2. *Let* $\|\cdot\| = \|\cdot\|_\infty$ *in* (4.5)*. Then, under A1, for any optimal policy* α^* *of* $\mathcal{M}$ *and optimal policy* β^* *of* $\mathcal{N}$*,* $\|V^{\alpha^*_L} - V^*\|_\infty \le \epsilon(\alpha^*) + \epsilon(\alpha^*_L)$*, where* $\alpha^*_L = L(\beta^*)$.

It is remarkable that none of these results uses contraction arguments. Note that the definition of α^* and β^* in Theorem 1 is different from the definition used in this corollary. While here α^*, β^* are required to be optimal, in Theorem 1 they are optimal only in a weaker, average sense. Note that choosing the norm in (4.5) to be the supremum norm makes $K_\rho = 1$ for any distribution ρ (which is great), but can increase the values of $\epsilon(\alpha^*)$ and $\epsilon(\alpha^*_L)$. Hence, the norm that optimizes the bound may very well be different from the supremum norm.

Finally, before the proof let us discuss the computation of the "optimal" policies in $\mathcal{N}$ mentioned in these two results. In Theorem 1, $\beta^* = \arg\max_{\beta \in \Pi_{\mathcal{N}}} v^{\beta}_{\phi_*(\rho)}$. One can show that this policy can be obtained from the solution of a linear program (LP)

$$\phi_*(\rho)^\top v \to \min \quad \text{s.t.} \quad v \geq \tilde{T}_a v, \quad a \in \mathcal{A}$$

where

$$(\tilde{T}_a v)(s) = g^a(s) + \gamma \int Q^a(ds'|s)v(s').$$

When $(Q^a)_{a\in\mathcal{A}}$ are non-negative valued and $\phi_*(\rho)$ has full support, any solution of this LP can be shown to satisfy the Bellman-optimality equation. With the constraint on $(Q^a)_{a\in\mathcal{A}}$ dropped, not much can be said about the LP. With the positivity condition on $(Q^a)_{a\in\mathcal{A}}$, the optimal policy β^* of Corollary 2 (*i.e.*, the policy that satisfies $v^{\beta^*} \geq v^{\beta}$ for any $\beta \in \Pi_{\mathcal{N}}$) exists if the Bellman-optimality equation has a solution. Again with no condition on $(Q^a)_{a\in\mathcal{A}}$, this is not guaranteed.

4.4.2 Proof

We need the following definitions pertaining MDPs: For any $V \in L^1(\mathcal{P})$, define the operator $T^a : L^1(\mathcal{P}) \to L^1(\mathcal{P})$ by $(T^a V)(x) = f^a(x) + \gamma(\mathcal{P}^a V)(x)$, where $\mathcal{P}^a : L^1(\mathcal{P}) \to L^1(\mathcal{P})$ is the operator defined by $(\mathcal{P}^a V)(x) = \int_{x'} \mathcal{P}^a(dx'|x)V(x')$. For a policy α, define the operator $T^\alpha : L^1(\mathcal{P}) \to L^1(\mathcal{P})$ by,

$$(T^\alpha V)(x) = \sum_{a\in\mathcal{A}} \alpha(x,a)(T^a V)(x), \quad x \in \mathcal{X}.$$

Further, define the policy mapping $\Pi_\alpha : L^1(\mathcal{P})^{\mathcal{A}} \to L^1(\mathcal{P})$ by $(\Pi_\alpha[V_a]_{a\in\mathcal{A}})(x) = \sum_{a\in\mathcal{A}} \alpha(x,a)V^a(x)$, for $x \in \mathcal{X}$. Therefore, we can rewrite T^α as

$$T^\alpha V = \Pi_\alpha(f + \gamma \mathcal{P} V),$$

where $f = [f^a]_{a\in\mathcal{A}}$ and $\mathcal{P}V = [\mathcal{P}^a V]_{a\in\mathcal{A}}$, for any $V \in L^1(\mathcal{P})$. The operator T^α is called the policy evaluation operator of policy α. As it is well known, for any policy α, $T^\alpha V^\alpha = V^\alpha$ holds, *i.e.*, V^α is the fixed point of T^α. (This equation is also known as the Bellman equation for V^α). The optimal value function also satisfies a fixed point equation: $V^* = T^* V^*$, the Bellman optimality equation. The operator $T^* : L^1(\mathcal{P}) \to L^1(\mathcal{P})$ in this equation is called the Bellman optimality operator and is defined by $(T^*V)(x) = \max_{a\in\mathcal{A}}(T^a V)(x)$. Furthermore, as mentioned previously, any policy α that satisfies that for any state $x \in \mathcal{X}$, $\alpha(x,\cdot)$ is supported on $\operatorname{argmax}_{a\in\mathcal{A}}(T^a V^*)(x)$ is an optimal policy.

Recall that for any measurable function v over $\mathcal{S}$, we let V_v denote a function over $\mathcal{X}$ defined by $V_v(x) = v(\phi(x))$. Now, we also introduce g_L^a to denote the function over $\mathcal{X}$ defined via $g_L^a(x) = g^a(\phi(x))$.

Let $\mathcal{F}_\phi(\mathcal{X}) = \{V_v \,|\, v : \mathcal{S} \to \mathbb{R}\}$. Note that $\mathcal{F}_\phi(\mathcal{X})$ is a vector space. For $a \in \mathcal{A}$, $\alpha \in \Pi_{\mathcal{M}}$, we define the operators $\mathcal{Q}_L^a, \hat{T}^a : \mathcal{F}_\phi(\mathcal{X}) \to \mathcal{F}_\phi(\mathcal{X})$ in the following way: Given $x \in \mathcal{X}$, $a \in \mathcal{A}$, $V_v \in \mathcal{F}_\phi(\mathcal{X})$,

$$\begin{aligned}(\mathcal{Q}_L^a V_v)(x) &= (\mathcal{Q}^a v)(\phi(x)) = \int_{s'} \mathcal{Q}^a(ds'|\phi(x))v(s'),\\ (\hat{T}^a V_v)(x) &= g_L^a(x) + \gamma(\mathcal{Q}_L^a V_v)(x)\,.\end{aligned}$$

Note that these are well-defined because for any $V \in \mathcal{F}_\phi(\mathcal{X})$ there is a unique function $v : \mathcal{S} \to \mathbb{R}$ such that $V = V_v$. We further define $\hat{T}^\alpha : \mathcal{F}_\phi(\mathcal{X}) \to \mathcal{F}_\phi(\mathcal{X})$ as follows: For $V \in \mathcal{F}_\phi(\mathcal{X})$, $\hat{T}^\alpha V = \Pi_\alpha \hat{T} V$, where $\hat{T} : \mathcal{F}_\phi(\mathcal{X}) \to \mathcal{F}_\phi(\mathcal{X})^{\mathcal{A}}$ is given by $(\hat{T} V_v) = [\hat{T}^a V_v]_{a \in \mathcal{A}}$.

Recall the definition of approximate value functions: For policy $\alpha \in \Pi_M$, the ($\mathcal{N}$-induced) approximate value function of α is $V_v^\alpha \doteq V_{v^{l(\alpha)}}$. For all $\beta \in \Pi_{\mathcal{N}}$, because $l(L(\beta)) = \beta$, $V_{v^\beta} = V_{v^{l(L(\beta))}}$, *i.e.*, V_{v^β} is the approximate value function of policy $L(\beta)$. Moreover, V_{v^β} is the fixed point of operator $\hat{T}^{L(\beta)}$:

Proposition 3. *For any policy $\beta \in \Pi_{\mathcal{N}}$, we have*

$$V_{v^\beta} = \hat{T}^{L(\beta)} V_{v^\beta}.$$

Proof. First, note that a standard contraction argument shows that v^β satisfies the Bellman equations: $v^\beta(s) = \sum_a \beta(s,a)(g^a(s) + \gamma \int_{s'} \mathcal{Q}^a(ds'|s) v^\beta(s'))$, $s \in \mathcal{S}$. Take any $x \in \mathcal{X}$. Then,

$$\begin{aligned}
(\hat{T}^{L(\beta)} V_{v^\beta})(x) &= \sum_{a \in \mathcal{A}} L(\beta)(x,a) \left(\hat{T}^a V_{v^\beta} \right)(x) \\
&= \sum_{a \in \mathcal{A}} \beta(\phi(x), a) \left(g_L^a(x) + \gamma (\mathcal{Q}_L^a V_{v^\beta})(x) \right) \\
&= \sum_{a \in \mathcal{A}} \beta(\phi(x), a) \left(g^a(\phi(x)) + \gamma \int_{s'} \mathcal{Q}^a(ds'|\phi(x)) v^\beta(s') \right) \\
&= v^\beta(\phi(x)) = V_{v^\beta}(x),
\end{aligned}$$

where the first equality holds due to the definition of $\hat{T}^{L(\beta)}$, the second holds due to the definition of $\hat{T}^a$ and the pullback of policy β, the third holds due to the definition of g_L^a and $\mathcal{Q}_L^a$, the fourth due to the Bellman equation for v^β, and the last holds due to the pullback of value functions. ☐

Corollary 4. *For any policy $\alpha \in \Pi_{\mathcal{M}}$,*

$$V_v^\alpha = \hat{T}^{\alpha'} V_v^\alpha, \textit{ where } \alpha' \doteq L(l(\alpha)). \tag{4.6}$$

Proof. Apply Proposition 3 to $\beta = l(\alpha)$. ☐

Let $\Pi_{\mathcal{M}}^L = \{L(\beta) \,|\, \beta \in \Pi_{\mathcal{N}}\}$ be the range space of L. Note that $L \circ l$ is an identity over $\Pi_{\mathcal{M}}^L$ but not over $\Pi_{\mathcal{M}} \setminus \Pi_{\mathcal{M}}^L$. For a policy $\alpha \in \Pi_{\mathcal{M}} \setminus \Pi_{\mathcal{M}}^L$, the operator $\hat{T}^{\alpha'}$ may be different from $\hat{T}^\alpha$ and as a result, V_v^α may be different from V^α.

A simple calculation shows that the following holds.

Proposition 5. *For any $\beta \in \Pi_{\mathcal{N}}$, $V_{v,\rho}^{L(\beta)} = v_{\phi_*(\rho)}^{\beta}$.*

Proof. We have

$$\begin{aligned}
V_{v,\rho}^{L(\beta)} &= \int V_v^{L(\beta)}(x)\,\rho(dx) && \text{(definition of } V_{v,\rho}^{L(\beta)}\text{)}\\
&= \int v^{l(L(\beta))}(\phi(x))\,\rho(dx) && \text{(definition of } V_v^{L(\beta)}\text{)}\\
&= \int v^{\beta}(\phi(x))\,\rho(dx) && \text{(because } l(L(\beta)) = \beta\text{)}\\
&= \int v^{\beta}(s)\,\phi_*(\rho)(ds)\,. && \text{(definition of } \phi_*(\rho)\text{)}
\end{aligned}$$

□

Recalling the definition of K_ρ, we immediately get the following result:

Proposition 6. *For any $\alpha \in \Pi_{\mathcal{M}}$, $|V_\rho^\alpha - V_{v,\rho}^\alpha| \le K_\rho \epsilon(\alpha)$.*

Proof. We have

$$\begin{aligned}
|V_\rho^\alpha - V_{v,\rho}^\alpha| &= \left|\int (V^\alpha(x) - V_v^\alpha(x))\rho(dx)\right|\\
&\le \int |(V^\alpha(x) - V_v^\alpha(x))|\,\rho(dx) && \text{(triangle inequality)}\\
&= \|V_\alpha - V_v^\alpha\|_{L^1(\rho)} && \text{(definition of } \|\cdot\|_{L^1(\rho)}\text{)}\\
&\le K_\rho \|V_\alpha - V_v^\alpha\| && \text{(definition of } K_\rho\text{)}\\
&= K_\rho \epsilon(\alpha). && \text{(definition of } \epsilon(\alpha)\text{)}
\end{aligned}$$

□

For a policy β and distribution ρ' over $\mathcal{S}$, we define $v_{\rho'}^{\beta} \doteq \int v^\beta(s)\rho'(ds)$ as the "expected" total discounted reward of β in $\mathcal{N}$ when the distribution of the initial state is ρ'. Given a measurable function $v : \mathcal{S} \to \mathbb{R}$, we also let $V_{v,\rho} = \int_{x\in\mathcal{X}} V_v(x)\rho(x)dx$ and given a policy $\alpha \in \Pi_{\mathcal{M}}$, we define $V_{v,\rho}^{\alpha} = \int V_{v^{l(\alpha)}} d\rho(x)$.

With this, we are ready to prove Theorem 1.

Proof. By the definition of α^*, $V_\rho^{\alpha_L^*} \le V_\rho^{\alpha^*}$, thus it remains to prove the lower bound on $V_\rho^{\alpha_L^*}$.

We have

$$V_{v,\rho}^{\alpha_L^*} = v_{\phi_*(\rho)}^{\beta^*} \geq \int v^{l(\alpha^*)}(s)\phi_*(\rho)(ds) = V_{v,\rho}^{\alpha^*},$$

where the first equality is due to Proposition 5, the inequality is due to the definition of β^*, and the last equality is due to the definition of $V_{v,\rho}^{\alpha^*}$ and Proposition 5. Therefore,

$$V_{\rho}^{\alpha_L^*} = V_{v,\rho}^{\alpha_L^*} + \left(V_{\rho}^{\alpha_L^*} - V_{v,\rho}^{\alpha_L^*}\right) \geq V_{v,\rho}^{\alpha^*} + I_1,$$

where we define $I_1 = V_{\rho}^{\alpha_L^*} - V_{v,\rho}^{\alpha_L^*}$. Now, $V_{v,\rho}^{\alpha^*} = V_{\rho}^{\alpha^*} + \left(V_{v,\rho}^{\alpha^*} - V_{\rho}^{\alpha^*}\right)$. Note the last term is just the policy evaluation error of α^*. Thus $V_{\rho}^{\alpha^*} - V_{v,\rho}^{\alpha^*} \leq K_\rho \epsilon(\alpha^*)$, due to Proposition 6, and similarly $I_1 \geq -K_\rho \epsilon(\alpha_L^*)$. Thus

$$V_{\rho}^{\alpha^*} - K_\rho(\epsilon(\alpha^*) + \epsilon(\alpha_L^*)) \leq V_{\rho}^{\alpha_L^*},$$

completing the proof. □

Finally, we provide the proof of Corollary 2.

Proof. According to Theorem 1, for any ρ, $0 \leq V_{\rho}^{\alpha^*} - V_{\rho}^{\alpha_L^*} \leq K_\rho(\epsilon(\alpha^*) + \epsilon(\alpha_L^*))$. Note that α^*, α_L^* are both independent of ρ. Given a point $x_0 \in \mathcal{X}$, we can pick $\rho_{x_0} = \delta_{x_0}(\cdot)$ to select this point (thus, by definition $\rho_{x_0}(U) = \mathbb{I}_{\{x_0 \in U\}}$ for any measurable $U \subset \mathcal{X}$ and so $\int f(x)\rho_{x_0}(dx) = f(x_0)$ for any measurable function f). Then for any $\alpha \in \Pi_{\mathcal{M}}$, $V_{\rho_{x_0}}^{\alpha} = \int V^\alpha(x)\delta_{x_0}(dx) = V^\alpha(x_0)$. Thus

$$0 \leq V^{\alpha^*}(x_0) - V^{\alpha_L^*}(x_0) \leq K_{\rho_{x_0}}(\epsilon(\alpha^*) + \epsilon(\alpha_L^*)).$$

According to the definition of the $L^1(\rho)$ norm, for any $V \in \mathcal{B}(\mathcal{X})$ and any $x_0 \in \mathcal{X}$,

$$\|V\|_{L^1(\rho_{x_0})} = \int |V(x)|\rho_{x_0}(dx) = |V(x_0)| \leq \|V\|_\infty,$$

which means that $K_{\rho_{x_0}} = 1$ for all $x_0 \in \mathcal{X}$. Therefore, for any $x_0 \in \mathcal{X}$, $0 \leq V^*(x_0) - V^{\alpha_L^*}(x_0) \leq \epsilon(\alpha^*) + \epsilon(\alpha_L^*)$, and $\|V^{\alpha_L^*} - V^*\|_\infty \leq \epsilon(\alpha^*) + \epsilon(\alpha_L^*)$. □

4.5 State Space Preserving Abstraction

When the feature map $\phi : \mathcal{X} \to \mathcal{S}$ is injective (and thus invertible), the generic bound of the previous section gives rise to a bound of a particularly appealing form. *When ϕ is a bijection, we can identify $\mathcal{S}$ with $\mathcal{X}$ without loss of generality and choose ϕ to be the identity map, an assumption that we will indeed make in this section.* [3] The factored linear action model considered in the previous section gives a useful example when $\mathcal{S} = \mathcal{X}$. In general, when $\mathcal{S} = \mathcal{X}$, the approximation happens implicitly using a transition kernel that introduces additional structure (invariances).

For simplicity, we also assume that $g^a \equiv f^a$, *i.e.*, the rewards are not approximated (the extension of the results to the general case is trivial). In summary, the pseudo-MDP considered in this section takes the form

$$\mathcal{N} = (\mathcal{X}, \mathcal{A}, (\hat{\mathcal{P}}^a)_{a\in\mathcal{A}}, (f^a)_{a\in\mathcal{A}})$$

(we replace $\mathcal{Q}^a$ by $\hat{\mathcal{P}}^a$ to emphasize that the approximate kernels are now over the state space of the source MDP).

Define $\|\hat{\mathcal{P}}\|_{1,\infty} = \sup_{x,a} \|\hat{\mathcal{P}}^a(\cdot|x)\|_1$. When $\|\hat{\mathcal{P}}\|_{1,\infty} \le 1$, corollary 2 together with a standard contraction argument leads to the following result:

Theorem 7. *Let*

$$\mathcal{N} = (\mathcal{X}, \mathcal{A}, (\hat{\mathcal{P}}^a)_{a\in\mathcal{A}}, (f^a)_{a\in\mathcal{A}})$$

be a pseudo-MDP such that $\|\hat{\mathcal{P}}\|_{1,\infty} \le 1$. Then, for any optimal policy $\hat{\alpha}^*$ of $\mathcal{N}$,

$$\|V^{\hat{\alpha}^*} - V^*\|_\infty \le \tfrac{2\gamma}{(1-\gamma)^2} \min\{ \|(\hat{\mathcal{P}} - \mathcal{P})V^*\|_\infty, \|(\hat{\mathcal{P}} - \mathcal{P})v^*\|_\infty\},$$

where $v^* = V_v^{\hat{\alpha}^*}$.

Note that v^* may be different from the optimal solution of the Bellman-optimality equation in $\mathcal{N}$. When $\hat{\mathcal{P}}^a(\cdot|x)$ is non-negative valued, they are guaranteed to be the same.

[3]An injective mapping is a function that preserves distinctness. Every element of the function's codomain has at most one element of its domain. Injective mapping is also called one-to-one mapping. A bijection or surjection mapping, on the other hand, uniquely maps all elements in both domains to each other.

Again, we see that it suffices if $\hat{\mathcal{P}}$ is a good approximation to $\mathcal{P}$ at V^*. The other term, $\|(\hat{\mathcal{P}} - \mathcal{P})\hat{v}^*\|_\infty$, is more helpful. We can estimate it and reduce it by feature learning. Since V^* is unknown, in practice one may choose a normed vector-space $\mathcal{F}$ of functions over $\mathcal{X}$ and construct $\hat{\mathcal{P}}$ such that it is a good approximation to $\mathcal{P}$ over $\mathcal{F}$ in the sense that $\epsilon(\mathcal{F}) = \sup_{V\in\mathcal{F},\|V\|_\mathcal{F}=1} \|(\hat{\mathcal{P}} - \mathcal{P})V\|_\infty$ is small (here, $\|\cdot\|_\mathcal{F}$ denotes the norm that comes with $\mathcal{F}$). Can this approach succeed? Let $\Delta\mathcal{P} = \hat{\mathcal{P}} - \mathcal{P}$. Then, for any $V \in \mathcal{F}$, $\|\Delta\mathcal{P}V^*\|_\infty \le \|(\hat{\mathcal{P}} - \mathcal{P})(V^* - V)\|_\infty + \|\Delta\mathcal{P}V\|_\infty \le 2\|V^* - V\|_\infty + \epsilon(\mathcal{F})\|V\|_\mathcal{F}$. Taking the infimum over $V \in \mathcal{F}$, we get the following result:

Corollary 8. *Under the same conditions as in theorem 7, for any optimal policy $\hat{\alpha}^*$ of $\mathcal{N}$, $\|V^{\hat{\alpha}^*} - V^*\|_\infty \le \frac{2\gamma}{(1-\gamma)^2} \inf_{V\in\mathcal{F}} \{2\|V^* - V\|_\infty + \epsilon(\mathcal{F})\|V\|_\mathcal{F}\}$.*

Thus, the approach will be successful as long as our bet that V^* is close to $\mathcal{F}$ is correct and in particular if the L^∞-projection of V^* to $\mathcal{F}$ has a small $\mathcal{F}$-norm. Note that corollary 8 can be viewed as a generalization/specialization of Theorem 3.2 of Grünewälder et al. (2012).[4] The assumption $\|\hat{\mathcal{P}}\|_{1,\infty} \le 1$ can be relaxed.

In the next section, we show how to learn models such that the assumption is satisfied.

[4]The specialization comes from the fact that while Theorem 3.2 considers all kind of approximations, we concentrate on the approximation induced by the approximate model as we find the approach that separates this from other approximation terms much cleaner.

4.5.1 Proof

We start with a general contraction result (see, e.g., Lemma 5.16 of Szepesvári 2001 for a somewhat weaker result of this type):

Lemma 9. *Let $\mathcal{V} = (\mathcal{V}, \|\cdot\|)$ be a Banach space, $\hat{T} : \mathcal{V} \to \mathcal{V}$ a γ-contraction on $\mathcal{V}$ and $T : \mathcal{V} \to \mathcal{V}$ a continuous operator on $\mathcal{V}$. Assume that the iteration $V_{n+1} = TV_n$ converges to V in $\|\cdot\|$. Then*

$$\|V - \hat{V}\| \le \frac{\|(\hat{T} - T)V\|}{1-\gamma},$$

where $\hat{V}$ is the fixed point of $\hat{T}$.

Proof. We have

$$\begin{aligned}\|V_{n+1} - \hat{V}_{n+1}\| &= \|TV_n - \hat{T}\hat{V}_n\| \\ &= \|TV_n - \hat{T}V_n + \hat{T}V_n - \hat{T}\hat{V}_n\| \\ &\le \|TV_n - \hat{T}V_n\| + \|\hat{T}V_n - \hat{T}\hat{V}_n\| \\ &\le \|TV_n - \hat{T}V_n\| + \gamma\|V_n - \hat{V}_n\|.\end{aligned}$$

Taking the limit of both sides as $n \to \infty$ and reordering gives the claimed result.

□

Recall that $\Delta\mathcal{P} = \mathcal{P} - \hat{\mathcal{P}}$, $\mathcal{S} = \mathcal{X}$ and ϕ is the identity, by assumption. The next lemma bounds the policy evaluation error induced by $\mathcal{N}$:

Lemma 10. *Assume that* $\sup_{x,a} \|\hat{\mathcal{P}}^a(\cdot|x)\|_1 \le 1$. *Then, for any policy* $\alpha \in \Pi_{\mathcal{M}} = \Pi_{\mathcal{N}}$, *it holds that*

$$\|V^\alpha - V_v^\alpha\|_\infty \le \frac{\gamma}{1-\gamma} \min(\|\Delta\mathcal{P}\, V^\alpha\|_\infty, \|\Delta\mathcal{P}\, V_v^\alpha\|_\infty)\,.$$

Proof. Since ϕ is the identity, $l(\alpha) = \alpha$. Hence,

$$V_v^\alpha = V_{v^\alpha} = v^\alpha\,.$$

Thus, it remains to bound $\|V^\alpha - v^\alpha\|_\infty$. For this, we use lemma 9 with $\mathcal{V} = (\mathcal{B}(\mathcal{X}), \|\cdot\|_\infty)$, $T = T^\alpha$ and $\hat{T} = \hat{T}^\alpha \doteq \Pi_\alpha(f + \gamma\hat{\mathcal{P}})$.[5] The conditions of lemma 9

[5]To be precise, we need to use the restrictions of these operators to $\mathcal{B}(\mathcal{X})$.

can be seen to be satisfied. Furthermore, V of the lemma is equal to V^α, the fixed point of T^α since T^α is a γ-contraction. Hence,

$$\|V^\alpha - v^\alpha\|_\infty \le \frac{1}{1-\gamma}\|(\hat{T}^\alpha - \hat{T}^\alpha)V^\alpha\|_\infty.$$

Now, $T^\alpha - \hat{T}^\alpha = \Pi_\alpha(f + \gamma\mathcal{P}) - \Pi_\alpha(f + \gamma\hat{\mathcal{P}}) = \gamma\Pi_\alpha(\mathcal{P} - \hat{\mathcal{P}})$ and the first part of the result follows since $\|(T^\alpha - \hat{T}^\alpha)V^\alpha\|_\infty = \gamma\|\Pi_\alpha\Delta\mathcal{P}V^\alpha\|_\infty \le \gamma\|\Delta\mathcal{P}V^\alpha\|_\infty$. The second part follows similarly, just reverse the role of T^α and $\hat{T}^\alpha$ in the application of lemma 9. □

We now show a similar bound on the difference between $V^* - v^*$ (here, v^* is the optimal value function in $\mathcal{N}$). For proving this result, the following notation will be useful: Define the "max" operator $M : \mathbb{R}^{\mathcal{X}\times\mathcal{A}} \to \mathbb{R}^{\mathcal{X}}$ by $(MQ)(x) = \max_{a\in\mathcal{A}} Q(x,a)$ given $Q \in \mathbb{R}^{\mathcal{X}\times\mathcal{A}}$ and $x \in \mathcal{X}$. With this definition, the Bellman optimality operator $T^* : L^1(\mathcal{P}) \to L^1(\mathcal{P})$ underlying $\mathcal{M}$ is given by

$$T^*V = M(f + \gamma\mathcal{P}V)\,, \qquad V \in L^1(\mathcal{P})\,.$$

Recall that T^* is a γ-contraction w.r.t. $\|\cdot\|_\infty$ and the fixed-point of T^* is V^*, the optimal state value function of $\mathcal{M}$. Now define the Bellman optimality operator $\hat{T}^* : L^1(\hat{\mathcal{P}}) \to L^1(\hat{\mathcal{P}})$ based on $\mathcal{N}$:

$$\hat{T}^*V = M(f + \gamma\hat{\mathcal{P}}V), \qquad V \in L^1(\hat{\mathcal{P}})\,.$$

When $\sup_{x,a}\|\hat{\mathcal{P}}^a(\cdot|x)\|_1 \le 1$, $\hat{T}^*$ is also a γ-contraction and its unique fixed point is v^*.

Lemma 11. *Assume* $\sup_{x,a}\|\hat{\mathcal{P}}^a(\cdot|x)\|_1 \le 1$. *Then,*

$$\|V^* - v^*\|_\infty \le \frac{\gamma}{1-\gamma}\min\{\|\Delta\mathcal{P}\,V^*\|_\infty, \|\Delta\mathcal{P}\,v^*\|_\infty\}\,.$$

Proof. To bound $\|V^* - v^*\|_\infty$ we use lemma 9. The conditions of the Lemma are satisfied with $\mathcal{V} = (\mathcal{B}(\mathcal{X}), \|\cdot\|_\infty)$, $T = T^*$, $\hat{T} = \hat{T}^*$. Further, since T^* is also a contraction, V of the Lemma equals V^*. Hence,

$$\|V^* - v^*\|_\infty \le \frac{1}{1-\gamma}\|(\hat{T}^* - T^*)V^*\|_\infty\,.$$

Now,

$$\|(\hat{T}^* - T^*)V^*\|_\infty = \|M(f + \gamma\hat{\mathcal{P}}V^*) - M(f + \gamma\mathcal{P}V^*)\|_\infty \le \gamma\|(\hat{\mathcal{P}} - \mathcal{P})V^*\|_\infty,$$

where the last inequality follows since

$$|\sup_{w\in W} f_1(w) - \sup_{w\in W} f_2(w)| \le \sup_{w\in W} |f_1(w) - f_2(w)|$$

holds for any two functions $f_1, f_2 : W \to \mathbb{R}$. Putting together the inequalities obtained we prove $\|V^* - v^*\|_\infty \le \frac{\gamma}{1-\gamma}\|\Delta\mathcal{P}V^*\|_\infty$.

The above proof also holds if we "exchange" the original MDP with the puseudo-MDP. Therefore, $\|V^* - v^*\|_\infty \le \frac{\gamma}{1-\gamma}\|\Delta\mathcal{P}v^*\|_\infty$.

□

We are now ready to prove theorem 7.

Proof. According to corollary 2, it suffices to bound $\epsilon(\alpha^*) + \epsilon(\hat{\alpha}^*)$. We have

$$\begin{aligned}\epsilon(\alpha^*) + \epsilon(\hat{\alpha}^*) &= \|V^{\alpha^*} - V_v^{\alpha^*}\|_\infty + \|V^{\hat{\alpha}^*} - V_v^{\hat{\alpha}^*}\|_\infty \\ &\le \frac{\gamma}{1-\gamma}(\|\Delta\mathcal{P}V^*\|_\infty + \|\Delta\mathcal{P}V_v^{\hat{\alpha}^*}\|_\infty) \\ &= \frac{\gamma}{1-\gamma}(\|\Delta\mathcal{P}V^*\|_\infty + \|\Delta\mathcal{P}v^*\|_\infty),\end{aligned}$$

where the second line is due to lemma 10, and the third line is due to $V_v^{\hat{\alpha}^*} = v^*$. Then, thanks to lemma 11,

$$\begin{aligned}\|\Delta\mathcal{P}v^*\|_\infty &\le \|\Delta\mathcal{P}(v^* - V^*)\|_\infty + \|\Delta\mathcal{P}V^*\|_\infty \\ &\le 2\|v^* - V^*\|_\infty + \|\Delta\mathcal{P}V^*\|_\infty \\ &\le \frac{2\gamma}{1-\gamma}\|\Delta\mathcal{P}V^*\|_\infty + \|\Delta\mathcal{P}V^*\|_\infty \\ &= \frac{1+\gamma}{1-\gamma}\|\Delta\mathcal{P}V^*\|_\infty.\end{aligned}$$

Thus $\epsilon(\alpha^*) + \epsilon(\hat{\alpha}^*) \le \frac{2\gamma}{(1-\gamma)^2}\|\Delta\mathcal{P}V^*\|_\infty$, finishing the proof for the first part.

Now for the second part, we have

$$\begin{aligned}\|\Delta\mathcal{P}V^*\|_\infty &\le \|\Delta\mathcal{P}(v^* - V^*)\|_\infty + \|\Delta\mathcal{P}v^*\|_\infty \\ &\le \frac{2\gamma}{1-\gamma}\|\Delta\mathcal{P}v^*\|_\infty + \|\Delta\mathcal{P}v^*\|_\infty \\ &= \frac{1+\gamma}{1-\gamma}\|\Delta\mathcal{P}v^*\|_\infty.\end{aligned}$$

where the second inequality is from Lemma 11.

□

4.6 Learning Factored Linear Models

In this section we propose two approaches of learning factored linear models including a least-squares approach and a constrained optimization approach. We then give the procedure of finding the fixed point of the Bellman optimality operator of the resulting pseudo-MDPs. Finally we propose a feature learning method.

4.6.1 The Least-Squares Model

In this section we show how factored linear models arise from a least-squares approach, essentially reproducing the model of Grünewälder et al. (2012) in a finite-dimensional setting from simple first principles (thus, hopefully catching the interest of readers who may shy away from the infinite dimensional setting consider by (Grünewälder et al., 2012)). The factored linear model that arises will be the basis of the feature learning method proposed in the next section.

As before, we will denote $\mathcal{Z} = \mathcal{X} \times \mathcal{A}$. Choose $V \in L^1(\mathcal{P})$ and suppose that we are interested in estimating the function $(x, a) \mapsto \int \mathcal{P}^a(dx'|x)V(x')$ where $(x, a) \in \mathcal{Z}$. Let $Z = (X, A)$ be a random state-action pair sampled from a distribution with full support over $\mathcal{Z}$ and $X' \sim P^A(\cdot|X)$. Then, $\int \mathcal{P}^a(dx'|x)V(x') = \mathbb{E}[V(X')|Z = (x, a)]$. Assume that we are given a mapping $\psi : \mathcal{X} \times \mathcal{A} \to \mathbb{R}^d$ to extract features based on state-action pairs and our goal is to find the best linear estimator $z \mapsto u^\top \psi(z)$ based on ψ of the function $z \mapsto \mathbb{E}[V(X')|Z = z]$. The parameter vector of the estimator that minimizes the expected squared error is $u^*(V) \in \arg\min_{u \in \mathbb{R}^d} \mathbb{E}[\big(V(X') - u^\top \psi(Z)\big)^2]$. A simple calculation shows that $u^*(V) = \mathbb{E}[\psi(Z)\psi(Z)^\top]^\dagger \mathbb{E}[\psi(Z)V(X')]$, where $M^\dagger$ denotes the pseudo-inverse of matrix M.

In practice, $u^*(V)$ is approximated based on a finite dataset, $(\langle z_i, x'_i \rangle, i = 1, \ldots, n)$. Defining $u_n(V) = \left(\Psi^\top \Psi\right)^\dagger \Psi^\top \bar{V}$, where $\Psi = [\psi(z_i)^\top] \in \mathbb{R}^{n \times d}$ and $\bar{V}_i = V(x'_i)$, $1 \le i \le n$, $u_n(V)$ optimizes the squared prediction error of $u^\top \psi(z)$ computed over $(\langle z_i, x'_i \rangle, i = 1, \ldots, n)$. Introducing $F = \Psi \left(\Psi^\top \Psi\right)^\dagger$ and letting $F_{i:}$ denote the ith row of F (i.e., $F_{i:} \in \mathbb{R}^{1\times d}$), we calculate

$$\begin{aligned} u_n(V)^\top \psi(x,a) &= \bar{V}^\top F \psi(x,a) \\ &= \int \sum_{i=1}^{n} V(x')\delta_{x'_i}(dx') F_{i:}\psi(x,a)\,. \end{aligned} \tag{4.7}$$

Thus with $\xi(dx') = \sum_{i=1}^{n} \delta_{x'_i}(dx') F_{i:}^\top$, if $\hat{\mathcal{P}}^a(dx'|x) = \xi(dx')^\top \psi(x,a)$ then given ψ, $(x,a) \mapsto \int \hat{\mathcal{P}}^a(dx'|x)V(x')$ is the best linear least-squares estimator of $(x,a) \mapsto \int \mathcal{P}^a(dx'|x)V(x')$ *for any* $V \in L^1(\mathcal{P})$. In this sense, $(\hat{\mathcal{P}}^a)_{a\in\mathcal{A}}$ is the "best" estimate of $(\mathcal{P}^a)_{a\in\mathcal{A}}$.

Since $(\hat{\mathcal{P}}^a)_{a\in\mathcal{A}}$ is of the form (4.3) with $f(x') = \mathbb{I}_{\{x'=x'_i\}} F_{i:}^\top$, the discussion after (4.3) applies: The Bellman optimality equation of the approximate model can be solved with finite resources up to any desired accuracy.

For computational purposes, it is worthwhile to define $\pi : \mathcal{Z} \to \mathbb{R}^n$ using $\pi_i(x,a) = F_{i:}\psi(x,a)$. Then, the prediction of $\mathbb{E}[V(X')|Z = (x,a)]$ simply becomes[6]

$$u_n(V)^\top \psi(x,a) = \bar{V}^\top \pi(x,a).$$

As discussed beforehand, if

$$\|\pi(x,a)\|_1 \le 1, \qquad x \in \{x'_1, \ldots, x'_n\},\ a \in \mathcal{A}\,, \tag{4.8}$$

holds, the Bellman optimality operator of the finite pseudo-MDP given by (4.4) underlying $(\hat{\mathcal{P}}^a)_a$ will be a contraction and thus $\hat{V}^*$, the optimal value function in the pseudo-MDP will exist.

The following counterexample shows that (4.8) is not guaranteed to hold. Consider an MDP with $\mathcal{S} = \{1,2\}$, and $\mathcal{A} = \{1,2\}$. The (state) feature vectors are, $\phi(1) = 1, \phi(2) = 2$. Let $D^{a_j} = diag(d_{1,j}, d_{2,j})$ with $d_{i,j}$ being the frequncy of taking a_j at state i, $i = 1,2; j = 1,2$. Let the samples be arranged such that samples of action a_1 appear first. Let $\Phi^\top = [1,2]$. We have

$$(\Psi^\top\Psi)^\dagger = \begin{pmatrix} (\Phi^\top D^{a_1}\Phi)^\dagger & 0 \\ 0 & (\Phi^\top D^{a_2}\Phi)^\dagger \end{pmatrix} = \begin{pmatrix} 1/(d_{1,1}+4d_{2,1}) & 0 \\ 0 & 1/(d_{1,2}+4d_{2,2}) \end{pmatrix}$$

[6] When using kernels to generate the features, the matrix Ψ will be an $n \times n$ symmetric matrix and the formula given here reduces to that of Grünewälder et al. (2012).

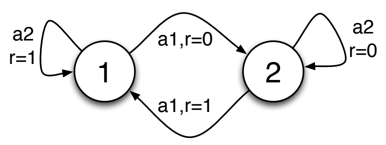

Figure 4.3: An MDP example used to show that least-squares model does not guarantee the L1-norm constraint.

Now

$$\|\pi(1,a_1)\|_1 = \|\Psi(\Psi^\top\Psi)^\dagger\psi(1,a_1)\|_1 = \sum_{i=1}^{n}\psi(x_i,b_i)^\top[1/(d_{1,1}+4d_{2,1}),0]^\top$$
$$= (d_{1,1}+2d_{2,1})/(d_{1,1}+4d_{2,1}).$$

Set $d_{1,1}=9, d_{2,1}=1$ so that $\|\pi(1,a_1)\|_1 \approx 0.8462$. Set $d_{1,2}=1, d_{2,2}=9$. Then $\|\pi(2,a_1)\|_1 = 2\|\pi(1,a_1)\|_1 \approx 1.6923$, $\|\pi(1,a_2)\|_1 = 0.5135$, and $\|\pi(2,a_2)\|_1 = 1.0270$.

Now look at the MDP in Figure 4.6.1, with $\mathcal{P}^{a_1}=[0,1;1,0]; \mathcal{P}^{a_2}=[1,0;0,1]$. $g^{a_2}(1,1)=g^{a_1}(2,1)=1.0, g^{a_1}(1,2)=0.0, g^{a_2}(2,2)=0$. The discount factor is 0.9. The features are specified as above. We used 9 pairs of $(x=1,a=1)$, one pair of $(x=2,a=1)$; one pair of $(x=1,a=2)$ and 9 pairs of $(x=2,a=2)$. Note this guarantees the same model as above. The L1-norm constraint is not satisfied. The AVI procedure using the model quickly diverges.

One solution is to normalize each $\pi(x,a)$ by the ℓ^1 norm (Grünewälder et al., 2012) to guarantee 4.8. As we saw earlier, this may lead to an unwanted performance loss. In the next section, we propose another solution.

4.6.2 The Constrained Optimization Model

We propose to modify the least-squares fitting problem by adding constraint (4.8). The resulting least squares problem can be formulated in terms of the matrix $F \in \mathbb{R}^{n\times d}$:

$$\text{minimize } \|\Psi F^\top - I_{n\times n}\|_F^2 \qquad (4.9)$$
$$\text{subject to } \sum_{j=1}^{n}|F_{j:}\psi(x_i',a)| \le 1, \qquad a\in\mathcal{A}, 1\le i\le n,$$

where $I_{n\times n}$ is the $n \times n$ identity matrix and $\|\cdot\|_F$ denotes the Frobenius norm. Note that the objective function is a quadratic function, while the constraints can be rewritten as linear constraints. To explain the objective function, note that by (4.7), for $V \in L^1(\mathcal{P})$ arbitrary, the least-squares prediction of

$$\int \mathcal{P}^{a_i}(dx'|x_i)V(x') \approx V(x'_i)$$

is $\bar{V}^\top F\psi(x_i,a_i)$. Hence, F should be such that $\sum_{i=1}^n(\bar{V}^\top F\psi(x_i,a_i) - V(x'_i))^2 = \|(\Psi F^\top - I_{n\times n})\bar{V}\|_2^2$ is small.

Choosing $V \in \{e_1,\ldots,e_n\}$ and summing, we get the objective of (4.9). Note that this suggest alternative objectives, such as $\sup_{\bar{V}:\|\bar{V}\|_2\le 1}\|(\Psi F^\top - I_{n\times n})\|_2 = \|\Psi F^\top - I\|_2$, which is again convex.

Let $y = (F_{1:},\ldots,F_{n:})^\top \in \mathbb{R}^{nd}$, $e = (e_1^\top,\ldots,e_n^\top)^\top$. The objective function of (4.9) can be written as

$$\|\Psi F^\top - I_{n\times n}\|_F^2 = \sum_{i=1}^n \|\Psi F_{i:}^\top - e_i\|_2^2 = \|Hy - e\|_2^2, \qquad (4.10)$$

where $H \in \mathbb{R}^{n^2\times nd}$ is defined by

$$H = \begin{pmatrix} \Psi & 0 & \ldots & 0 \\ 0 & \Psi & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \ldots & \Psi \end{pmatrix}.$$

Note that $H^\top H \in \mathbb{R}^{nd\times nd}$ is given by

$$H^\top H = \begin{pmatrix} \Psi^\top\Psi & 0 & \ldots & 0 \\ 0 & \Psi^\top\Psi & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \ldots & \Psi^\top\Psi \end{pmatrix}$$

To put (4.9) into the canonical form of linearly constrained quadratic optimization, introduce the variables $\xi_{j,ia} = |F_{j:}\psi(x'_i,a)|$. Further, let $S_j \in \mathbb{R}^{d\times nd}$ be the block matrix $S_j = (0,\ldots,0,I_{d\times d},0,\ldots,0)$ so that $S_jy = F_{j:}^\top$. With this, we can write

(4.9) as

$$\begin{aligned}
&\text{minimize } y^\top H^\top H y - 2e^\top H y \\
&\text{subject to} \\
&\xi_{j,ia} \geq \psi(x'_i, a)^\top S_j y, \quad 1 \leq i, j \leq n, a \in \mathcal{A}\,, \\
&\xi_{j,ia} \geq -\psi(x'_i, a)^\top S_j y, \quad 1 \leq i, j \leq n, a \in \mathcal{A}\,, \\
&\sum_{j=1}^{n} \xi_{j,ia} \leq 1, \quad 1 \leq i \leq n, a \in \mathcal{A}\,.
\end{aligned}$$

Denote the transition kernels derived from the solution of (4.9) by $(\tilde{\mathcal{P}}^a)_{a\in\mathcal{A}}$ and the resulting pseudo-MDP by $\tilde{N}$.

To summarize, to learn a model and to use it to produce a policy, the following steps are followed: *(i)* data is collected of the form $(\langle z_i, r_i, x'_i\rangle, i = 1, \ldots, n)$, where $z_i = (x_i, a_i) \in \mathcal{Z}$, $x'_i \in \mathcal{X}$ and $r_i \in \mathbb{R}$ (the intention is that $\langle z_i, r_i, x'_i\rangle$ represents a transition sampled from the true model); *(ii)* based on the data, matrix F and then the normalized table $(\tilde{\pi}_j(x_i, a))_{1\leq i,j\leq n, a\in\mathcal{A}}$ are calculated; *(iii)* value-iteration is used to find the optimal value function of the finite pseudo-MDP with n states where the reward at state i is r_i, [7] the transition kernel is $\mathcal{Q}^a(j|i) = \tilde{\pi}_j(x_i, a_i)$. Denote the computed optimal value function by v. We will view v as an n-dimensional vector over x'_i. Finally, computing an optimal action at state $x \in \mathcal{X}$ of the underlying the model that uses $(\tilde{\mathcal{P}}_a)_{a\in\mathcal{A}}$ is obtained by computing $\mathrm{argmax}_{a\in\mathcal{A}}\, g^a(x) + \gamma v^\top \tilde{\pi}(x, a)$. The pseudo code of the proposed algorithm is shown in Algorithm 3.

One can prove that when the constraints in the optimization problem in equation (4.9) are removed, the resulting solution is equal to the least-square solution.

Proposition 12. *Define* $F = \arg\min_{M\in\mathbb{R}^{n\times d}} \|\Psi M^\top - I_{n\times n}\|_F^2$. *We have* $F = \Psi(\Psi^\top\Psi)^\dagger$.

Proof. By setting the gradient of the right-hand side of 4.10 to zero, we have $2(H^\top H f - H^\top e) = 0$. This gives $f = (H^\top H)^\dagger H^\top e$, or $F_{i:} = \psi(x_i, a_i)^\top (\Psi^\top\Psi)^\dagger$ for $i = 1, \ldots, n$, which is equivalent to $F = \Psi(\Psi^\top\Psi)^\dagger$. $\square$

[7] Policy iteration may or may nor converge.

Algorithm 3 Value iteration for FSFLMs.

```
Input: A set of samples $(\langle x_i, a_i, r_i, x'_i \rangle)_{i=1,2,\ldots,n}$, a feature mapping $\psi : \mathcal{X} \times \mathcal{A} \to \mathbb{R}^d$, and a matrix $F \in \mathbb{R}^{n \times d}$.
/*Those $x'_i$ that are terminal have $\psi(x'_i, a) = \mathbf{0}$, which ensures $\pi(x'_i, a) = \mathbf{0}$; for $\forall a \in \mathcal{A}$. */
Output: A policy $\alpha$ and an estimate of $V^\alpha$ at the samples $(x'_i)_{i=1,2,\ldots,n}$
.................................... Modeling ....................................
Compute the action model $\pi$ for all $x'_i$:
For $i = 1, \ldots, n$
   For each action $a$
      Compute $\pi(x'_i, a) = F\psi(x'_i, a)$
   End
End
...................................... AVI ......................................
Initialize policy $\alpha$        /*$\alpha$ is specified at $(x'_i)_{i=1,2,\ldots,n}$ */
Initialize $\bar{V}^\alpha$     /*$n$-dimensional vector; $\bar{V}^\alpha(i)$ estimates $V^\alpha(x'_i)$ */
Generate vector $\bar{r}$ with $\bar{r}(i) = r_i$.
Repeat
   Solve $\bar{V}^\alpha$ for the current policy:     /* iteratively or directly*/
      $\bar{V}^\alpha(i) = \pi(x'_i, \alpha(x'_i))^\top(\bar{r} + \gamma \bar{V}^\alpha), \quad i = 1, 2, \ldots, n,$
   For $i = 1, \ldots, n$
      Compute $\alpha(x'_i) = \arg\max_{a \in \mathcal{A}} \pi(x'_i, a)^\top(\bar{r} + \gamma \bar{V}^\pi)$.
   End
End
```

We need an efficient solver for the constrained approach (standard QP solutions scale poorly with the number of constraints). Let matrix $A \in \mathbb{R}^{d \times |\mathcal{A}|n}$ be $A = [\Psi_{a_1}^\top, \Psi_{a_2}^\top, \ldots, \Psi_{a_{|\mathcal{A}|}}^\top]$ where Ψ_{a_k} is in $\mathbb{R}^{n \times d}$ and $\Psi_{a_k}(i,j) = \psi_j(x_i, a_k)$. The optimization problem can be written as

$$\min_{F : \|A^\top F^\top\|_{1,\infty} \le 1} \frac{1}{2}\|F\Psi^\top - I\|_F^2 \Leftrightarrow \min_{F, Y : Y = A^\top F^\top} \frac{1}{2}\|F\Psi^\top - I\|_F^2 + \delta(\|Y\|_{1,\infty} \le 1). \tag{4.11}$$

where $\delta(\cdot) = 0$ if $\cdot$ is true and ∞ otherwise. Let $\|Z\|_{p,q} := (\sum_i (\sum_j |Z_{ij}|^p)^{q/p})^{1/q}$, *i.e.*, the ℓ_q norm of $(y_1, y_2, \ldots)^\top$ where y_i is the ℓ_p norm of the i-th row of Z. It is well known that the dual norm of ℓ_p norm is the ℓ_{p^*} norm, where $1/p + 1/p^* = 1$. The dual norm of $\|\cdot\|_{p,q}$ is $\|\cdot\|_{p^*,q^*}$.

Note that we are deliberately decoupling $A^\top F^\top$ and Y. We solve the optimization problem on the right-hand side of 4.11 by applying Alternating Direction Method of Multipliers (ADMM) (Boyd et al., 2011), which gradually enforces

$Y = A^\top F^\top$ through minimizing augmented Lagrangian

$$L(F, Y, \Lambda) = \frac{1}{2}\|F\Psi^\top - I\|_F^2 + \delta(\|Y\|_{1,\infty} \leq 1)$$
$$- \operatorname{tr}(\Lambda^\top(Y - A^\top F^\top)) + \frac{1}{2\mu}\|Y - A^\top F^\top\|_F^2$$

in the following steps:

1. Initialize F_0 and set $Y_0 = A^\top F_0^\top$ and $\Lambda_0 = \mathbf{0}$. $t \leftarrow 1$.
2. $Y_t \leftarrow \arg\min_Y L(F_{t-1}, Y, \Lambda_{t-1})$.
3. $F_t \leftarrow \arg\min_F L(F, Y_t, \Lambda_{t-1})$.
4. $\Lambda_t \leftarrow \Lambda_{t-1} + \frac{1}{\mu}(A^\top F_t^\top - Y_t)$.
5. $t \leftarrow t+1$, and go to step 2. Terminate if the difference between Y_t and $A^\top F_t^\top$ falls below some threshold.

Step 2 essentially solves

$$\min_{Y:\|Y\|_{1,\infty}\leq 1} \frac{1}{2}\|Y - Z_t\|_F^2,$$

where $Z_t = \mu\Lambda_{t-1} + A^\top F_{t-1}^\top$. Note the constraint and objective are decoupled along rows, and therefore it suffices to solve

$$\min_{\mathbf{y}:\|\mathbf{y}\|_1\leq 1} \frac{1}{2}\|\mathbf{y}^\top - (Z_t)_{i:}\|_F^2,$$

where $(Z_t)_{i:}$ stands for the i-th row of Z_t. This can be solved in linear time by, *e.g.*, the algorithm of Duchi et al. (2008).

Step 3 minimizes an unconstrained quadratic in F:

$$\min_F \frac{1}{2}\operatorname{tr}(F\Psi^\top\Psi F^\top) + \frac{1}{2\mu}\operatorname{tr}(FAA^\top F^\top) - \operatorname{tr}(C_t^\top F),$$

where $C_t = \Psi - \Lambda_{t-1}^\top A^\top + \frac{1}{\mu}Y_t^\top A^\top$. Setting the gradient of the objective to $\mathbf{0}$, a solution is optimal if and only if $C_t = F\Psi^\top\Psi + \frac{1}{\mu}FAA^\top$. Thus $F_t = C_t(\Psi^\top\Psi + \frac{1}{\mu}AA^\top)^\dagger$. The matrix inversion can be performed once before the iteration starts.

The larger the value of μ is, the less effective is the constraint and thus the closer is the solution to the least-squares solution. In practice, μ is usually set to a small positive constant, *e.g.*, 0.01.

4.7 A Feature Selection Method

Both of the proposed approaches aim to obtain a model F by minimizing $\|\Psi F^\top - I\|_F^2$, which is equal to the sum of the mean square errors of predicting all the basis unit vectors that have length n. The smaller the error, the better the features predict an arbitrary value function. Alternatively, $\|\Psi F^\top - I\|_2^2$ can also be viewed as an upper bound on the empirical loss of an arbitrary value function at the sample points:

$$\sum_{i=1}^{n}(\bar{V}^\top F\psi(x_i, a_i) - V(x_i'))^2 \leq \|\Psi F^\top - I_{n\times n}\|_2^2\|\bar{V}\|_2^2.$$

It thus makes sense to select features from a dictionary that minimize the difference between $\Psi F^\top$ and I. Note that $\bar{V}^\top F\psi(x_i, a_i)$ is a sample of $(\hat{\mathcal{P}}V)$, and $V(x_i')$ is a sample of $\mathcal{P}V$ given the transition (x_i, a_i, x_i'). As a result, the loss function $\|\hat{\mathcal{P}}V^* - \mathcal{P}V^*\|_\infty$ is minimized by the selected features (given sufficient samples). According to Theorem 7, this reduces the upper bound of the performance loss and hence better performance is likely to be expected. A nice property of this feature selection method is that it does not require any transition data.

4.8 A Nonlinear Feature Generation Algorithm

Assume that we are given a dictionary of real-valued functions $\mathcal{D} = (\phi_p(x); p \in \mathcal{I})$, where $\mathcal{I}$ is a (potentially infinite) index set. The problem considered here is to design an algorithm that finds a good, parsimonious approximation of the optimal value function V^* based on some dataset $(\langle z_i, r_i, x'_i \rangle, i = 1, \ldots, n)$ of transitions using the dictionary elements ϕ_p.

Our method works iteratively: The method is initialized by selecting a small set of dictionary elements (this choice is left to the user). Denote the index set of the selected elements by $\mathcal{I}'$. Define the basis functions $B_1 = (\psi_{p,a}; p \in \mathcal{I}', a \in \mathcal{A})$, $\psi_{p,a} : \mathcal{Z} \to \mathbb{R}$ is given by $\psi_{p,a}(x, a') = \phi_p(x)\mathbb{I}_{\{a=a'\}}$. This is our initial sequence of $d_1 = |\mathcal{A}| \times |\mathcal{I}'|$ basis vectors for round one. In round k, we are given a sequence of basis functions $B_k = (\psi_1, \ldots, \psi_{d_k})$.[8] We use the data available and the method of the previous section to get $v_k \in \mathbb{R}^n$ and $\pi_k : \mathcal{Z} \times \mathbb{R}^n$. Next, if V_k defined using $V_k(x) = \max_{a \in \mathcal{A}} g^a(x) + \gamma v_k^\top \pi_k(x, a)$ is too close to V_{k-1}[9] in the sense that $\max_i |V_k(x_i) - V_{k+1}(x_i)| \leq \epsilon$ for some preselected, fixed constant $\epsilon > 0$ then choose a yet unselected dictionary element ϕ_p from $\mathcal{D}$, say, at random, and add $(\psi_{p,a}; a \in \mathcal{A})$ to B_k to get B_{k+1}.[10] Otherwise, add V_k to B_k to get B_{k+1}, finishing round k.

The idea of the method is to work with the current set of features as long as it gives approximations to the value function that are sufficiently dissimilar from the approximation of the previous round. Note that V_k is a nonlinear function of the basis functions due to the presence of $\max$ in its definition. One potential issue with the algorithm is that the complexity of evaluating the basis functions increases rapidly as more and more computed value functions are added. However, this cost may very well be offset by the gain in performance since these value functions are obviously highly useful. In practice, we observed that performance indeed rapidly improves (see below).

[8] Here, we took the liberty to reuse symbol ψ_i with a single index, hoping that this will not cause any confusion.

[9] It is also possible to use $\tilde{\pi}_k$. In our experiments, for simplicity and also because it worked quite well, we used π_k.

[10] For round one, $V_0(x) = \max_{a \in \mathcal{A}} g^a(x)$.

4.8.1 Theoretical Guarantee

From theorem 7 we know that the suboptimality of the policy obtained in round k is bounded as a function of $\|(\hat{\mathcal{P}}_k - \mathcal{P})V^*\|_\infty$, where $\hat{\mathcal{P}}_k$ is the transition model of the pseudo-MDP of round k. Our next theorem shows that the algorithm forces the related quantity, $\|(\hat{\mathcal{P}}^a_k - \mathcal{P}^a)V^*\|_a$ to decrease. Here $\|f\|_a^2 = \frac{1}{|i:a_i=a|}\sum_{i:a_i=a} f^2(x_i)$, *i.e.*, $\|\cdot\|_a$ is the empirical L^2-norm induced by those elements x_i where $a_i = a$ (we assume that there exists such elements). We denote by $\langle\cdot,\cdot\rangle_a$ the underlying inner product: $\langle f, g\rangle_a = \frac{1}{|i:a_i=a|}\sum_{i:a_i=a} f(x_i)g(x_i)$ and Π^a_k is the projection w.r.t. $\|\cdot\|_a$ to the space spanned by the functions in B_k.

Theorem 13. *Either V_k is in the subspace spanned by B_k, or, for any $a \in \mathcal{A}$, we have $\|\Delta\mathcal{P}^a_{k+1}V^*\|_a = |\sin\omega_{k,a}|\ \|\Delta\mathcal{P}^a_k V^*\|_a$, where $\omega_{k,a} = \langle g_k, f_k\rangle_a/(\|f\|_a\|g\|_a)$ is the angle between $g_k = V_k - \Pi^a_k V_k$ and $f_k = \Delta\mathcal{P}^a_k V^*$.*

4.8.2 Proof

For simplicity, the result is proven for finite MDPs; the extension to the general case is routine. Note that in this case $\mathcal{P}^a$ is a $|\mathcal{X}| \times |\mathcal{X}|$ matrix and $(\mathcal{P}^a)_{x',x} = \mathcal{P}^a(x'|x)$. Let ρ_a ($a \in \mathcal{A}$) be the distribution of the states where taking action a; D_a be a diagonal matrix with $D^a(i,i) = \rho_a(x_i)$. Define the inner product for any $z_1, z_2 \in f^{|\mathcal{S}|}$, $\langle z_1, z_2 \rangle_{D_a} = \sqrt{z_1^\top D_a z_2}$. Denote the corresponding L^2 norm by $\|\cdot\|_{D_a}$.

Lemma 14. *We have, for any $a \in \mathcal{A}$,*

$$\hat{\mathcal{P}}^a_{k+1} = \hat{\mathcal{P}}^a_k + \frac{1}{d^2}(\Pi^a_k V_k - V_k)(\Pi^a_k V_k - V_k)^\top D_a \mathcal{P}^a,$$

where $d = \|V_k - \Pi^a_k V_k\|_{D_a}$.

Proof. Let Φ_k be the $|\mathcal{S}| \times d_k$ state feature matrix at iteration k, $\Phi_k = [\phi(x)^\top]_{x \in \mathcal{S}}$, where $\phi(x) : \mathcal{S} \to \mathbb{R}^{d_k}$, d_k being the current number of feature functions. Then $\Pi^a_k = \Phi_k(\Phi_k^\top D_a \Phi_k)^{-1}\Phi_k^\top D_a$. According to the feature iteration algorithm, $\Phi_{k+1} = [\phi(x)^\top, V_k(x)]_{x \in \mathcal{S}}$. We have $\hat{\mathcal{P}}^a_{k+1} = \Pi^a_{k+1}\mathcal{P}^a$. We can finish by proving that $\Pi^a_{k+1} = \Pi^a_k + 1/d^2(\Pi^a_k V_k - V_k)(\Pi^a_k V_k - V_k)^\top D_a$ by noticing that

$$\begin{aligned}
(\Phi_{k+1}^\top D_a \Phi_{k+1})^{-1} &= ([\Phi_k, V_k]^\top D_a [\Phi_k, V_k])^{-1} \\
&= \begin{bmatrix} \Phi_k^\top D_a \Phi_k & \Phi_k^\top D_a V_k \\ V_k^\top D_a \Phi_k & V_k^\top D_a V_k \end{bmatrix}^{-1} \\
&= \begin{bmatrix} (\Phi_k^\top D_a \Phi_k)^{-1} + 1/d^2 v_k v_k^\top & -v_k/d^2 \\ -v_k^\top/d^2 & 1/d^2 \end{bmatrix},
\end{aligned}$$

where $v_k = (\Phi_k^\top D_a \Phi_k)^{-1}\Phi_k^\top D_a V_k$. □

Lemma 15. *For any vector $y \in \mathbb{R}^{|\mathcal{S}|}$ and any $a \in \mathcal{A}$, we have $\|\Delta\mathcal{P}^a_{k+1} y\|^2_{D_a} = \|\Delta\mathcal{P}^a_k y\|^2_{D_a} - d_y^2/d^2$, where $d_y = (\Pi^a_k V_k - V_k)^\top D_a \mathcal{P}_a y$.*

Proof.

$$\begin{aligned}
\|\Delta\mathcal{P}^a_{k+1} y\|^2_{D_a} &= (\hat{\mathcal{P}}^a_{k+1} y - \mathcal{P}^a y)^\top D_a (\hat{\mathcal{P}}^a_{k+1} y - \mathcal{P}^a y) \\
&= z^\top D_a z,
\end{aligned}$$

where

$$\begin{aligned}
z &= \hat{\mathcal{P}}^a_{k+1} y - \mathcal{P}^a y \\
&= \hat{\mathcal{P}}^a_k y + 1/d^2(\Pi^a_k V_k - V_k)(\Pi^a_k V_k - V_k)^\top D_a \mathcal{P}_a y - \mathcal{P}_a y \\
&= \hat{\mathcal{P}}^a_k y + d_y/d^2(\Pi^a_k V_k - V_k) - \mathcal{P}_a y,
\end{aligned}$$

according to Lemma 14 and the definition of d_y. Thus,

$$\begin{aligned}
\|\Delta\mathcal{P}^a_{k+1} y\|^2_{D_a} &= (\hat{\mathcal{P}}^a_k y - \mathcal{P}^a y)^\top D_a(\hat{\mathcal{P}}_a y - \mathcal{P}^a y) + \\
&\quad 2d_y/d^2(\hat{\mathcal{P}}^a_k y - \mathcal{P}^a y)^\top D_a(\Pi^a_k V_k - V_k) + d_y^2/d^2 \\
&= (\hat{\mathcal{P}}^a_k y - \mathcal{P}^a y)^\top D_a(\hat{\mathcal{P}}_a y - \mathcal{P}^a y) - 2d_y/d^2(\mathcal{P}^a y)^\top D_a(\Pi^a_k V_k - V_k) \\
&\quad + d_y^2/d^2 \\
&= \|\Delta\mathcal{P}^a_k y\|^2_{D_a} - d_y^2/d^2.
\end{aligned}$$

□

We are ready to prove Theorem 13.

Proof. According to Lemma 15, we have

$$\|\Delta\mathcal{P}^a_{k+1} V^*\|_{D_a} = \|\Delta\mathcal{P}^a_k V^*\|_{D_a} - \frac{((\Pi^a_k V_k - V_k)^\top D_a \mathcal{P}_a V^*)^2}{\|\Pi^a_k V_k - V_k\|^2_{D_a}},$$

or,

$$\|\Delta\mathcal{P}^a_{k+1} V^*\|_{D_a} = \|\Delta\mathcal{P}^a_k V^*\|_{D_a} - \frac{((\Pi^a_k V_k - V_k)^\top D_a(\mathcal{P}_a V^* - \Pi^a_k \mathcal{P}_a V^*))^2}{\|\Pi^a_k V_k - V_k\|^2_{D_a}},$$

If $\|\Delta\mathcal{P}^a_k V^*\|_{D_a} = 0$, the optimal policy is achieved; otherwise, note that $\Pi^a_k \mathcal{P}_a = \hat{\mathcal{P}}^a_k$, we have

$$(\epsilon^a_k)^2 = 1 - \frac{((V_k - \Pi^a_k V_k)^\top D_a(\Delta\mathcal{P}^a_k V^*))^2}{\|\Pi^a_k V_k - V_k\|^2_{D_a}\|\Delta\mathcal{P}^a_k V^*\|^2_{D_a}}.$$

We complete the proof by noticing that the second therm is just $\cos^2\theta_a$. □

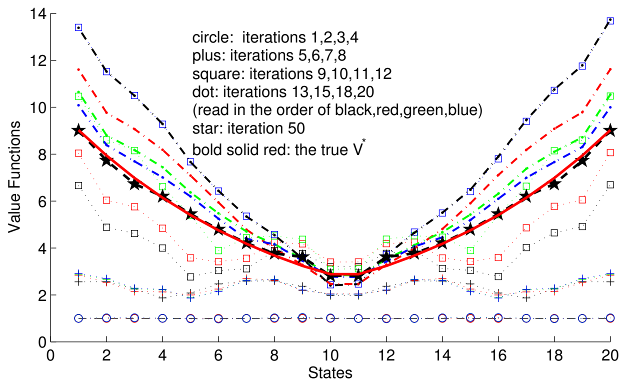

Figure 4.4: Value functions for the 20-state problem for various rounds.

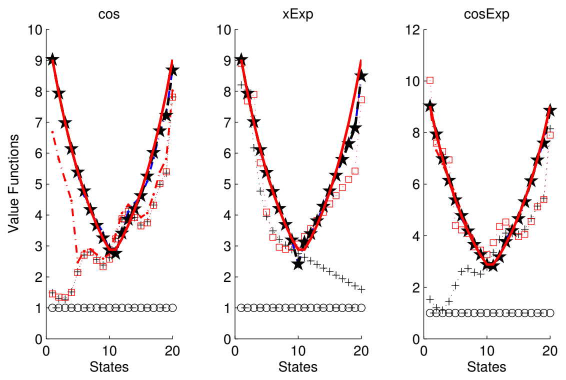

Figure 4.5: Value functions for the 20-state problem for rounds 1 (circle), 5 (plus), 10 (square), 15 (dotted) and 20 (star) for three different dictionary sets. The graph labeled by “cos” corresponds to $f_p(i) = \cos(i - p)$; “xExp” to $f_p(i) = (i - p)\exp^{-(i-p)}$; “cosExp” to $f_p(i) = \cos(i - p)\exp^{-(i-p)/20}$.

4.9 Empirical Results

The point of the experiments is to demonstrate the behavior of the feature generation method. In particular, we designed experiments to test the robustness of the method against changing the dictionary. We also investigate the power of the feature generation method by removing the option of adding the learned value function (effectively, setting $\epsilon = \infty$ in the algorithm). We chose a simple domain so that we can visualize the findings easier.

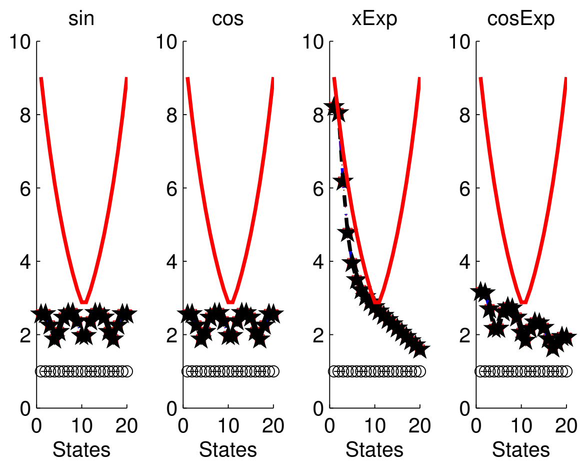

Figure 4.6: The learned value functions for 20-state chain with the feature-generation turned off. The value functions for rounds 1, 5, 10, 15, 20 and 50 are shown.

4.9.1 Chain-walks

The chain walk problem is already stated in Section 3.4.1. We consider here 20-state chain walk. The reward is one at states 1 and 20. Elsewhere the reward is zero. No exploration is used. The initial set of features consists of polynomials of degree zero and one. In both cases, neither LSPI or LAM-API can find the optimal policy with these features alone (Lagoudakis and Parr, 2003; Yao and Szepesvári, 2012). We used 10 samples of each state-action pair (thus 400 samples in total). During feature generation, the threshold for adding the obtained value function is $\epsilon = 10^{-4}$. The pth dictionary element to be added after the first round is $f_p(i) = \sin(i - p)$ (a deliberately poor choice for a dictionary).

Figure 4.4 shows the evolution of value functions during the rounds. During the first 40 rounds, the algorithm added a dictionary element only at rounds 3 and 5. All policies after round 9 are optimal. To confirm that the findings are robust when the dictionary is changed, we rerun the experiments with other dictionaries (cf. Figures 4.5). The experiments indeed confirm the robustness of the method. Finally, to test the contribution of the feature generation method we ran experiments with $\epsilon = \infty$ so that the value functions are never added to the basis set. In Figure 4.6, the nonlinear' feature generation method is turned off. The result in the figure shows that without the feature method, many more dictionary elements are necessary to obtain good results.

4.10 Chain Walk: LS diverges and constrained performs well

Consider a 4-state chain in Section 3.4.1. The discount factor was 0.99 in this experiment. The features are, $\psi(x) = [1, x, x^2]$, where $x \in \mathcal{X} = \{1, 2, 3, 4\}$. Value iteration is used for both the LS model and the constrained model. We used 10 samples for each state action pair (thus 80 samples in total). We performed up to 20 iterations for both the value iteration procedures. An ℓ^2 regularization factor 1.0 was used for the LS model.

Figure 4.7 shows that value iteration using the LS model quickly diverges, and using the constrained model is convergent. Figure 4.8 shows that the learned value function using the constrained model has a good shape (in fact, the learned policy is optimal). The value function of the LS model happens to have a good shape althought it is divergent.

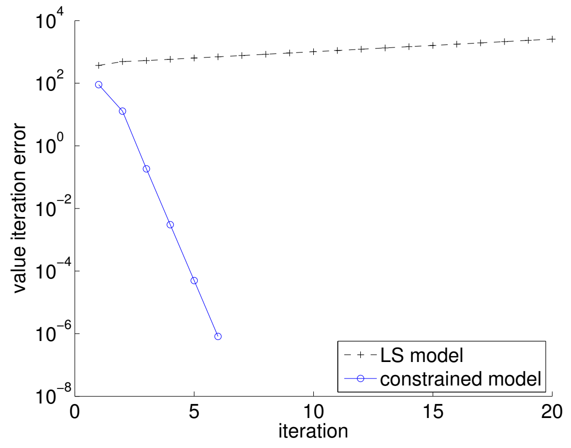

Figure 4.7: Iteration errors of the value iteration algorithms using the least-squares model and the constrained model.

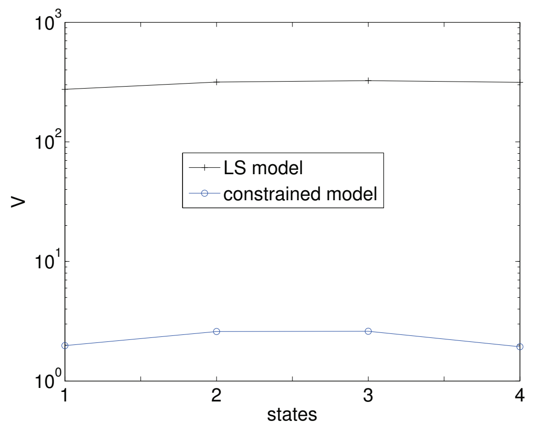

Figure 4.8: 4-state chain: Value functions learned using the least-squares model and the constrained model.

4.10.1 Chain-Walk: Feature Selection

We consider a 4-state chain (see Section 3.4.1).

The dictionary contains 55 RBFs with the mean varying from 0 to 10 and the variance from $[1, 1.75, 2.5, 3.25, 4]$. The dictionary is chosen to be extensive to contain good features. The original features are 3 polynomial features, $\psi(x) = [1, x, x^2]$, for $x = 1, 2, 3, 4$. We used 10 samples for each state action pair (thus 80 samples in total). For the reported results in this experiment, we used the constraint model for both feature methods. The results are similar using LS model with normalization.

We used matching pursuit to select four features from the dictionary. The selected RBFs are, $\phi(x) = \exp^{-\|x-u_i\|_2^2/(2\sigma_i^2)}$, with

$$(u_i, \sigma_i) \in \{(2, 1.00), (2, 3.25), (5, 1.00), (8, 3.25)\}.$$

Figure 4.9 shows the learned value function using the automatically selected RBFs as well as the value function learned by using the original polynomial features by Lagoudakis and Parr (2003). Both feature methods lead to the optimal policy, but the feature selection method gives a much more accurate estimate of the optimal value function. In particular, the learning error of the feature selection method is, $\|V^* - \hat{V}^*\|_2 = 5.2749 \times 10^{-6}$. Figure 4.10 shows the approximate identity matrix using the polynomial features, which has an error of 74.00. Figure 4.11 shows the approximate identity matrix using the feature selection method, which has an of error 72.00. This shows that improving features by reducing $\|\Psi F^\top - I\|_F^2$ gives a better value function.

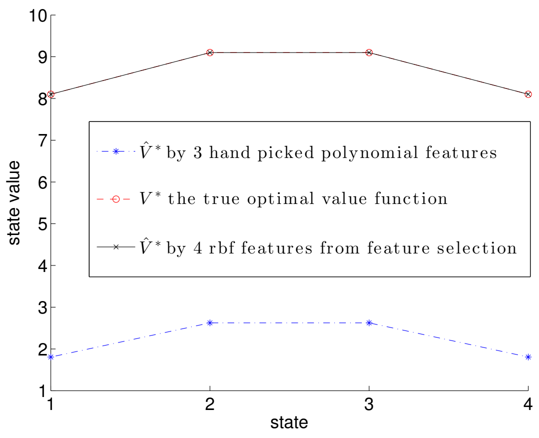

Figure 4.9: Value functions on 4-state chain.

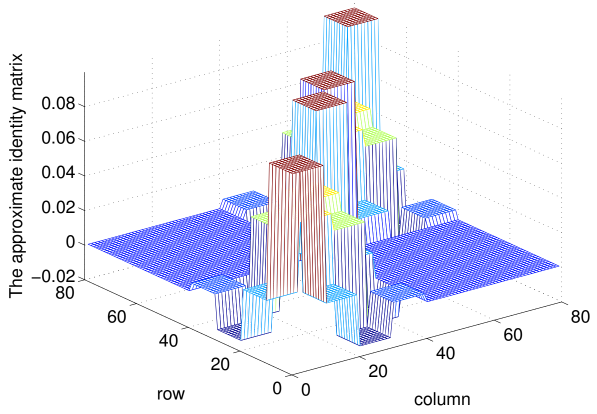

Figure 4.10: Approximation of the identity matrix by the three polynomial features.

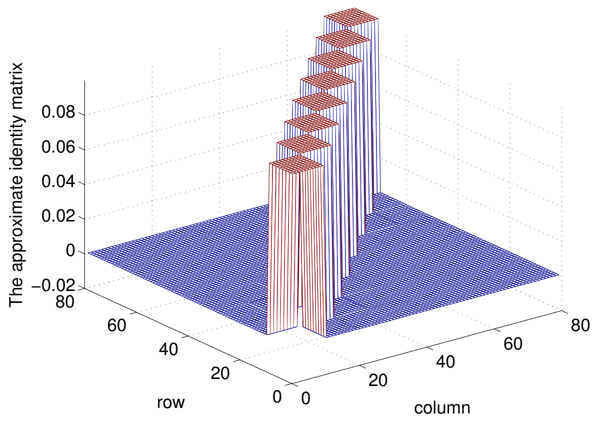

Figure 4.11: Approximation of the identity matrix by the four automatically selected RBFs.

4.10.2 Cart-pole Balancing

The problem is studied in Section 3.4.3. No exploration was used.

Recall that for both the LS model and the constraint model the goal is to learn a matrix F such that $\hat{I} = \Psi F^\top$ approximates the identity matrix $I_{n\times n}$ well where n is the number of samples. We first tried the nine radial basis functions plus a constant feature Lagoudakis and Parr (2003). For a state s, $\phi_i(s) = \exp(-||s - u_{i-1}||^2/2)$, $i = 1, 2, \ldots, 10$, where $u_0 = s$, and the other u_i are the points from the grid $\{-\pi/4, 0, \pi/4\} \times \{-1, 0, 1\}$ (Lagoudakis and Parr, 2003). The algorithms did not perform well with these features. It turns out that the approximation of the identity matrix is poor for both models. For example, the LS approximation is shown in Figure 4.12 using about $1,600$ samples collected by a random policy by starting the pole from a random state near the state $[0, 0]$. The diagonal part is well approximated but the other part is noisy.

To circumvent this problem, we used a new feature method which we call the tensor-product features. We first partitioned the state space into rectangular cells and then use the RBF features restricted to the cells to provide generalization. Both the LS model and the constraint model approximate the identity matrix well with this feature method. For example, Figure 4.13 shows the approximated identity matrix using LS. In these two figures, we partitioned each state dimension into three equal parts. There are effectively three grids laid in the state space because six of them are all failure states (with $|\theta| > \pi/2$) whose feature vector is all zero. To illustrate the matrix better, we sorted the samples according to the grid index that x_i belongs to and then according to the action a_i using a stable sorting algorithm. Because of this feature method the approximate matrix contains only diagonal blocks and outside these blocks the values are strictly zero. The sorting operation ensures that the diagonal part of the approximate identity matrix is in the order of action blocks, each of which contains the smaller approximate identity matrices for the grids. The approximated identity matrix (using ADMM) is 3D visualized in Figure 4.14. The ADMM algorithm was run with $\mu = 1.0$ and 30 iterations. The algorithm took 318 seconds for 30 iterations on a desktop with 1.7GHz Intel Core i7 and 8GB 1600 MHz DDR3.

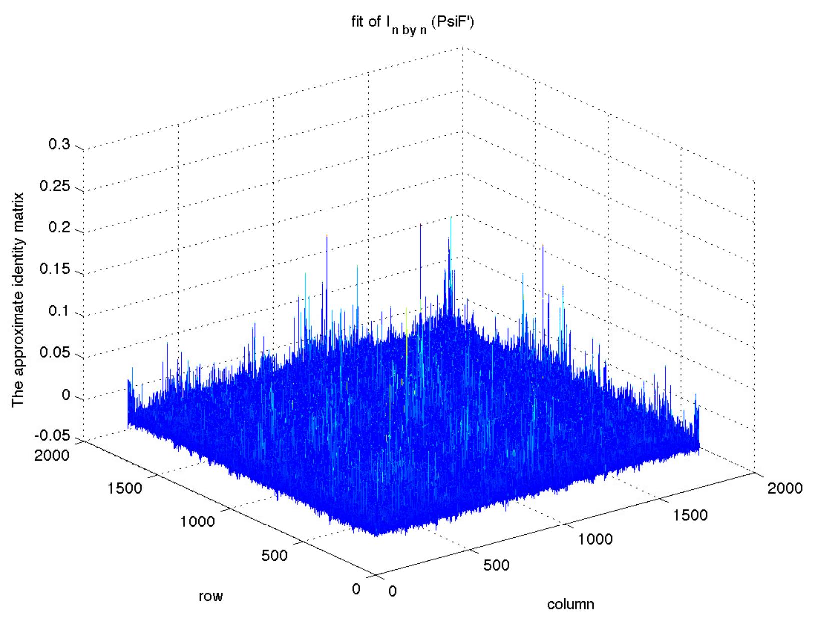

Figure 4.12: 3D plot of the approximate identity matrix using the RBF features used by Lagoudakis and Parr (2003).

In order to evaluate the performances of the normalized LS model and the constraint model, we conducted 30 independent runs of experiment. In each run, we collected a number of episodes of samples from the random policy. We learned the LS model and the constraint model, and then used them independently in AVI to compute an approximate optimal policy. The LS model used ℓ^1 normalization. Each model was used in the AVI procedure to produce a policy. We then evaluated each policy 100 times up to 3000 steps in each evaluation. The averaged number of balanced steps was then used as a measure of the quality of the model. Figure 4.15 shows the balanced steps by the policies for both models. The constraint model is substantially better than the LS model. The constraint model was solved using $\mu = 1.0$ and 10 iterations for all size of training data.

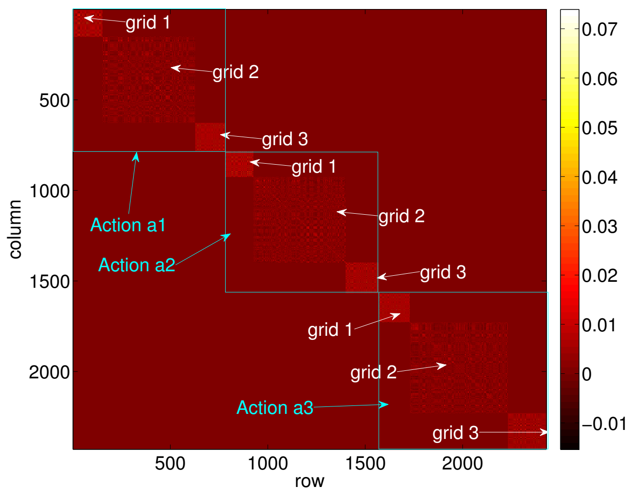

Figure 4.13: The approximate identity matrix by the tensor-product features (LS fit using 3 partitions in each state dimension).

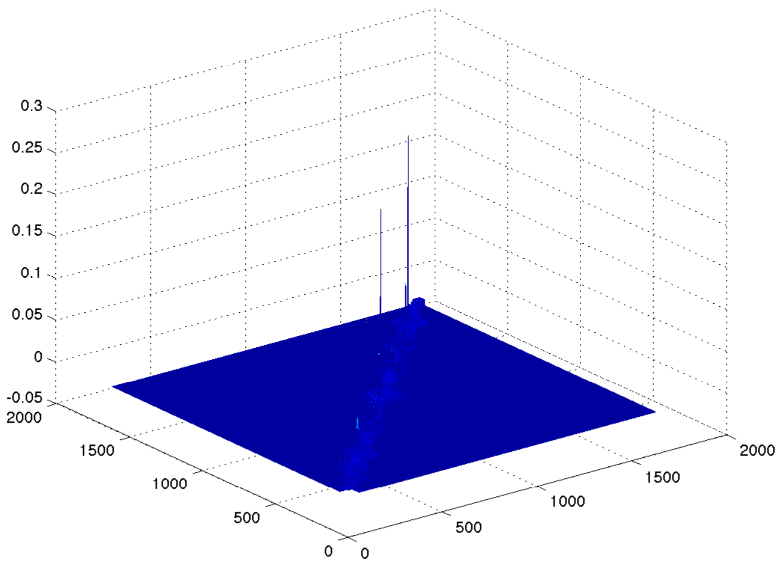

Figure 4.14: 3D plot of the approximate identity matrix by the tensor-product features .

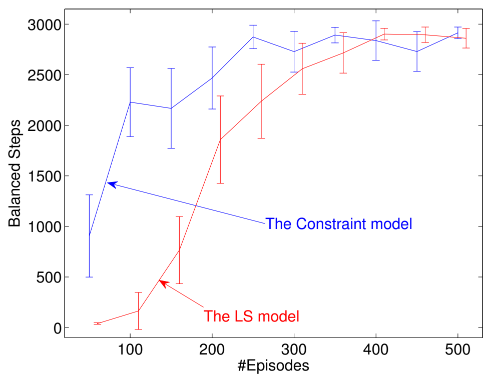

Figure 4.15: The balanced steps of the pole for the cart-pole system by the AVI algorithms using the normalized LS model and the constraint model.

4.11 Conclusion

In this chapter, we have proposed a pseudo-MDP framework. In this framework, we work with an approximate model to derive a near-optimal policy for a given MDP. We give a generic error bound for the performance of the learned policy. This result shows that only the policy evaluation errors of an optimal policy in the MDP and a pull-back policy of an optimal policy in the pseudo-MDP matter. This is interesting because it gives a direction for model-based reinforcement learning: an exhaustively accurate model is not necessary in order to learn a good policy—it only needs to be accurate in two projected directions (some terms in the error bound). Although one error term (the one that depends on an optimal policy of the MDP) is unknown, we show by a feature learning method that it can be approximated; so that we can learn new features to reduce it to give tighter error bound and better performance.

Our pseudo-MDP work is interesting not only because it relates to recent kernel-embedding models and recovers their error bound (Grünewälder et al., 2012), but it also opens the door to new models. We explored a class of models called *finitely supported factored linear models* (FSFLM) (see Section 4.3). These models have a nice property that they can be learned efficiently. We use least-squares and constrained optimization methods to learn them. We show that the least-squares without normalization can possibly diverge; while with normalization the least-squares model can give bad policies. The constrained optimization model is guaranteed to converge, and constantly gives good policies in our experiments. However, for the constrained model there are computation scalability issues that remain to be addressed. The procedures that are based on the dual representation that we also use need to work and sometimes invert $n \times n$ matrices for a sample size of n. This is clearly infeasible even for medium size data. In the supervised learning literature much research went into speeding up these operations, such as Nyström's method and variants (Le et al., 2013; Hsieh et al., 2014). It will be interesting to incorporate these into our setting. In the RL literature, there have been advancements for scenarios where ψ satisfies certain structural assumptions (Kveton and Theocharous, 2013), or sample subsampling (Kveton and Theocharous, 2012), which we can also try to incorporate.

Chapter 5

Universal Option Models

5.1 Introduction

Conventional methods for real-time abstract planning over options in reinforcement learning require a single pre-specified reward function. Here we consider planning with the same dynamics but multiple reward functions. This problem arises in a number of scenarios. In inverse reinforcement learning and apprenticeship learning there is a set of reward functions from which a good reward function is extracted (Ng and Russell, 2000; Abbeel et al., 2010; Syed, 2010). Some system designers iteratively refine their provided reward functions to obtain desired behavior, and will re-plan in each iteration. In real-time strategy games, several units on a team can share the same dynamics but have different time-varying capabilities, so selecting the best unit for a task requires knowledge of the expected performance for many units. Even article recommendation can be viewed as a multiple-reward planning problem, where each user query has an associated reward function and the relevance of an article is given by walking over the links between the articles (Page et al., 1998; Richardson and Domingos, 2002). We propose to unify the study of such problems within the setting of real-time planning with abstractions, where a reward function can be specified at any time and the expected option-conditional return for a reward function must be efficiently computed.

Function approximation deals with spatial abstraction. *Planning with temporal abstractions*, which provides temporal abstraction, enables one to make abstract decisions that involve sequences of low level actions. Options are often used to specify action abstraction (Sutton et al., 1999; Precup, 2000; Sorg and Singh, 2010). An option is a course of temporally extended actions, which starts execution at some states, and follows a policy in selecting actions until it terminates. When an option terminates, the agent can start executing another option. The traditional model of an option takes in a state and predicts the sum of the rewards in the course till termination, and the probability of terminating the option at any state. When the reward function is changed, abstract planning with the traditional option model has to start from scratch.

We introduce universal option models (UOM) as a solution to this problem. The

UOM of an option has two parts. A *termination prediction* part, as in the traditional option model, predicts the states where the option terminates. An *accumulation* part, new to the UOM, predicts the occupancies of all the states by the option after it starts execution. We also extend UOMs to linear function approximation, with which UOMs scale to problems with a large state space. We show that the UOM outperforms existing methods in two domains.

5.2 Universal Option Models (UOMs)

In this section, we define the UOM for an option, and prove a universality theorem stating that the traditional model of an option can be constructed from the UOM and a reward vector of the option.

For the basics of options, please refer to Section 2.5. The goal of UOMs is to make models of options that are independent of the reward function. We use the adjective "universal" because the option model becomes universal with respect to the rewards. In the case of MDPs, it is well known that the value function of a policy π can be obtained from the so-called *discounted occupancy function* underlying π, e.g., see (Barto and Duff, 1994). This technique has been used in inverse reinforcement learning to compute a value function with basis reward functions (Ng and Russell, 2000). The generalization to options is as follows. First we introduce the *discounted state occupancy function*, u^o, of option $o(\pi, \beta)$:

$$u^o(x, x') = \mathbb{E}\Big[\sum_{t=0}^{T-1} \gamma^t \, \mathbb{I}_{\{X_t = x'\}}\Big] . \tag{5.1}$$

Then,

$$R^o(x) = \sum_{x' \in \mathcal{X}} f^\pi(x') \, u^o(x, x') , \tag{5.2}$$

where f^π is the expected immediate reward vector under π and $(f^a)_{a\in\mathcal{A}}$, i.e., for any $x \in \mathcal{X}$, $f^\pi(x) = \sum_{a\in\mathcal{A}} \pi(x,a) f^a(x)$. For convenience, we shall also treat $u^o(x, \cdot)$ as a vector and write $u^o(x)$ to denote it as a vector. To clarify the independence of u^o from the reward function, it is helpful to first note that every MDP can be viewed as the combination of an immediate reward function, $(f^a)_{a\in\mathcal{A}}$, and a *reward-less MDP*, $\mathcal{M} = (\mathcal{X}, \mathcal{A}, (\mathcal{P}^a)_{a\in\mathcal{A}}, \gamma)$.

Definition 2. *The UOM of option o in a reward-less MDP is defined to be the pair (u^o, p^o), where u^o is the option's discounted state occupancy function, defined by* (5.1)*, and p^o is the option's discounted terminal state distribution, defined by* (2.7).

Theorem 16. *Fix an option $o(\pi, \beta)$ in a reward-less MDP $\mathcal{M}$, and let u^o be the occupancy function underlying o in $\mathcal{M}$. Let $(f^a)_{a\in\mathcal{A}}$ be some immediate reward* *function. Then, for any state $x \in \mathcal{X}$, the return of option o with respect to $\mathcal{M}$ and $(f^a)_{a\in\mathcal{A}}$ is given by $R^o(x) = (u^o(x))^\top f^\pi$.*

Proof. We start with the definition of the return of an option:

$$\begin{aligned}R^o(x) &= \mathbb{E}[R_1 + \gamma R_2 + \dots \gamma^{T-1} R_T] \\ &= \mathbb{E}\Big[\sum_{t=0}^{\infty} \gamma^t\, R_{t+1}\, \mathbb{I}_{\{t<T\}}\Big] = \sum_{t=0}^{\infty} \gamma^t\, \mathbb{E}[R_{t+1}\, \mathbb{I}_{\{t<T\}}]\,.\end{aligned}$$

Now,

$$\begin{aligned}&\mathbb{E}[R_{t+1}\, \mathbb{I}_{\{t<T\}}] = \mathbb{E}[\,\mathbb{E}[R_{t+1}\, \mathbb{I}_{\{t<T\}} | X_t]\,] \\ &= \mathbb{E}[\,\mathbb{I}_{\{t<T\}}\, \mathbb{E}[R_{t+1}|X_t]\,] = \mathbb{E}[\,\mathbb{I}_{\{t<T\}}\, f^\pi(X_t)\,]\,,\end{aligned}$$

where the second to last equality follows from the following reasoning: Let $E_1, E_2, \dots$ be the indicator random variables such that $E_i = 1$ if at step i, given X_i the option execution is stopped, i.e., these represent the results of coin flips, which determine if the process is continued. Note that given X_t, R_{t+1} and X_t are independently drawn from each other. Similarly, given X_t, R_{t+1} is independently sampled from E_i for $i < t$. Now, by its definition, T is the first time when $E_T = 1$. Thus, $\{t < T\} = \{E_1 = 0, \dots, E_t = 0\}$. Therefore,

$$\begin{aligned}&\mathbb{E}[R_{t+1}\, \mathbb{I}_{\{t<T\}}|X_t] = \mathbb{E}[R_{t+1}\, \mathbb{I}_{\{E_1=0,\dots,E_t=0\}}|X_t] \\ &= \mathbb{E}[\mathbb{E}[R_{t+1}\, \mathbb{I}_{\{E_1=0,\dots,E_t=0\}}|X_t, E_1, \dots, E_t]|X_t] \\ &= \mathbb{E}[\mathbb{I}_{\{E_1=0,\dots,E_t=0\}}\, \mathbb{E}[R_{t+1}|X_t, E_1, \dots, E_t]|X_t]\,.\end{aligned}$$

Now, $\mathbb{E}[R_{t+1}|X_t, E_1, \dots, E_t] = \mathbb{E}[R_{t+1}|X_t]$, since R_{t+1} is independent of $E_1, \dots, E_t$, given X_t. Thus, $\mathbb{E}[R_{t+1}\, \mathbb{I}_{\{t<T\}}|X_t] = \mathbb{E}[\mathbb{I}_{\{t<T\}}\, \mathbb{E}[R_{t+1}|X_t, E_1, \dots, E_t]|X_t]$, as promised.

Continuing our calculation, notice that by the law of total expectation, $\mathbb{E}[\,\mathbb{I}_{\{t<T\}}\, f^\pi(X_t)\,] = \sum_{x'\in\mathcal{X}} f^\pi(x')\mathbb{E}\{\mathbb{I}_{\{t<T, X_t = x'\}}\}$. Thus,

$$\begin{aligned}R^o(x) &= \sum_{x'\in\mathcal{X}} f^\pi(x')\, \mathbb{E}\Big[\sum_{t=0}^{\infty} \gamma^t\, \mathbb{I}_{\{t<T, X_t=x'\}}\Big] \\ &= \sum_{x'\in\mathcal{X}} f^\pi(x')\, u^o(x, x')\,,\end{aligned}$$

thus finishing the proof. □

5.3 UOMs with Linear Function Approximation

In this section, we introduce linear universal option models which use linear function approximation to compactly represent reward independent option-models over a potentially large state space. In particular, we build upon previous work where the approximate solution has been obtained by solving the so-called projected Bellman equations. We assume that we are given a function $\phi : \mathcal{X} \to \mathbb{R}^d$, which maps any state $x \in \mathcal{X}$ into its d-dimensional *feature representation* $\phi(x)$. Let $V_\theta : \mathcal{X} \to \mathbb{R}$ be defined by $V_\theta(x) = \theta^\top \phi(x)$, where the vector θ is a so-called weight-vector. [1] Fix an initial distribution μ over the states and an option $o = o(\pi, \beta)$. Given a reward function $f = (f^a)_{a \in \mathcal{A}}$, $0 \le \lambda \le 1$, the TD(λ) approximation $V_{\theta^{(\mathrm{TD},f)}}$ to R^o is defined as the solution to the following projected Bellman equations (Sutton and Barto, 1998):

$$\mathbb{E}\Big[\sum_{t=0}^{T-1} \{R_{t+1} + \gamma V_\theta(X_{t+1}) - V_\theta(X_t)\}\ Z_{t+1} \Big] = 0\,.$$

Here X_0 is sampled from μ, the random variables $(R_1, X_1, R_2, X_2, \ldots)$ and T (the termination time) are obtained by following o from this initial state until termination, and

$$Z_{t+1} = \sum_{\ell=0}^{t} (\gamma\lambda)^{t-\ell} \phi(X_\ell)\,. \tag{5.3}$$

It is easy to see that if $\gamma = \lambda = 0$ then $V_{\theta^{(\mathrm{TD},f)}}$ becomes the *least-squares* approximation $V_{\omega^{(\mathrm{LS},f)}}$ to the immediate rewards f under o given the features ϕ. The least-squares approximation to f is given by

$$\omega^{(\mathrm{LS},f)} = \arg\min_\omega J(\omega) = \mathbb{E}\Big[\sum_{t=0}^{T-1} \big\{R_{t+1} - \omega^\top \phi(X_t)\big\}^2 \Big].$$

We restrict our attention to this TD(0) solution in this chapter, and refer to ω as an (approximate) immediate reward model.

The *TD(0)-based linear UOM* (in short, linear UOM) underlying o (and μ) is a pair of $d \times d$ matrices (U^o, M^o), which generalize the tabular model (u^o, p^o).

[1]Note that the subscript in $V_\cdot$ always means the TD weight vector throughout this chapter.

Basically the ith row of U^o corresponds to the TD weight vector under an artificial "reward" signal, $\breve{r}^i = \phi^i$, which is the ith feature. Formally, $(U^o)^\top = [u_1, \ldots, u_d]$, $u_i \in \mathbb{R}^d$, where u_i is the weight vector such that V_{u_i} is the TD(0)-approximation to the return of o for the artificial reward. We update an estimate of U^o at each time step during the execution of the option, by

$$U^o \leftarrow U^o + \eta \left[\phi_t + \gamma U^o \phi_{t+1} - U^o \phi_t\right] \phi_t^\top, \tag{5.4}$$

where η is a step-size and ϕ_t is the feature vector of the tth state X_t. [2] Note that if we use tabular representation, then $u_{i,x} = u^o(x,i)$ holds for all $x, i \in \mathcal{X}$. Therefore our extension to linear function approximation is backward consistent with the UOM definition in the tabular case. However, this alone would not be a satisfactory justification of this choice of linear UOMs. The following theorem shows that just like the UOMs of the previous section, the U^o matrix allows the separation of the reward from the option models without losing information.

Theorem 17. *Fix an option $o = o(\pi, \beta)$ in a reward-less MDP, $\mathcal{M} = (\mathcal{X}, \mathcal{A}, (\mathcal{P}^a), \gamma)$, an initial state distribution μ over the states $\mathcal{X}$, and a function $\phi : \mathcal{X} \to \mathbb{R}^d$. Let U be the linear UOM of o w.r.t. ϕ and μ. Pick some reward function f and let $V_{\theta^{(\mathrm{TD},f)}}$ be the TD(0) approximation to the return R^o. Then, for any $x \in \mathcal{X}$,*

$$V_{\theta^{(\mathrm{TD},f)}}(x) = (\omega^{(\mathrm{LS},f)})^\top \left(U\phi(x)\right).$$

Proof. Let

$$A = \mathbb{E}\Big[\sum_{t=0}^{T-1} \phi(X_t)\left(\gamma^t \phi(X_{t+1}) - \phi(X_t)\right)^\top\Big],$$
$$C = \mathbb{E}\Big[\sum_{t=0}^{T-1} \phi(X_t)\phi(X_t)^\top\Big].$$

It is not hard to see that U satisfies $C + AU^\top = 0$, any $\theta^{(\mathrm{TD},f)}$ satisfies

$$b_f + A\theta^{\mathrm{TD},f} = 0 \tag{5.5}$$

and any $\omega^{(\mathrm{LS},f)}$ satisfies $b_f - C\omega^{(\mathrm{LS},f)} = 0$,[3] where $b_f = \mathbb{E}[\sum_{t=0}^{T-1} \phi(X_t) R_{t+1}]$. Let $\hat{\theta} = U^\top \omega^{\mathrm{LS},f}$. The claim of the theorem is that $\hat{\theta}$ satisfies (5.5). By simple algebra,

[2] Each u_i has a standard TD update, $u_i \leftarrow u_i + \eta(\phi_t^i + \gamma u_i^\top \phi_{t+1} - u_i^\top \phi_t)\phi_t$.

[3] Here, we allow for the non-uniqueness of solutions. Uniqueness follows under stronger assumptions, which we do not need here.

we indeed find $b_f + A\hat{\theta} = b_f + AU^\top \omega^{\mathrm{LS},f} = b_f - C\omega^{\mathrm{LS},f} = 0$, thus finishing the proof. $\square$

The significance of this result is that it shows that to compute the TD approximation of an option return corresponding to a reward function f, it suffices to find $\omega^{(\mathrm{LS},f)}$ (the least squares approximation of the expected one-step reward under the option and the reward function f), provided one is given the U matrix of the option. We expect that finding a least-squares approximation (solving a regression problem) is easier than solving a TD fixed-point equation. Note that the result also holds for standard policies, but we do not explore this direction in this thesis.

The definition of M^o. The matrix M^o serves as a state predictor, and we call M^o the *transient matrix* associated with option o. Given a feature vector ϕ, $M^o\phi$ predicts the (discounted) expected feature vector where the option stops. When option o is started from state X and stopped at state X_T in T time steps, we update an estimate of M^o by

$$M^o \leftarrow M^o + \eta(\gamma^T \phi(X_T) - M^o\phi(X))\phi(X)^\top .$$

Formally, M^o is the solution to the associated linear system,

$$\mathbb{E}_{\mu,o}[\, \gamma^T \phi(X_T)\phi(x)^\top \,] = M^o\, \mathbb{E}_{\mu,o}[\, \phi(X)\phi(X)^\top \,] \,, \tag{5.6}$$

where $\mathbb{E}_{\mu,o}$ is the expectation operator with respect to μ (the distribution where X_0 is sampled from) and the option o. Notice that M^o is thus just the least-squares solution of the problem when $\gamma^T\phi(X_T)$ is regressed on $\phi(X)$, given that we know that option o is executed. Again, this way we obtain the terminal distribution of option o in the tabular case.

Our goal is to find an option model that can be used to compute a TD approximation to the value function of a high-level policy h (flattened) over a set of options $\mathcal{O}$. The following theorem shows that the linear UOM suits this purpose. In practice, this means that we can speed up the computation of multiple V^h that depend on different reward functions.

A high-level policy h defines a Markov chain over $\mathcal{X} \times \mathcal{O}$. Assume that this Markov chain has a unique stationary distribution, μ_h. Let $(X, o) \sim \mu_h$ be a draw from this stationary distribution. The following theorem shows that the value function of h can be computed from option returns and transient matrices.

Theorem 18. *Let $V_\theta(X) = \phi(X)^\top\theta$. Under the above conditions, if θ solves*

$$\mathbb{E}_{\mu_h}[\,(R^o(X) + (M^o\phi(X))^\top\theta - \phi(X)^\top\theta)\phi(X)\,] = 0, \tag{5.7}$$

where $\mathbb{E}_{\mu_h}$ is the expectation operator with respect to the stationary distribution μ_h. then V_θ is the TD(0) *approximation to the value function of h.*

Proof. Add $\#$ to the state space and extend ϕ to the new state space $\mathcal{X} \cup \{\#\}$ through $\phi(\#) = 0$. Let $X = X_0, R_1, X_1, R_2, \ldots$ be the state-reward sequence obtained while following h. Further, let $Q_1, Q_2, \ldots$ be an i.i.d., Bernoulli sequence such that $\mathcal{P}(Q_1 = 0) = \gamma$. Choose (Q_t) to be independent of all the other random variables. For $t \geq 1$, let $\hat{X}_t = \#$ if $Q_t = 1$ or $\hat{X}_{t-1} = \#$, otherwise let $\hat{X}_t = X_t$. We claim that

$$\mathbb{E}[\phi(X)\phi(\hat{X}_T)^\top] = \mathbb{E}[\gamma^T\phi(X)\phi(X_T)^\top]\,. \tag{5.8}$$

Indeed,

$$\begin{aligned}
\mathbb{E}\big[\phi(X)\phi(\hat{X}_T)^\top\big] &= \mathbb{E}\big[\phi(X)\phi(X_T)^\top\mathbb{I}_{\{Q_1=\ldots=Q_T=0\}}\big] \\
&= \mathbb{E}\Big[\,\mathbb{E}\big[\,\phi(X)\phi(X_T)^\top\,\mathbb{I}_{\{Q_1=\ldots=Q_T=0\}}\,\big|\,T, X_T, X\,\big]\,\Big] \\
&= \mathbb{E}\Big[\phi(X)\phi(X_T)^\top\mathbb{E}\big[\,\mathbb{I}_{\{Q_1=\ldots=Q_T=0\}}\,\big|\,T, X_T, X\,\big]\,\Big] \\
&= \mathbb{E}\big[\phi(X)\phi(X_T)^\top\gamma^T\,\big]\,,
\end{aligned}$$

where the first equality follows since $\phi(\#) = 0$ and the last equality follows by the choice of (Q_t). Thus, the claim is proved.

Now, with a similar reasoning, and using the strong Markov property, one can show that $V_\theta = \theta^\top\phi$ is a TD(0) approximation to V^h if and only if

$$\mathbb{E}[(R^o + V_\theta(\hat{X}_T) - V_\theta(X))\,\phi(X)] = 0\,. \tag{5.9}$$

(The proof is similar to proving that the value function of a high-level policy h satisfies a Bellman equation which uses the option models, see, e.g., Equation (8) of Sutton et al. (1999).) Let $\hat{\theta}$ be a solution to (5.7). Our goal is to show that

$\hat{\theta}$ satisfies (5.9). Clearly, it suffices to check if the second equality holds in the following series of equalities:

$$\begin{aligned}\mathbb{E}\big[\phi(X)\phi(\hat{X}_T)^\top\hat{\theta}\big] &= \mathbb{E}[\phi(X)V_{\hat{\theta}}(\hat{X}_T)] \\ &= \mathbb{E}\big[\phi(X)(M^o\phi(X))^\top\hat{\theta}\big] \\ &= \mathbb{E}\big[\phi(X)\phi(X)^\top(M^o)^\top\hat{\theta}\big]\,.\end{aligned}$$

After taking transposes, this boils down to checking

$$\mathbb{E}[\phi(\hat{X}_T)\phi(X)^\top] = \mathbb{E}[M^o\,\phi(X)\phi(X)^\top].$$

Now, by (5.8), $\mathbb{E}[\phi(\hat{X}_T)\phi(X)^\top] = \mathbb{E}[\gamma^T\phi(X_T)\phi(X)^\top] = \mathbb{E}\Big[\mathbb{E}\big[\,\gamma^T\phi(X_T)\phi(X)^\top\,|\,o\,\big]\Big]$. By the definition M^o (cf. (5.6)),

$$\mathbb{E}\big[\,\gamma^T\phi(X_T)\phi(X)\,\big|\,o\big] = \mathbb{E}\big[\,M^o\phi(X)\phi(X)^\top\,\big|\,o\,\big]\,.$$

hence,

$$\begin{aligned}\mathbb{E}\big[\phi(\hat{X}_T)\phi(X)^\top\big] &= \mathbb{E}\Big[\mathbb{E}\big[\,M^o\phi(X)\phi(X)^\top\,\big|\,o\,\big]\Big] \\ &= \mathbb{E}\Big[\,M^o\phi(X)\phi(X)^\top\,\Big]\,,\end{aligned}$$

thus finishing the proof.

□

Recall that Theorem 17 states that the U matrices can be used to compute the option returns given an arbitrary reward function. Thus given a reward function, the U and M matrices are all that one would need to solve the TD solution of the high-level policy. The merit of U and M is that they are *reward independent*: Once they are learned, they can be saved and used for different reward functions for different situations at different times.

5.4 Learning and Planning with UOMs

In this section we give incremental, TD-style algorithms for learning and planning with linear UOMs. We start by describing the learning of UOMs while following some high-level policy h, and then describe a Dyna-like algorithm that estimates the value function of h with learned UOMs and an immediate reward model.

5.4.1 Learning Linear UOMs

Assume that we are following a high-level policy h over a set of options $\mathcal{O}$, and that we want to estimate linear UOMs for the options in $\mathcal{O}$. Let the trajectory generated by following this high-level policy be $\ldots, X_t, Q_t, O_t, A_t, X_{t+1}, Q_{t+1}, \ldots$. Here, $Q_t = 1$ is the indicator for the event that option O_{t-1} is terminated at state X_t and so $O_t \sim h(X_t, \cdot)$. Also, when $Q_t = 0$, $O_t = O_{t-1}$. Upon the transition from X_t to X_{t+1}, Q_{t+1}, the matrix U^{O_t} is updated as follows:

$$\begin{aligned}
U^{O_t}_{t+1} &= U^{O_t}_t + \eta^{O_t}_t \, \delta_{t+1} \, Z^\top_{t+1}, \quad \text{where} \\
\delta_{t+1} &= \phi(X_t) + \gamma U^{O_t}_t \phi(X_{t+1}) \mathbb{I}_{\{Q_{t+1}=0\}} - U^{O_t}_t \phi(X_t), \\
Z_{t+1} &= \phi(X_t) + \gamma \lambda Z_t \mathbb{I}_{\{Q_t=0\}}
\end{aligned}$$

and $\eta^{O_t}_t \geq 0$ is the learning-rate at time t associated with option O_t. Note that when option O_t is terminated the temporal difference δ_{t+1} is modified so that the next predicted value is zero. Also, the eligibility trace is reset when a new option is selected.

The (M^o) matrices are updated using the least-mean square algorithm. In particular, matrix M^{O_t} is updated when option O_t is terminated at time $t+1$, i.e., when $Q_{t+1} = 1$. In the update we need the feature $(\tilde{\phi}_\cdot)$ of the state which was visited at the time option O_t was selected and also the time elapsed since this time $(\tau_\cdot)$:

$$\begin{aligned}
M^{O_t}_{t+1} &= M^{O_t}_t + \tilde{\eta}^{O_t}_t \mathbb{I}_{\{Q_{t+1}=1\}} \left\{ \gamma^{\tau_t} \phi(X_{t+1}) - M^{O_t}_t \tilde{\phi}_t \right\} \tilde{\phi}^\top_t, \\
\tilde{\phi}_{t+1} &= \mathbb{I}_{\{Q_{t+1}=0\}} \tilde{\phi}_t + \mathbb{I}_{\{Q_{t+1}=1\}} \phi(X_{t+1}), \\
\tau_{t+1} &= \mathbb{I}_{\{Q_{t+1}=0\}} \tau_t + 1 .
\end{aligned}$$

These variables are initialized to $\tau_0 = 0$ and $\tilde{\phi}_0 = \phi(X_0)$.

Algorithm 4 shows the pseudo-code of learning UOMs.

Theorem 19. *Assume that the stationary distribution of h is unique, all options in $\mathcal{O}$ terminate with probability one and that all options in $\mathcal{O}$ are selected at some state with positive probability.*[4] *If the step-sizes of the options are decreased towards*

[4]Otherwise, we can drop the options in $\mathcal{O}$ which are never selected by h.

Algorithm 4 Dyna-Universal: Learning UOMs.

```
Input: A feature mapping $\phi$, a policy $h$ over a set of options.
Output: Linear UOMs for the options.
parameters: $\eta$, the modelling step-size.
Initialize $U^o$, $M^o$ for all the options
/* run the following for each episode */
Initialize $\phi_0$
Do at each time step
    Select an option $o$ according to policy $h$. If terminating, set $\phi_{t+1} = 0$; otherwise, take an action according to the policy associated with $o$, and observe the next feature vector $\phi_{t+1}$
    $U^o \leftarrow U^o + \eta\left[\phi_t + \gamma U^o \phi_{t+1} - U^o \phi_t\right]\phi_t^\top$
    If terminating, update $M^o$ for the $\phi_k$ at which $o$ was started:
    $M^o \leftarrow M^o + \eta[\gamma^{t-k}\phi_t - M^o \phi_k]\phi_k^\top$
```

zero so that the Robbins-Monro conditions hold for them, [5] *then for any* $o \in \mathcal{O}$, $M_t^O \to M^o$ *and* $U_t^o \to U^o$ *with probability one, where* (U^o, M^o) *are defined in the previous section.*

Proof. The proof can be reduced to studying the individual (U_t^o, M_t^o) pairs as follows: Using an appropriate sequence of stopping times and relying on the strong Markov property, one can argue that the updates for all options can be viewed on their own. Now for a single option, the convergence of M_t^o follows since it is an LMS update rule. The convergence of U_t^o essentially follows from the previous results in Section 6.3.4 of (Bertsekas and Tsitsiklis, 1996) (for on-line updating, one should combine this argument with that of (Jaakkola et al., 1994, Theorem 3)).

□

5.4.2 Learning Reward Models

In conventional settings, a single reward signal will be contained in the trajectory when following the high level policy, $\ldots, X_t, Q_t, O_t, A_t, R_{t+1}, X_{t+1}, Q_{t+1}, \ldots$. We can learn an immediate reward model, ω^{O_t}, for this reward signal with a least-mean square update rule:

$$\omega_{t+1}^{O_t} = \omega_t^{O_t} + \tilde{\eta}_t^{O_t} \mathbb{I}_{\{Q_{t+1}=0\}} \left\{ R_{t+1} - \omega^{O_t \top} \phi(X_t) \right\} \phi(X_t)\,.$$

[5]That is, the sum of the step-sizes diverges, while the sum of their squares converges.

In other settings, the immediate reward model can be constructed in different ways. For example, more than one reward signal can be of interest, so multiple immediate reward models can be learned in parallel. Moreover, such additional reward signals might be provided at any time. In some settings, the immediate reward model for a reward function can be provided directly from knowledge of the environment and features where the immediate reward model is independent of the option.

5.4.3 Policy Evaluation with UOMs and Reward Models

Consider the process of policy evaluation for a high-level policy over options from a given set of UOMs when learning a reward model. When starting from a state X with feature vector $\phi(X)$ and following option o, the return $R^o(X)$ is estimated from the reward model ω^o and the expected feature occupancy matrix U^o by $R^o(X) \approx (\omega^o)^\top U^o \phi(X)$. The TD(0) approximation to the value function of a high-level policy h can then be estimated online from Theorem 18. Interleaving updates of the reward model learning with these planning steps for h gives a Dyna-like algorithm.

Algorithm 5 shows the pseudo code of evaluating a high level policy using Dyna-style planning architecture.

Algorithm 5 Dyna-Universal: linear Dyna with UOMs for evaluating a high-level policy h.

```
Input: Linear UOMS and policy $h$.
Output: TD solution $\theta$ for the value function of $h$.
parameters: $\alpha$, the planning step-size.
Initialize $\omega^o$ for all the options, and the policy weight parameter vector $\theta$
/* run the following for each episode */
Initialize $\phi_0$
Do at each time step
  Select an option $o$. If not terminating, take an action according to the policy
associated with the option and observe a reward $R_{t+1}$
  $\omega^o \leftarrow \omega^o + \eta \left[R_{t+1} - \phi_t^\top \omega^o\right] \phi_t$
  Do planning for $\tau$ times
    sample a feature vector $\phi$
    select an option $\epsilon$ according to policy $h$ for $\phi$
    /* Update with the UOM$(U^\epsilon$, $M^\epsilon)$ and $f^\epsilon$ of the option $\epsilon$ */
    $R \leftarrow (U^\epsilon \phi)^\top \omega^\epsilon$
    $\phi' \leftarrow M^\epsilon \phi$
    $\theta \leftarrow \theta + \alpha(R + \phi'^\top \theta - \phi^\top \theta)\phi$
```

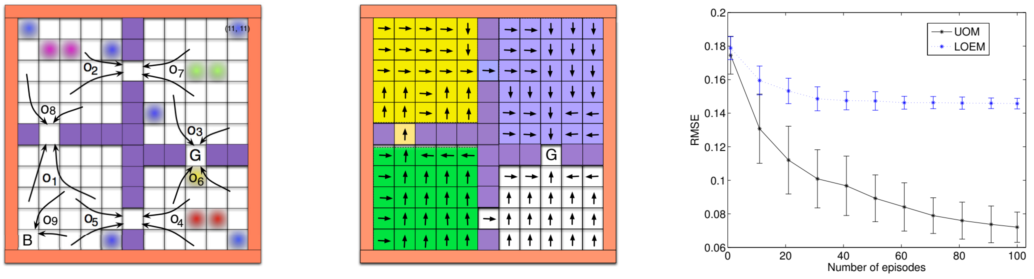

Figure 5.1: (a: left) A Star Craft local mission map, consisting of four bridged regions, and nine options for the mission. (b: middle) A high-level policy $h =< o_1, o_2, o_3, o_6 >$ initiates the options in the regions, with deterministic policies in the regions as given by the arrows: o_1 (green), o_2 (yellow), o_3 (purple), and o_6 (white). Outside these regions, the policies select actions uniformly at random. (c: right) The expected performance of different units can be learned by simulating trajectories (with the standard deviation shown by the bars), and the UOM method reduces the error faster than the LOEM method.

5.5 Empirical Results

In this section, we provide empirical results on choosing game units to execute specific policies in a simplified real-time strategy game and recommending articles in a large academic database with more than one million articles.

We compare the UOM method with a method of Sorg and Singh (2010), who introduced the *linear-option expectation model (LOEM)* that is applicable for evaluating a high-level policy over options. Their method estimates (M^o, b^o) from experience, where b^o is equal to $(U^o)^\top \omega^o$ in our formulation. This term b^o is the expected return from following the option, and can be computed incrementally from experience once a reward signal or an immediate reward model are available.

A simplified Star Craft 2 mission. We examined the use of the UOMs and LOEMs for policy evaluation in a simplified variant of the real-time strategy game Star Craft 2, where the task for the player was to select the best game unit to move to a particular goal location. We assume that the player has acces to a black-box game simulator. There are four game units with the same constant dynamics. The internal status of these units dynamically changes during the game and this affects the reward they receive in enemy controlled territory. We evaluated these units, when their rewards are as listed in the table below (the rewards are associated with

	Game Units			
Enemy Locations	*Battlecruiser*	*Reapers*	*Thor*	*SCV*
fortress (yellow)	0.3	-1.0	1.0	-1.0
ground forces (green)	1.0	0.3	1.0	-1.0
viking (red)	-1.0	-1.0	1.0	-1.0
cobra (pink)	1.0	0.5	-1.0	-1.0
minerals (blue)	0	0	0	1.0

Table 5.1: The reward functions of all our agents (column; first letter capitalized). On facing an enemy agent or minerals (a row), a reward is given according to the corresponding entry of the table. All the other rewards are zero.

the previous state and are not action-contingent). A game map is shown in Figure 5.1 (a). The four actions could move a unit left, right, up, or down. With probability $2/3$, the action moved the unit one grid in the intended direction. With probability $1/3$, the action failed, and the agent was moved in a random direction chosen uniformly from the other three directions. If an action would move a unit into the boundary, it remained in the original location (with probability one). The discount factor was 0.9. Features were a lookup table over the 11×11 grid. For all algorithms, only one step of planning was applied per action selection. The planning step-size for each algorithm was chosen from $0.001, 0.01, 0.1, 1.0$. Only the best one was reported for an algorithm. All data reported were averaged over 30 runs.

We defined a set of nine options and their corresponding policies, shown in Figure 5.1 (a), (b). These options are specified by the locations where they terminate, and the policies. The termination location is the square pointed to by each option's arrows. Four of these are "bridges" between regions, and one is the position labelled "B" (which is the player's base at position $(1, 1)$). Each of the options could be initiated from anywhere in the region in which the policy was defined. The policies for these options were defined by a shortest path traversal from the initial location to the terminal location, as shown in the figure. These policies were not optimized for the reward functions of the game units or the enemy locations.

To choose among units for a mission in real time, a player must be able to efficiently evaluate many options for many units, compute the value functions of the various high-level policies, and select the best unit for a particular high-level goal. A high-level policy for dispatching the game units is defined by initiating different options from different states. For example, a policy for moving units from the base "B" to position "G" can be, $h =< o_1, o_2, o_3, o_6 >$. Another high-level policy could move another unit from room 2 to "G" by a different route with $h' =< o_8, o_5, o_6, o_3 >$.

We evaluated policy h for the Reaper unit above using UOMs and LOEMs. We first pre-learned the U^o and M^o models using the experience from 3000 trajectories. Using a reward function that is described in the above table, we then learned ω^o for the UOM and and b^o for the LEOM over 100 simulated trajectories, and concurrently learned θ. As shown in Figure 5.1 (c), the UOM model learns a more accurate estimate of the value function from fewer episodes, when the best performance is taken across the planning step size. Learning ω^o is easier than learning b^o because the stochastic dynamics of the environment is factored out through the pre-learned U^o. These constructed value functions can be used to select the best game unit for the task of moving to the goal location.

This approach is computationally efficient for multiple units. We compared the computation time of LOEMs and UOMs with linear Dyna on a modern PC with an Intel 1.7GHz processor and 8GB RAM in a MATLAB implementation. Learning U^o took 81 seconds. We used a recursive least-squares update to learn M^o, which took 9.1 seconds. Thus, learning an LOEM model is faster than learning a UOM for a single fixed reward function, but the UOM can produce an accurate option return quickly for each new reward function. Learning the value function incrementally from the 100 trajectories took 0.44 seconds for the UOM and 0.61 seconds for the LOEM. The UOM is slightly more efficient as ω^o is more sparse than b^o, but it is substantially more accurate, as shown in Figure 5.1 (c). We evaluated all the units and the results are similar.

Article recommendation. Recommending relevant articles for a given user query can be thought of as predicting an expected return of an option for a dynamically specified reward model. Ranking an article as a function of the links between articles in the database has proven to be a successful approach to article recommendation, with PageRank and other link analysis algorithms using a random surfer model (Page et al., 1998). We build on this idea, by mapping a user query to a reward model and pre-specified option for how a reader might transition between articles. The ranking of an article is then the expected return from following references in articles according to the option. Consider the policy of performing a random-walk between articles in a database by following a reference from an article that is selected uniformly at random. An article receives a positive reward if it matches a user query (and is otherwise zero), and the value of the article is the expected discounted return from following the random-walk policy over articles. More focused reader policies can be specified as following references from an article with a common author or keyword.

We experimented with a collection from DBLP that has about 1.5 million articles, 1 million authors, and 2 millions citations (Tang et al., 2008). We assume that a user query q is mapped directly to an option o and an immediate reward model f_q^o. For simplicity in our experiment, the reward models are all binary, with three non-zero features drawn uniformly at random. In total we used about 58 features, and the discount factor was 0.9. There were three policies. The first followed a reference selected uniformly at random, the second selected a reference written by an author of the current article (selected at random), and the third selected a reference with a keyword in common with the current article. Three options were defined from these policies, where the termination probability beta was 1.0 if no suitable outgoing reference was available and 0.25 otherwise. High-level policies of different option sequences could also be applied, but were not tested here. We used bibliometric features for the articles extracted from the author, title, venue fields.

We generated queries q at random, where each query specified an associated option o and an option-independent immediate reward model $\omega_q^o = f_q$. We then computed their value functions. The immediate reward model is naturally constructed for these problems, as the reward comes from the starting article based on its features, so it is not dependent on the action taken (and thus not the option). This approach is appropriate in article recommendation as a query can provide both terms for relevant features (such as the venue), and how the reader intends to follow references in the paper. For the UOM based approach we pre-learned U^o, and then computed $U^o\omega_q^o$ for each query. For the LOEM approach, we learned a b_q for each query by simulating 3000 trajectories in the database (the simulated trajectories were shared for all the queries). The computation time (in seconds) for the UOM and LOEM approaches are shown in the table below.

Number of reward functions	10	100	500	1,000	10,000
LOEM	0.03	0.09	0.47	0.86	9.65
UOM	0.01	0.04	0.07	0.12	1.21

The table shows that LOEM is less computationally efficient than UOMs for ranking articles. LOEM and other conventional option models do not scale to handle many real-time queries on large databases. The UOMs proposed in this thesis scale to handle large numbers of reward functions and are especially suitable for large online systems.

5.6 Discussion and Conclusion

We proposed a new way of modelling options in both tabular representation and linear function approximation, which is called the universal option model. We showed how to learn UOMs and how to use them to construct the TD solution of option returns and value functions of policies, and prove their theoretical guarantees. It is important to emphasize that the focus of this thesis is not the extension of linear Dyna to options which is the work of LEOM (Sorg and Singh, 2010), but a new computational device by which learning new value functions for existing options becomes efficient, both computationally and information theoretically. UOMs are advantageous in large online systems. Estimating the return of an option given a new reward function with the UOM of the option is reduced to a one-step regression. Computing option returns dependent on many reward functions in large online games and search systems using UOMs is much faster than using previous methods for learning option models.

Chapter 6

Conclusions

6.1 Summary

In this thesis, we proposed a model-based API algorithm called LAM-API in Chapter 3, which first learns linear action modes and then use them for off-policy evaluation and control. In Chapter 4, we proposed a pseudo-MDP framework in which we learn an optimal policy in the pseudo-MDP and then pull it back to the original MDP. In Chapter 5, we studied evaluating high-level policies with options in environments where there are multiple reward functions, and proposed using universal option models to construct returns efficiently.

6.2 Contributions

There are three contributions in this thesis:

- Our experiments show that our LAM-API performs comparably to LSPI (arguably one of the most powerful RL algorithms), and often converges much quicker (Yao and Szepesvári, 2012).
- In the pseudo-MDPs framework, we give a performance error bound for the approach. Surprisingly, the error bound shows that the quality of the policy derived from an optimal policy of the pseudo-MDP is governed only by the policy evaluation errors of an optimal policy in the original MDP and the "pull-back" policy of an optimal policy in the pseudo-MDP. This result gives a direction for model-based reinforcement learning: an exhaustively accurate

model is not necessary in order to learn a good policy—it only needs to be accurate in two projected directions (some terms in the error bound). This result is interesting because performance error bound of the recent kernel embedding AVI (Grünewälder et al., 2012) can be derived using our error bound (Yao et al., 2014a).

The pseudo-MDP framework not only includes the kernel embedding model (Grünewälder et al., 2012) but also opens the door to new models. We propose a least-squares approach and a constrained optimization approach of learning factored linear models, which can be used for AVI. The least-squares approach without normalization can possible diverge while with normalization (a standard practice in kernel-based methods) it is guaranteed to converge but possibly to bad policies even with good features. The constrained optimization approach is guaranteed to converge and give good policies in our experiments. We explored a feature learning method based on our error bound, which can find good features in a simple experiment.

- The universal option model enables a very efficient and simple generation of option returns. We provide algorithms of learning this model as well as planning algorithms for generating returns and value functions. We also prove the convergence of these algorithms (Yao et al., 2014b). In a simulated game playing scenario, we show that selecting the best unit in a team for executing a task can benefit in computation efficiency from universal option models. In article recommendation, we show that recommending articles to users with personal preference reward functions has a significant computation advantage using universal option models over traditional models.

6.3 Open Questions

Our LAM-API algorithm suggests actions models are effective and efficient for off-policy learning and control. It would be interesting to build other (such as nonlinear) action models, from which we may have more accurate projections and hence better performance.

In the pseudo-MDP framework, the constrained optimization model we proposed is guaranteed to converge, but there are computation scalability issues that remain to be addressed. The procedures that are based on the dual representation that we also use need to work and sometimes invert $n \times n$ matrices for a sample size of n. This is clearly infeasible even for medium size data. In the supervised learning literature much research went into speeding up these operations, such as Nyström's method and variants (Le et al., 2013; Hsieh et al., 2014). It will be interesting to incorporate these into our setting. In the RL literature, there have been advancements for scenarios where ψ satisfies certain structural assumptions (Kveton and Theocharous, 2013), or sample subsampling (Kveton and Theocharous, 2012), which we can also try to incorporate.

Bibliography

Abbeel, P., Coates, A., and Ng, A. Y. (2010). Autonomous helicopter aerobatics through apprenticeship learning. *Int. J. Rob. Res.*, 29(13):1608–1639.

Antos, A., Szepesvári, C., and Munos, R. (2008). Learning near-optimal policies with Bellman-residual minimization based fitted policy iteration and a single sample path. *Machine Learning*, 71(1):89–129.

Baird, L. C. (1995). Residual algorithms: Reinforcement learning with function approximation. In A. Prieditis, S. R., editor, *Proceedings of the 12th International Conference on Machine Learning (ICML 1995)*, pages 30–37, San Francisco, CA, USA. Morgan Kaufmann.

Barto, A. and Duff, M. (1994). Monte carlo matrix inversion and reinforcement learning. In *In Advances in Neural Information Processing Systems 6*, pages 687–694. Morgan Kaufmann.

Bertsekas, D. P., Borkar, V. S., and Nedič, A. (2004). Improved temporal difference methods with linear function approximation. In Si, J., Barto, A. G., Powell, W. B., and Wunsch II, D., editors, *Learning and Approximate Dynamic Programming*, chapter 9, pages 235–257. IEEE Press.

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

Boyan, J. A. (2002). Technical update: Least-squares temporal difference learning. *Machine Learning*, 49:233–246.

Boyan, J. A. and Moore, A. W. (1995). Generalization in reinforcement learning: Safely approximating the value function. In Tesauro, G., Touretzky, D., and Leen, T., editors, *Advances in Neural Information Processing Systems 7 (NIPS-7)*, pages 369–376, Cambridge, MA, USA. MIT Press.

Boyd, S., Parikh, N., Chu, E., Peleato, B., and Eckstein, J. (2011). Distributed optimization and statistical learning via the alternating direction method of multipliers. *Found. Trends Mach. Learn.*, 3(1):1–122.

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

Choi, J., Laibson, D., Madrian, B., and Metrick, A. (2007). Reinforcement learning and savings behavior. Technical report, CF Working Paper 09-01, Yale.

Coates, A., Abbeel, P., and Ng, A. Y. (2008). Learning for control from multiple demonstrations. *ICML*.

Crites, R. H. and Barto, A. (1995). Improving elevator performance using reinforcement learning. *NIPS*.

Duchi, J., Shalev-Shwartz, S., Singer, Y., and Chandra, T. (2008). Efficient projections onto the l1-ball for learning in high dimensions. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 272–279, New York, NY, USA. ACM.

Ernst, D., Geurts, P., and Wehenkel, L. (2005). Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6:503–556.

Geramifard, A., Bowling, M. H., Zinkevich, M., and Sutton, R. S. (2007). iLSTD: Eligibility traces and convergence analysis. In Schölkopf, B., Platt, J., and Hoffman, T., editors, *Advances in Neural Information Processing Systems 19 (NIPS-19)*, pages 441–448, Cambridge, MA, USA. MIT Press. (December 4–7, 2006).

Grünewälder, S., Lever, G., Baldassarre, L., Pontil, M., and Gretton, A. (2012). Modelling transition dynamics in MDPs with RKHS embeddings. In Langford, J. and Pineau, J., editors, *Proceedings of the 29th International Conference on Machine Learning (ICML 2012)*, volume 28 of *JMLR Workshop and Conference Proceedings*.

Hsieh, C.-J., Si, S., and Dhillon, I. S. (2014). Fast prediction for large-scale kernel machines. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 27*, pages 3689–3697. Curran Associates, Inc.

Jaakkola, T., Jordan, M., and Singh, S. (1994). On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, 6(6):1185–1201.

Koller, D. and Parr, R. (2000). Policy iteration for factored mdps. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI-00*, pages 326–334. Morgan Kaufmann.

Kveton, B. and Theocharous, G. (2012). Kernel-based reinforcement learning on representative states. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada.*

Kveton, B. and Theocharous, G. (2013). Structured kernel-based reinforcement learning. In desJardins, M. and Littman, M. L., editors, *AAAI*. AAAI Press.

Lagoudakis, M. and Parr, R. (2003). Least-squares policy iteration. *Journal of Machine Learning Research*, 4:1107–1149.

Le, Q., Sarlós, T., and Smola, A. (2013). Fastfood–approximating kernel expansions in loglinear time. *JMLR: W&CP – Proceedings of The 30th International Conference on Machine Learning*, 28(3):1391–1399.

Li, L., Littman, M. L., and Mansley, C. R. (2009). Online exploration in least-squares policy iteration. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '09, pages 733–739, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

Munos, R. and Szepesvári, Cs. (2008). Finite-time bounds for fitted value iteration. *Journal of Machine Learning Research*, 9:815–857.

Nedič, A. and Bertsekas, D. P. (2003). Least squares policy evaluation algorithms with linear function approximation. *Discrete Event Dynamic Systems*, 13(1):79–110.

Ng, A. Y. and Russell, S. J. (2000). Algorithms for inverse reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, pages 663–670, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Ormoneit, D. and Sen, S. (2002). Kernel-based reinforcement learning. *Machine Learning*, 49:161–178.

Page, L., Brin, S., Motwani, R., and Winograd, T. (1998). The PageRank citation ranking: Bringing order to the web. Technical report, Stanford University.

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 Cohen, W., McCallum, A., and Roweis, S., editors, *Proceedings of the 25th International Conference Machine Learning (ICML 2008)*, volume 307 of *ACM International Conference Proceeding Series*, pages 752–759, New York, NY, USA. ACM.

Powell, W. B., George, A., Simão, H., Scott, W., Lamont, A., and Stewart, J. (2012). Smart: A stochastic multiscale model for the analysis of energy resources, technology, and policy. *INFORMS J. on Computing*, 24(4):665–682.

Precup, D. (2000). *Temporal Abstraction in Reinforcement Learning*. PhD thesis, University of Massachusetts, Amherst.

Puterman, M. (1994). *Markov Decision Processes — Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY.

Richardson, M. and Domingos, P. (2002). The Intelligent Surfer: Probabilistic Combination of Link and Content Information in PageRank. In *Advances in Neural Information Processing Systems 14*. MIT Press.

Silver, D., Sutton, R. S., and Mueller, M. (2007). Reinforcement learning of local shape in the game of Go. *IJCAI*.

Sorg, J. and Singh, S. (2010). Linear options. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 1 - Volume 1*, AAMAS '10, pages 31–38, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

Sutton, R. (1984). *Temporal credit assignment in reinforcement learning*. PhD thesis, Department of Computer Science, University of Massachusetts, Amherst, MA 01003.

Sutton, R. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning*, pages 216–224. San Mateo, CA. DYNA.

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

Sutton, R. S., Maei, H. R., Precup, D., Bhatnagar, S., Silver, D., Szepesvári, Cs., and Wiewiora, E. (2009a). Fast gradient-descent methods for temporal-difference learning with linear function approximation. In Danyluk, A., Bottou, L., and Littman, M., editors, *Proceedings of the 26th Annual International Conference on Machine Learning (ICML 2009)*, volume 382 of *ACM International Conference Proceeding Series*, pages 993–1000, New York, NY, USA. ACM.

Sutton, R. S., Precup, D., and Singh, S. P. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112:181–211.

Sutton, R. S., Szepesvári, Cs., Geramifard, A., and Bowling, M. H. (2008). Dyna-style planning with linear function approximation and prioritized sweeping. In McAllester, D. and Myllymäki, P., editors, *Proceedings of the 24th Conference in Uncertainty in Artificial Intelligence (UAI'08)*, pages 528–536, Corvallis, OR, USA. AUAI Press.

Sutton, R. S., Szepesvári, Cs., and Maei, H. R. (2009b). A convergent $O(n)$ temporal-difference algorithm for off-policy learning with linear function approximation. In Koller, D., Schuurmans, D., Bengio, Y., and Bottou, L., editors, *Advances in Neural Information Processing Systems 21 (NIPS-21)*, pages 1609–1616. Curran Associates. (December 8–10, 2008).

Syed, U. A. (2010). *Reinforcement Learning Without Rewards*. PhD thesis, Princeton University.

Szepesvári, C. (2001). Efficient approximate planning in continuous space Markovian decision problems. *AI Communications*, 13(3):163–176.

Szepesvári, Cs. (2010). *Algorithms for Reinforcement Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers.

Tang, J., Zhang, J., Yao, L., Li, J., Zhang, L., and Su, Z. (2008). Arnetminer: extraction and mining of academic social networks. *SIGKDD*, pages 990–998.

Tesauro, G. (1994). TD-Gammon, a self-teaching backgammon program achieves master-level play. *Neural Computation*, 6:215–219.

Tsitsiklis, J. N. and Van Roy, B. (1996). Feature-based methods for large scale dynamic programming. *Machine Learning*, 22:59–94.

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

Van Roy, B. (2006). Performance loss bounds for approximate value iteration with state aggregation. *Math. Oper. Res.*, 31(2):234–244.

Yao, H., Bhatnagar, S., and Szepesvári, Cs. (2009a). LMS-2: Towards an algorithm that is as cheap as LMS and almost as efficient as RLS. In *Proc. of the 48th IEEE Conference on Decision and Control (CDC-09)*, pages 1181–1188.

Yao, H. and qiang Liu, Z. (2008). Preconditioned temporal difference learning. In Cohen, W. W., Mccallum, A., and Roweis, S. T., editors, *Proceedings of the 25th International Conference on Machine Learning (ICML-08)*, pages 1208–1215.

Yao, H., Sutton, R. S., Bhatnagar, S., Diao, D., and Szepesvári, C. (2009b). Multi-step dyna planning for policy evaluation and control. In *Advances in Neural Information Processing Systems 22: 23rd Annual Conference on Neural Information Processing Systems 2009. Proceedings of a meeting held 7-10 December 2009, Vancouver, British Columbia, Canada.*, pages 2187–2195.

Yao, H., Szepesvári, C., Pires, B. A., and Zhang, X. (2014a). Pseudo-mdps and factored linear action models. In *IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL), 2014*, pages 1–9. IEEE.

Yao, H., Szepesvari, C., Sutton, R., Bhatnagar, S., and Modayil, J. (2014b). Universal option models. In *Advances in Neural Information Processing Systems 27*, pages 990–998.

Yao, H. and Szepesvári, Cs. (2012). Approximate policy iteration with linear action models. In Hoffmann, J. and Selman, B., editors, *Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI-12)*, pages 1212–1217. AAAI Press.