Automatic basis function construction for reinforcement learning and approximate dynamic programming

Philipp W. Keller

Master of Science in Computer Science

School of Computer Science

McGill University

Montreal, Quebec

February 2008

A Thesis submitted to McGill University
in partial fulfillment of the requirements for the degree of
Master of Science in Computer Science

ACKNOWLEDGEMENTS

Most of all, I thank my advisors, Doina Precup and Shie Mannor. Doina was the first to get me excited about the possibilities of artificial intelligence, first introduced me to research, and taught me a great deal of the knowledge that went into this thesis, besides much else. Shie, of course, provided the idea at the root of this work, was never without ideas when the going got tough, and also taught me a great deal. But I thank him most of all for broadening my perspectives at a crucial point in my academic life, and helping me decide on a future career path.

I will be eternally grateful to both Doina and Shie for their help and teaching. But I would never have finished this thesis, had they not been so understanding, patient and supportive while I went through a very difficult time personally. They never gave up on me, and didn't let me give up on myself, for which I owe them another great debt.

I also thank the many members of, and visitors to, the Reasoning and Learning Lab, on whom I could always count for support, friendship, and pleasant distraction. They are the people who made my time there enjoyable, and who made my academic and personal challenges seem manageable. I thank the entire faculty of the School of Computer Science, who have collectively taught me much of what I know, and of whom many members have given me invaluable suggestions and personal support at the right moment.

Finally, I thank my family for always remaining supportive of my plans and concerned for my happiness despite everything they were going through while I was completing my degree.

ABSTRACT

We address the problem of automatically constructing basis functions for linear approximation of the value function of a Markov decision process (MDP). Our work builds on results by Bertsekas and Castañon (1989) who proposed a method for automatically aggregating states to speed up value iteration. We propose to use neighbourhood component analysis , a dimensionality reduction technique created for supervised learning, in order to map a high-dimensional state space to a low-dimensional space, based on the Bellman error, or on the temporal difference (TD) error. We then place basis functions in the lower-dimensional space. These are added as new features for the linear function approximator. This approach is applied to a high-dimensional inventory control problem, and to a number of benchmark reinforcement learning problems.

ABRÉGÉ

Nous addressons la construction automatique de fonctions base pour l'approximation linéaire de la fonction valeur d'un processus de décision Markov. Cette thèse se base sur les résultats de Bertsekas et Castañon (1989), qui ont proposé une méthode pour automatiquement grouper des états dans le but d'accélérer la programmation dynamique. Nous proposons d'utiliser une technique récente de réduction de dimension afin de projeter des états en haute dimension dans un espace à basse dimension. Nous plaçons alors des fonctions base radiales dans ce nouvel espace. Cette technique est appliquée à plusieurs problèmes de référence standards pour les algorithmes d'apprentissage par renforcement, ainsi qu'à un problème de contrôle d'inventaire en haute dimension.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1
Introduction

A reinforcement learning (RL) agent adapts its behaviour in an unknown environment in response to reward signals generated as a result of its actions. The agent's goal is to learn a policy maximizing its long-term reward. As a sub-area of machine learning, RL differs from supervised and unsupervised learning in that no explicit set of training data is used to learn fixed relationships among the data. Instead, the agent's behavious is updated in response to data and reward signals resulting directly from interactions with the environment.

Markov Decision Processes (MDPs) are generally used to model the environment. In value-based RL, the agent maintains estimates of the total expected return given a behaviour policy when starting in each state. In this context, statistical and dynamic programming methods are used to compute the estimates and update the agent's policy. A closely related field is optimal stochastic control.

In environments with a large discrete state space, or a continuous state space, function approximation methods must be used to represent value functions since it is not possible to store a value for each state. The majority of current research in reinforcement learning relies on linear function approximators which represent the value of a state as a linear combination of basis functions. The coefficients are learned from data, but in most work to date, the basis functions themselves are specified by the designer of the system.

This is a significant barrier to the application of RL since designing an approximator for a given task can be difficult and time consuming. Moreover, even with careful engineering, continuous problems with more than 10-12 dimensions cannot be handled by current techniques [18]. Recently, a significant amount of work has been devoted to constructing bases for value functions automatically[13, 14, 17, 22, 28]. While these methods look very promising, there is no empirical or theoretical evidence to date of their efficiency in large problems.

This thesis proposes a different approach for constructing basis functions automatically. The state space of the MDP is projected to a lower dimensional space based on estimates of error in the current value function approximation. New basis functions are defined in this low-dimensional space to compensate for the errors. The function approximator is trained as usual, and the process is repeated. The novel properties of the algorithm are:

1. The ability to deal with high-dimensional state representations: the state of many realistic problems is naturally described by relatively large real-valued vectors. Dimensionality reduction is used to identify relevant features of the state space, and thus avoid the "curse of dimensionality" faced when working in the original space.
2. An efficient heuristic RBF selection procedures to build easily evaluated basis functions suitable for use in reinforcement learning with large sample sizes.
3. Rather than directly using the Bellman error as a basis function as in previous work, a lower-dimensional policy evaluation problem is solved to yield more

appropriate basis functions for the original MDP. Choosing basis functions for the low-dimensional MDP is tractable.

4. Instead of fixing the approximate Bellman error at a given iteration as a basis function, the set of RBFs used to represent *all* the Bellman errors seen so far are used as the basis functions for the value function approximation.
5. A pruning algorithm is used to eliminate redundant RBFs, discard RBFs which become obsolete as the approximation to the value function changes, and bound the total number of RBFs used in the approximator.

This thesis is organised as follows. Chapter 2 summarizes the necessary background and notation for MDPs and RL algorithms. Chapter 3 discusses various exitsting automatic basis function selection approaches and motivates the approach taken here. Chapter 4 describes neighbourhood component analysis (NCA), the dimensionality reduction method used in the algorithm. Chapter 5 presents the algorithm for automatic basis function construction. Chapter 6 outlines experimental results obtained for a discrete MDP, a continuous-state control problem, and an inventory control problem. Finally, Chapter 7 discusses recent related approaches and concludes with some proposed future work in the area.

CHAPTER 2
Background and Notation

This chapter reviews relevant background on Markov Decision Processes and standard reinforcement learning algorithms. Extensive treatments of dynamic programming and reinforcement learning are provided by Bertsekas [1], and Sutton and Barto [24] respectively.

2.1 Markov Decision Processes

A *Markov Decision Process* (MDP) is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ where

- $\mathcal{S}$ is the space of possible states of the environment,
- $\mathcal{A}$ is a set of actions available to the agent at any time
- $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ defines a conditional probability distribution over state transitions given an action,
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a reward function assigning immediate rewards to an action taken in a given state, and
- $\gamma \in (0, 1)$ is a discount factor.

A stationary *policy* $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ defines the probability of selecting each action in each state. The *value function* $V^\pi : \mathcal{S} \to \mathbb{R}$ for a given policy π is defined as

$$V^\pi(s) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k \, \mathcal{R}(s_{t+k}, a_{t+k}) \middle| \, s_t = s \right],$$

where s_t and a_t represents the state and action at time t when actions are selected according to the policy π. The value function assigns to each state the expected discounted return of the agent when starting in that state and following the fixed policy.

Similarly, the *action-value function* $Q^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is defined as

$$Q^\pi(s,a) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k \mathcal{R}(s_{t+k}, a_{t+k}) \middle| s_t = s, a_t = a \right].$$

Thus

$$V^\pi(s) = \int_{a \in \mathcal{A}} \pi(s,a) Q^\pi(s,a).$$

The goal of the agent is to find a policy maximizing it's expected discounted return for each state. Such a policy yields the unique *optimal value function*

$$V^*(s) = \max_\pi V^\pi(s) = \max_{a \in \mathcal{A}} Q^*(s,a) \quad \forall s \in \mathcal{S}.$$

A deterministic *optimal policy* is to act greedily with respect to the optimal value function V^*, i.e.

$$\pi^*(s_t) \in \arg\max_{a \in \mathcal{A}} \mathbb{E}\left[\mathcal{R}(s_t, a) + \gamma V^*(s_{t+1})\right],$$

where, by abuse of notation, $\pi^*(s)$ denotes the *deterministic* action to be taken in state s.

A standard approach to finding the optimal value function is *policy iteration*, where a policy π is fixed and the corresponding value function V^π is computed. This value function is then used to improve the policy, and the process is repeated. This thesis is mainly concerned with the *policy evaluation* step, algorithms for which are discussed in the next section.

Without loss of generality the rewards in this and the following discussion may be replaced with costs. In such a case the optimal policy minimizes rather than maximizes the action-value function.

2.2 Dynamic Programming and Reinforcement Learning

In this section the state space $\mathcal{S}$ is assumed to be large but finite. Hence, the expected immediate reward under policy π can be represented as a vector $R \in \mathbb{R}^{|\mathcal{S}|}$, and the transition probabilities under π can be represented as a matrix $P \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{S}|}$, where both P and R are indexed by the elements of $\mathcal{S}$ with

$$R_s^\pi = \mathbb{E}\left[\mathcal{R}(s_t, a_t)|s_t = s, a_t = \pi(s_t)\right]$$

and

$$P_{ss'}^\pi = \mathbb{P}\left[s_{t+1} = s'|s_t = s, a_t = \pi(s_t)\right].$$

When considering a single fixed policy, we may drop the superscript π. The parameters R and P depend on π, $\mathcal{R}$ and $\mathcal{P}$, and define a *Markov chain with rewards* on the state space with transition probability matrix P.

A value function can also be represented as a vector. For the given policy π, the value function $V^\pi \in \mathbb{R}^{|\mathcal{S}|}$ is the unique solution to the Bellman equations:

$$V^\pi = R + \gamma P V^\pi. \tag{2.1}$$

Successive approximations of V^π can be obtained by repeatedly applying the Bellman backup operator

$$V^{k+1} = T(V^k) \stackrel{\text{def}}{=} R + \gamma P V^k. \tag{2.2}$$

This method is known as dynamic programming, and is known to converge to the true value function.

Often the matrices R and P are not known but sample trajectories of the form $\langle s_0, r_0, s_1, r_1, \ldots, s_T, r_T \rangle$ consisting of the sequence of states and rewards observed when following a policy are available. In this case one can estimate the model parameters (P, R) and apply the method just described.

Alternatively, *temporal difference* (TD) learning can be used to learn the value function directly from trajectories without explicitely representing the model parameters R and P. In TD learning, the value function is updated after each state transition according to

$$\hat{V}(s_t) \leftarrow \hat{V}(s_t) + \alpha \left(r_t + \gamma \hat{V}(s_{t+1}) - \hat{V}(s_t) \right), \tag{2.3}$$

where $0 < \alpha < 1$ is a step-size parameter. This approach assumes that a seperate estimate for each state is maintained in a lookup table, as in dynamic programming. This is a specific instance of the TD(λ) family of algorithms. The reader is referred to Sutton and Barto [24] for details.

2.3 Value Function Approximation

Whether or not the model parameters P and R are available, it is not possible to compute an explicit representation of the value function V^π if the cardinality of the state space is large, or if the state space is continuous. In such cases an approximation $\hat{V}$ of the value function can be computed and subsequently used for policy improvement. When no model is available, the reinforcement learning algorithm TD(λ) can still be used. Instead of applying the TD updates (2.3) to a tabular value function estimate, analogous updates are applied directly to the value function approximator.

Theoretical results [26, 25] show that some convergence guarantees can still be obtained for linear function approximators. This is consequently the most commonly studied type of approximator. The value function approximation has the form $\hat{V} = \Phi\theta$, where Φ is a $|\mathcal{S}| \times m$ matrix in which each row contains the feature vector for a particular state, and θ is a vector of m parameters. Typically it is desirable to have $m \ll |\mathcal{S}|$. Usually, Φ is assumed to be given, and the learning algorithm adjusts θ.

Some common types of linear approximators are state aggregation, tile coding and radial basis functions. *State aggregation* is perhaps the simplest type of approximator. Conceptually, states are grouped together and a single value is stored for each group of states. The feature vector for a given state s is defined as

$$\phi_j(s) = \begin{cases} 1 & \text{if state } s \text{ is in group } j \\ 0 & \text{otherwise} \end{cases}, \qquad \forall j \in \{1, 2, \ldots, m\}.$$

And the feature matrix has the form

$$\Phi = \begin{bmatrix} \phi^\top(s_1) \\ \phi^\top(s_2) \\ \vdots \\ \phi^\top(s_{|\mathcal{S}|}) \end{bmatrix}.$$

In *soft state aggregation* each state may belong to multiple groups with a certain weight. Thus multiple or all entries in a row of Φ may be nonzero, but they must sum to 1. *Radial basis functions* (RBF) networks are another common type of approximator where, when

using Gaussian basis functions and assuming $\mathcal{S} \subset \mathbb{R}^n$,

$$\phi_j(s_i) = \exp\left(-\frac{1}{2}(s - c_j)^\top W_j^{-1}(s - c_j)\right),$$

and $c_j \in \mathbb{R}^n, W_j \in \mathbb{R}^{n \times n}$ are the center and covariance respectively of the j^{th} Gaussian basis function.

Approximate dynamic programming (ADP) and *temporal difference learning* (TD(λ)) are methods for learning the vector of parameters θ given a feature matrix Φ. The main difference between the two is that ADP assumes that the model (the vectors R and P) is known, while TD relies solely on sampled trajectories of the system.

In ADP, one seeks to find θ such that $\hat{V} = \Phi\theta$ is a good approximation to V^π. This can be accomplished by sampling some representative subset of the states, computing the feature matrix for the sampled subset of states Φ_{sample} and solving the for the least-squares fit

$$\min_\theta \left\|\Phi_{\text{sample}}\theta - R - \alpha P \Phi_{\text{sample}}\theta\right\|_2.$$

This step is repeated analogously to the dynamic programming algorithm for smaller discrete MDPs. Convergence of this algorithm is not guaranteed.

In TD learning, an estimate of the value function is updated in response to observed state transitions as

$$\theta \leftarrow \theta z_t \left(r_t + \left(\gamma\phi(s_{t+1}) - \phi(s_t)\right)\theta\right)$$
$$z_0 = \phi(s_0) \qquad , \qquad z_t = \gamma\lambda z_{t-1} + \phi(s_t) \quad \forall t > 0.$$

The vectors z_t are *eligibility traces*. Using positive values for λ accelerates convergence in practice and yields a different solution. This aspect of TD learning is not treated further since the effect of the λ parameter in the batch version of $TD(\lambda)$ actually used in this work is less important and the parameter will be set to $\lambda = 0$.

2.4 Least Squares Temporal-Difference Learning

Least squares TD is a batch method for computing a linear approximation to the value function of an MDP from sampled data. This section describes the LSTDQ extension of LSTD, as well as a related Bellman residual minimizing algorithm.

2.4.1 LSTD and LSTDQ

LSTD(λ) was introduced by Bradtke & Barto [5] for the important case $\lambda = 0$ and extended by Boyan [3] to general $\lambda \in [0, 1]$. LSTD(λ) converges in the limit to the same parameters as the incremental TD(λ) algorithm but does not require a learning rate, and makes more efficient use of data samples at the expense of increased complexity per iteration. In this work we consider only the case where $\lambda = 0$. LSTDQ is an extension of LSTD used in of least-squares policy iteration, a method discussed below. This algorithm is described here since it reduces to LSTD when there is a single action available to the agent.

LSTDQ learns the state-action value function Q^π for a fixed policy π from a set of sampled MDP transitions

$$D = \{(s_t, a_t, r_t, s'_t) | t = 1, 2, \ldots, L\}$$

by computing

$$A = \Phi^\top(\Phi - \gamma\Phi') \text{ and } b = \Phi^\top R$$

where

$$\Phi = \begin{bmatrix} \phi(s_1, a_1)^\top \\ \vdots \\ \phi(s_t, a_t)^\top \\ \vdots \\ \phi(s_L, a_L)^\top \end{bmatrix}, \quad \Phi' = \begin{bmatrix} \phi(s'_1, \pi(s'_1))^\top \\ \vdots \\ \phi(s'_t, \pi(s'_t))^\top \\ \vdots \\ \phi(s'_L, \pi(s'_L))^\top \end{bmatrix}, \quad R = \begin{bmatrix} r_1 \\ \vdots \\ r_t \\ \vdots \\ r_L \end{bmatrix},$$

and solving the system

$$Aw = b.$$

If a model of the MDP is available, the t^{th} row of the matrix Φ' and vector R may be replaced by estimates

$$\Phi'_{t,\cdot} = \sum_{s \in \mathcal{S}} \mathcal{P}(s_t, a_t, s')\phi(s', \pi(s')) \tag{2.4}$$

and

$$R_t = \sum_{s' \in \mathcal{S}} \mathcal{P}(s_t, a_t, s')\mathcal{R}((s_t, a_t, s').$$

Alternatively, for large or continuous state spaces, terms of the summation may be sampled according to $\mathcal{P}(s_t, a_t, \cdot)$. This may allow for greater accuracy when solving problems with stochastic transition probabilities, compared to simply using the single successor state.

2.4.2 Bellman Residual Minimization

Alternatively, the Bellman residual minimizing solution given a sample may be computed as the least-squares solution to

$$\Phi w \approx R + \gamma \Phi' w$$

$$(\Phi - \gamma \Phi')w \approx R$$

with

$$A = (\Phi - \gamma\Phi')^\top(\Phi - \gamma\Phi') \quad \text{and} \quad b = (\Phi - \gamma\Phi')^\top R.$$

This method is discussed by Lagoudakis and Parr [12] in relation to LSTDQ, as well as by Sutton and Barto [24] and Baird [10] in the on-line setting. For non-deterministic MDPs it would be necessary do use "doubled samples" to obtain unbiased estimates. The matrix A has the form

$$A = \sum_t \left(\phi(s_t, a_t) - \gamma\phi(s'_t, \pi(s'_t))\right)\left(\phi(s, a) - \gamma\phi(s'_t, \pi(s'_t))\right)^\top. \tag{2.5}$$

When normalized, it is intended to estimate the expectation over all state-action pairs

$$\overline{A} = \mathbb{E}\left[\left(\phi(s, a) - \gamma\phi(s', \pi(s'))\right)\left(\phi(s, a) - \gamma\phi(s'', \pi(s''))\right)^\top\right]$$

where (s, a) is sampled from a given distribution while s' and s'' are sampled independently according to $\mathcal{P}(s, a, \cdot)$. In order to obtain an unbiased sample for A, it would be necessary to sample two successor states independently using a generative model of the environment.

This is not a problem when the MDP is deterministic or nearly deterministic, such is in many control problems. If the features for the successor states are sampled from the sum (2.4) using a model, each term of (2.5) has the form

$$\begin{aligned} A_t = &\left(\frac{1}{N}\sum_{s'\in\mathcal{T}}\left(\phi(s_t, a_t) - \gamma\phi(s'_t, \pi(s'_t))\right)\right)\left(\frac{1}{N}\sum_{s''\in\mathcal{T}}\left(\phi(s_t, a_t) - \gamma\phi(s''_t, \pi(s''_t))\right)\right)^\top \\ = &\ \frac{1}{N^2}\sum_{s'\in\mathcal{T}}\sum_{s''\in\mathcal{T}}\left(\phi(s_t, a_t) - \gamma\phi(s'_t, \pi(s'_t))\right)\left(\phi(s_t, a_t) - \gamma\phi(s''_t, \pi(s''_t))\right)^\top \end{aligned}$$

where $\mathcal{T}$ consists of N independently sampled successor states. The states s' and s'' are independent in all but N of the N^2 terms. This approach may be preferable to actually drawing independent samples since only a single term needs to be computed for each sample (s_t, a_t) when building the estimate A.

2.4.3 Regularization

The matrix A may not have full rank or be ill-conditioned if the sample is too small, or if the features are not linearly independent. For both LSTDQ and Bellman residual minimization, *ridge regression* may be employed to ensure a unique solution to the system $Aw = b$. The system

$$(A + \Lambda)w = b$$

is solved instead, where Λ is a diagonal matrix, generally a small multiple of the identity.

In the case of Bellman residual minimization, rather than minimising $\|(\Phi - \gamma\Phi')w - R\|_2$, the solution w minimizes the objective

$$\|(\Phi - \gamma\Phi')w - R\|_2 + \|\Lambda w\|_2.$$

Such regularization can also act as a safeguard against overfitting since it penalizes large weights. One way to select the diagonal elements of Λ is via a validation set, where Λ is selected to minimize the Bellman residual of a second validation sample different from that used to compute the actual estimates.

2.4.4 Choice of Algorithm

Lagoudakis and Parr [12] provide a comparison between the two methods. Schoknecht [20] showed that the solution found by both algorithm minimizes a different quadratic objective. In essence, LSTD finds a fixed point of the projected Bellman

operator in the range of the feature matrix Φ, and thus the Bellman residual is orthogonal to the range space of Φ. In contrast, the Bellman error minimizing solution is in general not a fixed point of the projected Bellman operator.

LSTD has been observed to provide better policies and is generally preferred in practice. However, the Bellman error minimizing approach provides a different insight on the representation power of a given set of features.

2.5 Policy Iteration and Least Squares Policy Iteration

In standard policy iteration, a value function approximation is computed via LSTD after each policy update. Thus on-policy samples are required at each policy iteration. Furthermore, a model of the environment is required in order to complete the policy update since one must solve the optimization problem

$$\pi(s_t) \in \arg\max_{a \in \mathcal{A}} \mathbb{E}\left[\mathcal{R}(s,a) + \gamma \hat{V}(s_{t+1}) | a_t = a, s_t = s\right].$$

In contrast, *least-squares policy iteration* (LSPI) uses LSTDQ to compute an approximate action-value function $\hat{Q}$ given a sample of MDP transitions D and an initial policy π. This facilitates the policy improvement step if no model or only an approximate model are available, as the above optimization problem is replaced by

$$\pi(s_t) \in \arg\max_{a \in \mathcal{A}} \hat{Q}(s,a).$$

Since the transitions need not be sampled according to the current policy, it is also possible to reuse samples if the available data is limited or costly to obtain.

CHAPTER 3
Automatic Basis Function Selection in Reinforcement Learning

This chapter reviews methods for automatically selecting basis functions for use in linear approximators. The algorithm presented in Chapter 5 is largely inspired by the *adaptive state aggregation* algorithm for solving discrete MDPs with explicit models, which is outlined in the first section. A number of methods for adapting or creating basis functions during learning are briefly discussed. Two current broad approaches are contrasted: on the one hand, *proto-value functions* are a framework for constructing global basis functions based on an analysis of state transitions, and on the other, Bellman error based methods seek to create new basis functions using information available during learning, namely the Bellman residual.

3.1 Adaptive State Aggregation

Bertsekas and Castañon [2] propose a state aggregation method for the purpose of speeding up policy evaluation in the case where the state space is sufficiently small that the value function can be explicitely represented as a vector $V \in \mathbb{R}^{|\mathcal{S}|}$, and where the R and P matrices are available explicitely.

Dynamic programming is used to compute V^π, but applications of the Bellman backup operator,

$$V^{k+1} = T(V^k) = R + \gamma P V^k,$$

are interleaved with aggregation iterations of the form

$$\overline{V}^k = V^k + \Psi y,$$
$$V^{k+1} = T(\overline{V}^k) = R + \gamma P \overline{V}^k$$

where Ψ is an $|\mathcal{S}| \times m$ matrix for a small value of m (e.g. $m = 2$) and $y \in \mathbb{R}^m$. Here, Ψy is a low rank correction to V^k.

Though the value function is being represented explicitely, the correction takes the form of a linear function approximator. This will be exploited in chapter 5 to generalize the approach to the case where we are defining new basis functions for an existing linear approximator $\hat{V}$.

Under the assumptions that $\Psi^\dagger(I - \gamma P)\Psi$ is nonsingular and that $\Psi^\dagger\Psi = I$, y is chosen to solve the system

$$y = R_A + \gamma P_A y \quad \Leftrightarrow \quad y = (I - \gamma P_A)^{-1} R_A, \tag{3.1}$$

where

$$P_A = \Psi^\dagger P \Psi,$$
$$R_A = \Psi^\dagger (T(V^k) - V^k).$$

$\Psi^\dagger = (\Psi^\top\Psi)^{-1}\Psi^\top$ denotes the pseudoinverse, and we thus require that the columns of Ψ be linearly independent so that the pseudoinverse can be computed and so that $\Psi^\dagger\Psi = I$. The matrix Ψ is selected to be a state aggregation matrix, though this is not required by the error analysis.

The system (3.1) is a small policy evalution defined on the aggregate Markov chain with trasition probabilities P_A and rewards R_A. Each state corresponds to a group of

states in the original MDP, P_A contains the group-to-group transition probabilities, and the rewards R_A represent the average Bellman residuals for the states in each group.

The choice of y is intended to provide and additive correction to V^k which will bring it close to the solution V^π of (3.1). The Bellman residual can be expressed as,

$$\begin{aligned} T(V^k) - V^k &= R + \gamma P V^k - V^k \\ &= V^\pi - \gamma P V^\pi + \gamma P V^k - V^k \\ &= (I - \gamma P)(V^\pi - V^k), \end{aligned}$$

where we would like to have $\Psi y \approx V^\pi - V^k$. Substituting in Ψy and multiplying the equation be $\Psi^\dagger$ yields the system (3.1) :

$$\begin{aligned} \Psi^\dagger(T(V^k) - V^k) &= \Psi^\dagger(I - \gamma P)\Psi y, \\ R_A &= (I - \gamma P_A)y. \end{aligned}$$

Bertsekas and Castañon show that the Bellman error after an aggregation iteration is described by a sum of two terms:

$$\begin{aligned} T(V^{k+1}) - V^{k+1} &= E_1 + E_2 \\ E_1 &= (I - \Pi)(T(V^k) - V^k) \\ E_2 &= \gamma(I - \Pi)P\Psi y \end{aligned} \tag{3.2}$$

where the matrix $\Pi = \Psi\Psi^\dagger$ is an orthogonal projection onto the range space of Ψ (it is symmetric and $\Pi^2 = \Pi$.) In fact, for any $x \in \mathbb{R}^{|\mathcal{S}|}$, $(\Pi x)_s$ is the mean value of the elements of x in state s's aggregate group. Then the first error term E_1 is the projection of the Bellman errors onto the nullspace of Ψ, and its s-th element is the Bellman error at state s minus the mean Bellman error in state s's aggregate group. A careful choice of Ψ

can bound the contribution of E_1 in terms of the Bellman error $(T(V^k) - V^k)$ at iteration k.

Indeed, Bertsekas and Castañon propose to divide the range of Bellman errors at iteration k into m equally sized intervals of length $F(T(V^k) - V^k)/m$, with $F(x) = (\max_s x_s - \min_s x_s)$, and group the states with errors in each interval. This is shown to provide a reduction in the first term by a factor of at least $2/m$ in terms of the F pseudonorm defined above. The second term E_2 depends on how well the aggregation Ψ preserves the action of the transition matrix P, and is shown to be small for certain interesting classes of transition matrices. The final algorithm ensures convergence in general despite any error introduced by E_2 by interleaving applications of the Bellman operator with the aggregation steps.

3.2 Basis Function Adaptation and Selection

A number of function approximators for reinforcement learning have been proposed in recent years. This section briefly describes some methods for automatically constructing or updating basis functions for reinforcement learning.

Singh *et al.* propose an algorithm to update a soft state aggregation [21]. Unfortunately the number of parameters required remains larger than the number of states, since the feature vector for each state must be stored. The method is shown to converge almost surely for Q-learning, and consequently for TD(0), when a fixed aggregation is used. A heuristic for adaptive state aggregation is proposed, which uses stochastic gradient descent on the squared Bellman error to update the feature matrix Φ.

Menache *et al.* [14] propose algorithms to update the parameters of radial basis functions during TD learning, using gradient descent or the cross-entropy method.

Unlike state aggregation, radial basis functions have a compact representation. The algorithms interleave applications of LSTD with optimization of the basis functions, using estimates of the weighted squared Bellman error as the objective. It is observed that the gradient-descent algorithm converges to local minima, while the cross-entropy method avoids this problem.

Ratitch & Precup [17] propose an approximator similar to *normalized* radial basis function networks, where basis functions are selected online during learning to be centered near visited states. The resulting approximator has the property of being local, which is not necessarily the case for unnormalized RBF networks. Thus it aims to allocate limited resources to approximate the value function only in relevant regions of the state space. The online nature of the algorithm allows the approximator to be constructed and updated in a simple, efficient and intuitive as learning takes place.

In a similar spirit, explicit manifold representations apply knowledge of the state space topology to construct local approximators. The approach avoids the problem that states whose representations are close in Euclidean space, may in fact not be quickly reachable from each other and thus have wildly different values. In the work of Smart [22], short trajectories are used to determine reachable states from a starting point, and the visited states are grouped into a "chart" with its own function approximator. This is repeated for a number of states, and the resulting overlapping local approximators are blended to construct a global value function approximator. This approximator successfully represents discontinuities in the value function, but requires the additional step of constructing the local charts before learning the value function. Moreover, the topology of the state space may depend strongly on the policy. Thus the topology must

be updated on policy improvement steps in policy iteration, or for Q-learning, a seperate approximator is needed for each action.

3.3 Proto-value Functions

Mahadevan and Maggioni propose proto-value functions [13]. This approach applies spectral analysis of the state space topology to build global basis functions. In the discrete case, trajectories following some policy are used to construct a weighted graph in which successively visited states are connected. A subset of the eigenvectors of a diffusion operator on the graph are used as basis functions. The framework is quite different from the methods considered in this thesis, and the reader is directed to [13] for a comprehensive exposition, and for extensions to continuous-state MDPs.

A major advantages of the approach is that the construction of the basis functions depends only on the trasition function of the MDP. Once basis functions are constructed, they can be reused to solve MDPs with different reward structures. This may be helpful if one wishes to solve a number of problems on the same state space, and is also an advantage of manifold learning. However, proto-value functions reflect *global* properties of the state space such as symmetries or bottleneck states, which avoids the duplication of effort required with local approximators. Finally, errors in the reward estimates have no negative impact.

On the other hand, significant changes to the policy may change the topology of the state space enough to require the entire analysis to be repeated, possibly negating the utility of re-using basis functions. Furthermore, learning an accurate transition model is non-trivial and may require a large number of samples. This is the major disadvantage of the approach, since most of the methods mentioned in the previous section do not require

any expensive pre-processing steps. Of course, when a complete model is available, this limitation is less serious.

3.4 Bellman Error Basis Functions

An alternative approach, which includes the method proposed in this thesis, is to generate new basis functions based on the Bellman error of an existing value function approximation. Earlier work relating to this thesis was presented in [11], and that approach was applied in conjunction with LSPI to a three-dimensional maze navigation problem by Sprague [23]. The relationship of these methods to the other approaches mentioned is discussed further in Section 7.2. The *norm* of the Bellman error is also used to determine basis functions in earlier works mentioned above [21, 14], thought they do not directly use Bellman error information to construct basis functions.

From a more theoretical point of view, Parr *et al.* [16] provide a theoretical analysis of *Bellman error basis functions* (BEBFs). A BEBF is simply the Bellman error $(T\hat{V} - \hat{V})$ where $\hat{V}$ is the current value function approximation. It may be stored exactly if the state space is small enough, or it may be approximated. In the case where an MDP model is available and the exact Bellman error for each state is stored, BEBFs are shown to form an orthonormal basis. Moreover, successively generating new exact BEBFs is shown to provide an improved approximation at each step. In the case where only sample trajectories are available, and the Bellman error is approximated, the authors provide a condition guaranteeing an improvement in the approximator.

While both Proto-value functions and BEBFs provide an orthogonal basis for the value function, it is important to note that the bases are of a different nature, and that Proto-value functions generate an orthogonal basis regardless of whether an exact model

is used, in contrast to BEBFs. Further differences between the approaches are discussed in Section 7.2.

CHAPTER 4
Dimensionality Reduction

A key idea of the proposed basis function construction method is to find a low dimensional projection of the state space in which to work. This chapter presents a dimensionality reduction algorithm originally developped for optimizing nearest neighbour classification, and its adaptation for the purpose of computing projections of the state space.

The state of the MDP is assumed to be represented as a vector of real numbers. The goal of the dimensionality reduction algorithm is to find a linear projection $A \in \mathbb{R}^{d \times n}$ from the state space to a lower-dimensional space $\mathbb{R}^d$ which preserves only the relevant features of the MDP value function. We are presented with a set of pairs $\langle x_i, y_i \rangle$, where each $x_i \in \mathbb{R}^{\aleph}$ is the representation of an MDP state, and y_i is a measure of the value funtion approximation error at that state. In the supervised learning context of NCA, x_i is the input and y_i is the target output.

4.1 Neighbourhood Component Analysis

Neighbourhood Component Analysis (NCA) is a method for computing a distance metric to optimize nearest-neighbour classification performance [7]. NCA is designed to learn weighted distance metrics between points $x, y \in \mathbb{R}^n$,

$$\mathcal{D}(x, y) = (x - y)^\top Q (x - y),$$

where Q is a matrix weighting each dimension. NCA searches for a Q of the form $A^\top A$ where A may be restricted to be in $\mathbb{R}^{d\times n}$ for a fixed $d \ll n$. In this case, the distance metric can be re-written as:

$$\mathcal{D}(x,y) = (x-y)^\top A^\top A(x-y) = (Ax-Ay)^\top (Ax-Ay).$$

Hence, A is in fact a linear transformation from $\mathbb{R}^n$ to $\mathbb{R}^d$, and classification can take place in this lower dimensional space using the Euclidean norm. This reduces computational and storage requirements and provides an effective dimensionality reduction algorithm.

In the classification context, the metric is learned maximizing the objective

$$f_{class}(A) = \sum_i \sum_{j\in C_i} p_{ij} = \sum_i \sum_{j\in C_i} \frac{\exp\left(-\|Ax_i - Ax_j\|^2\right)}{\sum_{k\neq i}\exp\left(-\|Ax_i - Ax_k\|^2\right)}$$

where p_{ij} defines the probability of selecting point j as the neighbour of point i when using a *stochastic* nearest neighbour selection rule for classification, i.e. one where the class returned for a query point i is the class of the neighbour randomly selected according to the neighbourhood probabilities p_{ij}. The objective $f_{class}(A)$ is then effectively the *expected number of points correctly classified* under leave-one-out cross-validation.

A local maximum of f_{class} is found for $A \in \mathbb{R}^{d\times n}$ using a gradient descent algorithm such as delta-bar-delta or conjugate-gradient. The optimization takes place over $\mathbb{R}^{d\times n}$ for a predetermined d. Though only a local minimum is found by this algorithm, Goldberger *et al.* report success with no or few restarts on a number of

standard machine learning benchmarks, as well as on an image classification task [7]. The main compurational cost of the algorithm is the computation of the gradient of f_{class}.

A shortcoming of the approach is the need to explicitely choose the target dimension d *a priori*. This parameter may be chosen via cross-validation if the metric is being learned off-line for performing classification, but this is not desireable if the NCA algorithm is being used as a subroutine without direct intervention.

The original authors suggest extending their algorithm to the case where continuous target values are given instead of class labels by assigning points with nearby labels to the same class. The following section uses a different approach however, preferring to directly incorporate the continus targets into the objective.

4.2 NCA for Regression

The method is extended nearest-neighbour regression by modifying the objective appropriately.

For brevity, $x_{ij} = (x_i - x_j)$ and $y_{ij} = (y_i - y_j)$ denote the differences between the points i and j, and between their target values, respectively. Let the objective be the sum of the expected square regression errors under leave-one-out cross-validation:

$$f(Q) = \sum_i \sum_{j \neq i} (y_i - y_j)^2 p_{ij} = \sum_i \sum_{j \neq i} \frac{y_{ij}^2 \exp(-x_{ij}^\top Q x_{ij})}{\sum_{k \neq i} \exp(-x_{ik}^\top Q x_{ik})} = \sum_i \frac{t_i}{s_i}, \tag{4.1}$$

where

$$s_i = \sum_{j \neq i} \exp(-x_{ij}^\top Q x_{ij}) \qquad \text{and} \qquad t_i = \sum_{j \neq i} y_{ij}^2 \exp(-x_{ij}^\top Q x_{ij})$$

Denoting $\phi_{ij} = \exp(-x_{ij}^\top Q x_{ij})$, the gradient of f with respected to the factor A in $Q = A^\top A$ is

$$\begin{aligned}
\frac{\partial f}{\partial A} &= \sum_i s_i^{-2} \left(\frac{\partial t_i}{\partial A} s_i - \frac{\partial s_i}{\partial A} t_i \right) \\
&= \sum_i s_i^{-2} \left(\left(-2 s_i \sum_{j \neq i} y_{ij}^2 \phi_{ij} A x_{ij} x_{ij}^\top \right) - \left(-2 t_i \sum_{j \neq i} \phi_{ij} A x_{ij} x_{ij}^\top \right) \right) \\
&= 2 \sum_i \sum_{j \neq i} \left(\frac{t_i}{s_i^2} - \frac{y_{ij}^2}{s_i} \right) \phi_{ij} A x_{ij} x_{ij}^\top \\
&= 2 \sum_i \sum_{j \neq i} \alpha_{ij} A x_{ij} \left(x_i^\top - x_j^\top \right) \\
&= 2 \sum_i \left(\sum_j \alpha_{ij} A x_{ij} \right) x_i^\top - 2 \sum_j \left(\sum_i \alpha_{ij} A x_{ij} \right) x_j^\top \\
&= 2 \left(M_1 X^\top - M_2 X^\top \right)
\end{aligned}$$

where

$$\alpha_{ij} = \left(\frac{t_i}{s_i^2} - \frac{y_{ij}^2}{s_i} \right) \exp(-x_{ij}^\top Q x_{ij})$$

is a scalar multiplier for each of the $N^2 - N$ terms, X is the n-by-N matrix containing the points x_i in the columns, M_1 is the d-by-N matrix with i^{th} row $\sum_j \alpha_{ij} A x_{ij}$, and M_2 has j^{th} row $\sum_i \alpha_{ij} A x_{ij}$. The constraint $j \neq i$ for the summation can be dropped since when $i = j$, $x_{ij} = 0$.

The gradient can be evaluated efficiently by precomputing the values of s_i and t_i for each point. The result of a single evaluation of the factor $\phi_{ij} = \exp(-x_{ij}^\top Q x_{ij})$ for each pair of points is stored and reused. The evaluation of s_i, t_i and α_{ij} then only require scalar operations. The distances between all pairs of points in low dimensions,

$Ax_{ij} = Ax_i - Ax_j$, are precomputed and reused to form the matrices M_1 and M_2 and the factors ϕ_{ij}.

Minimizing the objective f would ideally also optimize the norm of the matrix Q, however it was observed that in many cases $\|Q\|$ tended to become large. This is problematic, since for sufficiently large Q, the value of $\exp(-x_{ij}^\top Q x_{ij})$ becomes too small and causes floating-point underflows.

In terms of the original nearest-neighbour classification formulation, this effect was seen as desirable. The norm of Q determines the variance of the the probability distribution used to select neighbours during classification, and leaving it free during learning is intended to optimize this parameter. In this spirit it is considered undesireable to fix the norm of the metric. In a previous formulation, presented in [11], a regularization term proportional to the Frobenius norm of Q was added. However this required the choice of an extra parameter determining the strength of the preference for small Q which needed to be chosen based on the data being used.

Instead a hard nonlinear constraint bouding the norm of Q may be added to the optimization problem . This is integrated into the objective using the log-barrier method [4], yielding the objective

$$g(A) = f(A) - \frac{1}{t} \log \left(1 - \frac{\|A\|_F^2}{q_{max}} \right),$$

with gradient

$$\frac{\partial g}{\partial A} = \frac{\partial f}{\partial A} + \frac{2}{t} \left(1 - \frac{\|A\|_F^2}{q_{max}} \right)^{-1} A.$$

When the x_i and y_i are normalized to the $[0, 1]$ range, a suitable bound on the norm of A is $q_{max} = 100$. This bounds the value of $\exp(-x_{ij}^\top Q x_{ij})$ from below by $\exp(-100)$,

which is amply large to avoid underflow when using double-precision floating point numbers. The value of t is gradually increased during optimization.

A simpler and more efficient approach is to simply prevent ϕ_{ij} from going to zero by setting it to $\phi_{ij} = \exp(-x_{ij}^\top Q x_{ij}) + \epsilon$ where ϵ is near the machine epsilon. This has the effect of assigning all neighbours with equal probability to a point if the norm of Q becomes too large. This last approach is preferred for its simplicity.

4.3 NCA Optimization

Two minimization algorithms were tried to optimize the objective over the set $\mathbb{R}^{d \times n}$. The first used the Fletcher-Reeves conjugate-gradient method [15, 4]. For each search direction, the line search algorithm provided by Nocedal and Wright [15] is used to find a local minimum. The search is halted when no progress is made during a line search in the direction of the gradient, i.e. after having reset the search direction.

The second algorithm is MATLAB's *fminunc* function, which uses the BFGS Quasi-Newton method with a mixed quadratic and cubic line search procedure.

The latter approach requires fewer objective evaluations at the expense of more gradient evaluations. As implemented, the difference in evaluation time for the objective and its gradient is generally within a factor of 2-3, so this method is preferred.

It was noted that when the optimization reached a local minimum, further progress could be achieved by restarting the search at the Cholesky factor of $Q = A^\top A$. Besides this observation, the choice of starting point seems to have little impact: the attained objective value is similar even though the actual solutions may vary greatly. Thus the initial transformation is chosen randomly.

4.4 Dimensionality Reduction

When the search terminates, only the principal eigenvectors of $Q = A^\top A$ are retained to provide for dimensionality reduction. Two criteria are used to determine the number of eigenvectors to keep: first the leading eigenvalues whose sum is greater than 95% of the sum of all the eigenvalues are kept, then any eigenvalues representing less that 5% of the total sum are discarded. This is a slightly more agressive version of the *percent-of-variance* criterion commondly used in principal component analysis. There is also a hard limit on the number of dimensions permited by the choice of d used in the optimization, thus this value should be chosen to be sufficiently large. This procedure yields the transformation

$$A^* = \begin{bmatrix} \sqrt{\lambda_1} v_1^\top \\ \sqrt{\lambda_2} v_2^\top \\ \vdots \\ \sqrt{\lambda_k} v_k^\top \end{bmatrix}$$

where $\lambda_1, \ldots, \lambda_k$ are the $k \leq d$ largest eigenvalues of Q and $v_1, \ldots, v_k$ are the corresponding eigenvectors.

4.5 Performance and Potential Improvements

The NCA algorithm finds satisfactory projections in all the data sets considered. Many modifications are possible however. First, note that the computation of the gradient is independent of the regression loss function used. The squared error was used because it is common and provides satisfactory results, however an arbitrary cost measure could be used. Indeed, if the appropriate binary loss function is used, the algorithm reduces to the orginial NCA method for classification.

The optimization algorithm may yield different solutions depending on the starting point, but the value of the objective and the number of dimensions selected is usually the same. In practice, the search may be repeated and the solution yielding the minimum output dimension with the highest "percent-of-variance" measure is retained. The final dimension k has not been observed to change by more than 1 for the same data set.

The time needed to compute the objective grows quadratically in the number of points, and linearly in the input dimension. The optimization generally takes less than 5 seconds for an eight-dimensional problem with a sample of 200 points. Gradient evaluations take less than one second for input dimensions up to ≈ 4000 with sample sizes of 400 points, though memory limitations become a concern at this point and using sparse matrices may degrade performance.

It is not necessary to use the same set of points as "centers" and "test points". That is, the sets of points indexed by i and j in the objective may be different. In particular, it may be preferable to have a reduced set of centers to which a point may be assigned to accelerate gradient evaluations.

Other modifications to the objective were considered. Using an un-normalized version of the objective, i.e. simply taking $f(Q) = \sum_i t_i$, yielded similar solutions in most cases, but seemed less robust. Finally, a formulation more similar to the original NCA formulation for classification was presented in a prior paper [11]. This approach was less efficient overall, required an *a priory* choice of the the low dimension d, and required the selection of a regularization constant.

4.6 Downsampling

For large data sets it is possible to sample terms from both the inner and outer sums of the objective (4.1). To handle the case where most of the points have similar labels and a small portion of points have significantly different labels, such downsampling is not done uniformly. Instead, the range of the labels is divided into N bins, and the points in each bin are assigned equal probability of being retained such that their total probability is $1/N$. Distinct sets of points are used for the inner and outer sum.

CHAPTER 5
Automatic Basis Function Construction

This chapter presents the proposed method for automatically constructing basis function. The algorithm can be used in various reinforcement learning settings whether or not a model or a generative model of the problem are available.

5.1 Oveview of the Algorithm

The algorithm extends the approach of Bertsekas and Castañon described in Section 3.1 to the case where the value function is being represented by a linear function approximator $\hat{V} = \Phi\theta$. Given a current approximation, a new set of basis functions is created to fit an *additive correction* of the form Ψw to the approximator. Since the new features share the linear form of the original approximator, the new basis functions can be added to the old ones to augment the feature matrix Φ:

$$\begin{aligned}\hat{V}^{k+1} &= \hat{V}^k + \Psi w = \Phi^k\theta^k + \Psi w \\ &= \begin{bmatrix}\Phi^k & \Psi\end{bmatrix}\begin{bmatrix}\theta^k \\ w\end{bmatrix} = \Phi^{k+1}\begin{bmatrix}\theta^k \\ w\end{bmatrix}.\end{aligned}$$

The weight vector could be set to $\theta^{k+1} = [\theta^k\ w]^\top$, but all the elements of θ^{k+1} are recomputed since this may provide an improved solution. Since the weights θ^{k+1} are recomputed, the choice of w appears to have little importance, as only the range of the new basis functions Ψ affects the final result. However, the correction Ψw plays an

important role conceptually. It links this work to that of Bertsekas and Castañon and provides the motivation for the approach.

The choice of the correction Ψw is made based on the Bellman error of the current value function approximation. In order to tackle problems with high-dimensional state spaces, dimensionality reduction is used to map the state space to a lower-dimensional space while preserving the structure of the Bellman error. The NCA formulation of Section 4.2 is used for this purpose, but other algorithms may be appropriate. In the resulting low-dimensional projection of the state space, it is tractable to efficiently place new basis functions which aim to approximate the Bellman error, rather than the unknown value function directly. The method considered here for selecting basis functions is a single-layer radial basis function network.

In effect, two seperate mappings are created and $\psi : \mathbb{R}^n \rightarrow \mathbb{R}^m$ is their composition. First, the linear projection $A : \mathbb{R}^n \rightarrow \mathbb{R}^d$ computed by NCA transforms states to low-dimensional vectors in $\mathbb{R}^d$. Then a set of features defined in $\mathbb{R}^d$ maps these vectors to $\mathbb{R}^m$ where m is the number of basis functions added. Different algorithms that those selected here may be used to create either mapping.

A pruning algorithm is used to eliminate any redundant features which may be added, to remove features which have become obsolete as the algorithm progresses, and to limit the total number of features as more are generated. The overall procedure for learning a value function from a given trajectory is shown in Algorithm 1.

BFS $(d_{max}, m_{max}, K_1, K_2, tol, \langle s_t, r_t, s'_t \rangle_{t=1...T}, \gamma)$

d_{max} : maximum dimension of each projection

m_{max} : maximum number of basis functions to add per iteration

K_1 : maximum number of iterations

K_2 : maximum number of NCA calls per iteration

tol : tolerance on value function accuracy

$\langle s_t, r_t, s'_t \rangle_{t=1...T}$: sampled MDP transitions

γ : MDP discount factor

$k \leftarrow 0$

$\phi^0(\cdot) \leftarrow 1(\cdot)$

$\hat{V}^0 \leftarrow$ FITVF$(\Phi^0, [r_t], \gamma)$

repeat

$k \leftarrow k + 1$

$[e_t^{k-1}] \leftarrow$ ESTIMATEERRORS$([s_t], [r_t], [s'_t], \hat{V}^{k-1})$

$\psi(\cdot) \leftarrow$ CREATEFEATURES $(d_{max}, m_{max}, K_2, [s_t], [e_t], [s'_t], \gamma)$

$\phi^k(\cdot) \leftarrow$ PRUNE$([\phi^{k-1}(\cdot)\ \psi(\cdot)], [s_t], [r_t], tol/2)$

$\hat{V}^k \leftarrow$ FITVF$(\phi^k([s_t]), [r_t], \phi^k([s'_t]), \gamma)$

until $\|\hat{V}^k - \hat{V}^{k-1}\|_2 \leq tol$ or $k = K_1$

return $\phi^k(\cdot)$

Algorithm 1: Basis function selection for approximating $V(s)$. This is the main procedure which iteratively updates a set of features. For approximating an action-value function, state-action pair samples $\langle (s_t, a_t), r_t, (s_{t+1}, a_{t+1}) \rangle$ may be substituted for the state-only samples. The subroutines ESTIMATEERRORS, CREATEFEATURES, FITVF and PRUNE may have different implementations depending on the context.

5.2 Value Function Correction

Given the current value function approximation $\hat{V}^k$, the aim is to find a correction W such that

$$\hat{V}^k + W = R + \gamma P\left(\hat{V}^k + W\right).$$

Rearranging the terms yields

$$\begin{aligned} R + \gamma P\hat{V}^k - \hat{V}^k &= W - \gamma PW \\ T\hat{V}^k - \hat{V}^k &= W - \gamma PW \\ W &= (T\hat{V}^k - \hat{V}^k) + \gamma PW. \end{aligned}$$

This defines a new policy evaluation problem with the same transition probabilities on the same state space, but with the current Bellman errors as the costs in each state. Call this new problem the *correction MDP*. If this problem could be solved exactly, the corrected value function would be the solution to the original problem. However only an approximate solution can be obtained. We seek to find basis functions and weights $\Psi w \approx W$. Denote the orthogonal projection onto the range of Ψ by $\Pi = \Psi(\Psi^\top\Psi)^{-1}\Psi^\top$, and define $E = (T\hat{V}^k - \hat{V}^k)$ as shorthand for the current Bellman error.

If the correction is simply taken to be the projection of E onto the range of Ψ, the Bellman error for the corrected value function is given by

$$\begin{aligned} T\overline{V}^k - \overline{V}^k &= T(\hat{V}^k + \Pi E) - (\hat{V}^k + \Pi E) \\ &= g + \gamma P\hat{V}^k - \hat{V}^k + \gamma P\Pi E - \Pi E \\ &= (I - \Pi)E + \gamma P\Pi E. \end{aligned} \tag{5.1}$$

The first term is the portion of the Bellman error which cannot be represented as a linear combination of the new basis functions. The second term results from the action of the transition matrix P on the correction, which may produce a new error orthogonal to the new and the existing basis functions.

Without taking into account any knowlegde of the transition matrix P one can attempt to minimize the resulting Bellman error by selecting Ψ to efficiently represent E and ignore the second term. This is in fact the approach taken by Bertsekas and Castañon to accelerate the convergence of dynamic programming: a state aggregator is automatically constructed to aggregate states with similar Bellman errors. See Section 3.1 for details. The approach is also valid for function approximation. However, the next section describes how information from sampled trajectories can be used to reduce the second term above.

5.3 Solving the Correction MPD

Consider instead the error resulting from adding an approximate solution to the correction MDP. Let $\hat{W} = \Pi W$ be the least-squares projection of the fixed point W onto the range of the the features Ψ. The Bellman error of the corrected value function is

$$\begin{aligned} T\overline{V}^k - \overline{V}^k &= T(\hat{V}^k + \hat{W}) - (\hat{V}^k + \hat{W}) \\ &= g + \gamma P\hat{V}^k - \hat{V}^k + \gamma P\Pi W - \Pi W \\ &= E + \gamma P\Pi W - \Pi W \\ &= T_E(\Pi W) - \Pi W \end{aligned}$$

where T_E denotes the Bellman operator on the correction MDP. This quantity is simply the Bellman error for the correction MDP. Using the fact that $W = E + \gamma PW$ It can be

written as

$$\begin{aligned} T\overline{V}^k - \overline{V}^k &= E + (\gamma P - I)\Pi W \\ &= E + (\gamma P - I)\Pi(E + \gamma P W) \\ &= (I - \Pi)E + \gamma P \Pi(E + \gamma P W) - \Pi \gamma P W \\ &= (I - \Pi)E + \gamma(P\Pi - \Pi P)W, \end{aligned}$$

where the last expression is comparable to equation (5.1). The magnitude of W may be much larger that E in general. We have only the bound

$$\|W\|_\infty \leq \sum_{t=0}^{\infty} \gamma^t \|E\|_\infty = \frac{\|E\|_\infty}{1-\gamma}.$$

Thus there is only an improvement if $(P\Pi W \approx \Pi P W)$.

In terms of the solution W, the updated error is

$$\begin{aligned} T\overline{V}^k - \overline{V}^k &= E + \gamma P \Pi W - (I - (I - \Pi))W \\ &= E + \gamma P \Pi W - W + (I - \Pi)W \\ &= E + \gamma P \Pi W - (E + \gamma P W) + (I - \Pi)W \\ &= (I - \Pi)W - \gamma P (I - \Pi)W. \\ &= (I - \gamma P)(I - \Pi)W. \end{aligned}$$

This quantity may be minimized by selecting Π to minimize the second factor. That is, we seek simply to choose features to minimize the Bellman error of the correction MDP. Algorithm 2 is used to choose features. An approximation $\hat{W}$ of the value function at each state in the sample is maintained. On each iteration, NCA is used to learn a

mapping to a low-dimensional space and a number of basis functions are placed in this new space. An approximate solution to the correction MDP is computed using these basis functions, and this new approximation of W is used on the subsequent iteration.

The method is only a heuristic, and there is no guarantee that the solution improves over the iterations. However, on the first iteration, the algorithm is simply choosing features to fit the Bellman error. Thus, even if there is no improvement in subsequent iterations, the same basis functions will be returned as when using Bellman error basis functions as in [11] and [16]. It should be noted that the norm of the Bellman error may be a bad indicator of the quality of basis functions: in the experiments presented here, as in the previous results of [11], an initial *increase* of the Bellman error norm is observed when adding new features.

The specific feature selection method is shown in Algorithm 3. A number of points are chosen at random from the sample to be RBF centers, with probability proportional to the Bellman error magnitude, and the widths are chosen to minimize the Bellman error on a validation sample. A variable resolution grid search is done on the width parameter σ for all RBFs. Then redundant features are pruned using the algorithm descrived in section 5.7. This methode requires a large number of calls to the FITVF procedure to approximately solve the policy evaluation problem. However, the expense is minimal as long as the number of features m_{max} and the sample size T are small. The first parameter is arbitrary and affects the accuracy of the approximator, while the second may be adjusted by downsampling, as for NCA. A moderately efficient MATLAB implementation of LSTD can quickly compute the solution for samples of size ~ 10000, which should be sufficient given the low-dimensional state space.

CREATEFEATURES $(d_{max}, m_{max}, K_2, [s_t], [e_t], [s'_t], \gamma)$
$\hat{W} \leftarrow [e_t]$
for $\ell = 1, 2, \ldots, K_2$ **do**
  $A \leftarrow \text{NCA}([s_t], \hat{W}, d_{max})$
  $\psi^\ell(\cdot) \leftarrow \text{SELECTFEATURES}([s_t], \hat{W}, A, m_{max})$
  $\hat{W} \leftarrow \text{FITVF}(\psi^\ell([s_t]), [e_t^{k-1}], \gamma)$
  $err_\ell \leftarrow \left\| \text{ESTIMATEERRORS}([s_t], [e_t^{k-1}], [s'_t], \hat{W}) \right\|_2$
end for
return $\psi^L(\cdot)$ such that $L = \arg\min_\ell err_\ell$

Algorithm 2: Feature creation through the solution of the corrective MDP. The Bellman errors serve as the initial value function approximation $\hat{W}$. On each iteration, an NCA transformation is computed and features are selected in the low-dimensional space. The value function approximation obtained after fitting is used as the new $\hat{W}$ on the subsequent iteration. The set of features yielding the minimum Bellman error for the correction MDP is retained. Setting K_2 to 1 results in basing the choice of eatures only on the Bellman error.

SELECTFEATURES $(A, m_{max}, [s_t], [e_t], [s'_t], \gamma, tol)$
$p_t \leftarrow |e_t| / \sum_i |e_i| \qquad \forall t$
Choose m_{max} points at random according to probabilities p_t,
  denote them $c_1, c_2, \ldots, c_{m_{max}}$
Define radial basis functions

$$\psi_j(s) = \exp\left(-\frac{1}{2\sigma}(As - c_j)^\top (As - c_j)\right)$$

Select the parameter σ via a grid search
$\psi(\cdot) \leftarrow \text{PRUNE}(\psi(\cdot), [s_t], [e_t], tol)$
return $\psi(\cdot)$

Algorithm 3: Low-dimensional feature selection heuristic.

5.4 Estimating Bellman errors

If a complete MDP model is available, the Bellman errors may be computed exactly. If only a generative model is available, they can be estimated by sampling. If no model is available, they must be estimated from available observations. We seek an approximation

$$e_t^{k-1} \approx \mathcal{R}(s_t) + \gamma \sum_{s \in \mathcal{S}} \mathcal{P}_{s_t,s} \hat{V}^{k-1}(s) - \hat{V}^{k-1}(s_t).$$

If only sampled transitions are available, the error is set to the TD error

$$e_t^{k-1} = r_t + \gamma \hat{V}^{k-1}(s_{t+1}) - \hat{V}^{k-1}(s_t).$$

This is equivalent to simply taking a single sample for each state.

5.5 Fitting the Value Function

The subroutine FITVF computes weights for the features in order to approximate the value function. Different algorithms are used, reflecting the discussion of Section 2.4. The availability of a generative model and the degree of stochasticity would generally determine whether or not to resample transitions. The more difficult decision is on the choice between LSTD and Bellman residual minimizing fits. One the one hand, LSTD usually results in better policies, and in a Bellman error which is orthogonal to the range of the current basis functions. On the other hand, since the algorithm uses the norm of the Bellman error as a criterion, minimizing this quantity may be preferable. Different approaches are explored empririically in Chapter 6. It appears best to use LSTD in the main BFS procedure, however, for some problems the residual minimizing fit seems better when selecting features.

5.6 Dimensionality Reduction

The NCA algorithm is used to find a linear projection A from the state space to $\mathbb{R}^d$. The dimension of the state space is determined by the NCA algorithm as described in Section 4.4. Generally sub-sambling is required, for the purpose of the experiments in this thesis, 400 points and 400 centers are selected for the inner and outter sums of the NCA objective, respectively.

5.7 Feature Pruning

In order to avoid maintaining an excessive number of basis functions as the algorithm progresses, features irrelevant to the current approximation of the value function are removed at each iteration. Two approaches are considered: the first is a general purpose pruning algorithm for neural networks, the second explicitely computes the change in the value function resulting from fitting it without using one of the features.

5.7.1 Optimal Brain Surgeon Feature Pruning

The *Optimal Brain Surgeon* (OBS) algorithm [9]. The method is a successor to magnitude-based pruning and Optimal Brain Damage, and was originally developped to prune weights in large neural networks.

Given a linear approximator $\hat{f}(s) = \phi(s)^\top w$ The weight vector w is assumed to minimize the training error

$$E = \frac{1}{2N} \sum_{k=1}^{N} \left(w^\top \phi(s_k) - f(s_k) \right)$$

The Taylor series of the error with respect to the network weights is

$$\delta E = \left(\frac{\partial E}{\partial w} \right)^\top \delta w + \frac{1}{2} \delta w^\top H \delta w + O(\|\delta w\|^3)$$

If the weights w minimize the error, the first term is zero. Terms of order higher than 3 are ignored, and only the second term involving the Hessian $H = \frac{\delta^2 E}{\delta w^2}$ is retained, implying the assumption that the error is locally quadratic. Thus the estimated change in error for a weight change δw is given by $\delta E = \frac{1}{2}\delta w^\top H \delta w$.

If dealing with very large networks, the inverse Hessian cannot be computed efficiently. A crucial contribution of Hassibi and Stork was to provide a recursive algorithm for updating the approximation. However, for relatively small basis function sets, this is not required.

PRUNEOBS$(\phi(\cdot), [s_t], [r_t], tol)$
repeat
  $w \leftarrow FitVF(\phi([s_t]), [r_t], \gamma)$
  Compute $H^{-1}, L, \delta w$
  $N \leftarrow \dim(w)$
  repeat
    Choose weights $q_1, \ldots, q_n$ for $n \leq N$ such that $\sum_{i=1}^{L}(q_i) \leq tol$
    $w' \leftarrow w + \sum \delta w(q_i)$
    $N \leftarrow n/2$
  until $N = 0$
  $\phi(\cdot) \leftarrow$ rows of $\phi(\cdot)$ with nonzero weight
until Bellman error increase for $\phi([s_t])w'$ is less than tol

Algorithm 4: Optimal Brain Surgeon pruning procedure.

Unfortunately, applying the algorithm directly when using LSTD fitting violates the assumptions on the objective. Indeed, the removal of a basis function which minimally changes a least-squares fit, may substantially change a LSTD fit. Since OBS assumes that the initial fit is the optimum on the training set, though possible overfit, and essentially limits the change induces by the removal of a basis function, it is noted

that this change can also be computed explicitely. This approach is pursued, rather than extending OBS.

5.7.2 Function Change Feature Pruning

Since linear function approximators are being used, a more specific pruning method can be used. Both LSTD and Bellman residual minizing least-squares compute the weights as the solution to a linear system

$$Aw = b$$

Assume that A is full-rank, i.e. that the features are linearly independent and that the sample used to form A and b is sufficiently large. Partition A as

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}.$$

Then the inverse of A can be expressed in terms of the Shur complement of A_{22} as

$$\begin{aligned} B = A^{-1} &= \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} \\ &= \begin{bmatrix} (A_{11} - A_{12}A_{22}^{-1}A_{21})^{-1} & A_{11}^{-1}A_{12}(A_{21}A_{11}^{-1}A_{12} - A_{22})^{-1} \\ (A_{21}A_{11}^{-1}A_{12} - A_{22})^{-1}A_{21}A_{11}^{-1} & (A_{22} - A_{21}A_{11}^{-1}A_{12})^{-1} \end{bmatrix}. \end{aligned}$$

The submatrix of the inverse

$$B_{11} = (A_{11} - A_{12}A_{22}^{-1}A_{21})^{-1}$$

is thus available, and using the matrix inversion lemma [8],

$$\begin{aligned} A_{11}^{-1} &= (B_{11}^{-1} + A_{12}A_{22}^{-1}A_{21})^{-1} \\ &= B_{11} - B_{11}A_{12}(A_{21}B_{11}A_{21} + A_{22})^{-1}A_{21}B_{11}. \end{aligned}$$

The weight change resulting from eliminating the weights of w_2 is

$$\begin{aligned} \delta w &= \begin{bmatrix} A_{11}^{-1}b_1 \\ 0 \end{bmatrix} - A^{-1}b \\ &= \left(\begin{bmatrix} B_{11} - B_{11}A_{12}(A_{21}B_{11}A_{21} + A_{22})^{-1}A_{21}B_{11} & 0 \\ 0 & 0 \end{bmatrix} - \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} \right) b \\ &= - \begin{bmatrix} B_{11}A_{12}(A_{21}B_{11}A_{21} + A_{22})^{-1}A_{21}B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} b \quad \stackrel{\text{def}}{=} \quad \delta B \cdot b \qquad (5.2) \\ &= - \begin{bmatrix} B_{11}A_{12}(A_{21}B_{11}A_{21} + A_{22})^{-1}A_{21}B_{11}b_1 + B_{12}b_2 \\ w_2 \end{bmatrix} \end{aligned}$$

Rather than recomputing the inverse of A_{11}, the new weights after removing a subset of the features can be obtained with this formula. Requiring only the inversion of the smaller matrix. If only a single weight is being eliminated, the weight change is given by

$$\begin{aligned} \delta w_1 &= -\frac{A_{21}B_{11}b_1}{(A_{21}B_{11}A_{21} + A_{22})}(B_{11}A_{12}) - B_{12}b_2 \\ \delta w &= \begin{bmatrix} \delta w_1 \\ -w_2 \end{bmatrix} \qquad (5.3) \end{aligned}$$

since A_{12}, A_{21} and B_{12} are vectors, and A_{22} is scalar. There is a risk of innaccuray due to numerical cancellation when computing the weights in this manner since the weight change δw_1 tends to be very small relative to the two terms of the equation. However, if a higher accuracy is required, this situation can be detected and avoided by explicitely forming the matrix in (5.2) and computing $\delta w = (B + \delta B)b - Bb$.

From the weight change, one can estimate the change in the value of the function by $\epsilon = \|\Phi(w + \delta w) - \Phi w\|_2 = \|\Phi \delta w\|_2$, where Φ is the feature matrix for a representative set of states.

Algorithm 5 greedily prunes features causing minimal change in the function approximator based on two criteria: first, any set of features of which the removal causes a change in the function of less than tol is accepted, and second, as long as there are more than N features, the single feature of which the removal causes the least change in the function is pruned. In the latter case, the function changes are recomputed between each pruning step. In the first case however, the changes are not recomputed so that features having a very small impact on the results may be pruned quickly.

In practice, it may become necessary to recompute the inverse from scratch if the residual of $Aw = b$ becomes too large due to compounded numerical errors. Similarly, the residual of the pruned matrix $A_{11}(B_{11} + \delta B_{11})b_1 = b_1$ may become largre, limiting the number of features which can be pruned per step. Regardless, it is desirable to limit the number of features pruned at a time to bound the time required to compute δB. The verification of these conditions is omitted from the pseudo-code for clarity.

PRUNE$(\phi(\cdot), [s_t], [r_t], tol, N)$
$[w, A, b] \leftarrow FitVF(\phi([s_t]), [r_t], \gamma)$
$B = A^{-1}$
repeat
  for $i = 1, 2, \ldots, n$ **do**
    Compute δw when removing feature i using equation (5.3)
    $\delta_i \leftarrow \|\Phi \delta w\|_2$
  end for
  $I = \{\}$
  while $\exists \delta_i \leq tol$ **or** I is empty and $n > N$ **do**
    $j = \arg\min_i \delta_i$
    Compute δB when removing features $I \cup \{j\}$ using equation (5.2)
    $\delta w \leftarrow \delta B b$
    if $\|\Phi \delta w\|_2 \leq tol$ **or** I is empty and $n > N$ **then**
      $I \leftarrow I \cup \{j\}$
      $\epsilon \leftarrow \|\Phi \delta w\|_2$
      $B' \leftarrow B + \delta B$
    end if
    $\delta_j \leftarrow \infty$
  end while
  $B \leftarrow B'$
  $tol \leftarrow tol - \epsilon$
  Remove rows and columns in I from A, b, B
until $\delta w > tol + \epsilon$ **and** $n \leq N$

Algorithm 5: Pruning procedure

The problem of numerical cancellation in computing the δ_i mentioned above is of little consequence since the weight changes are only used to prioritize the features and this has not been observed to cause any problems using double-precision arithmetic.

CHAPTER 6
Experimental Results

The algorithm is applied to an inventory control problem as well as to extended versions of benchmark problems for reinforcement learning. It is important to note that for all the benchmark problems, the state space is mapped up to a higher dimension with noise.

6.1 Discrete MDPs

Discrete MDPs allow the exact value function to be computed for comparison with the results of the algorithm. Two problems adapted from the litterature are examined: a chain-walk and a three-room robot navigation problem. The first provides a simple vizualization of the value function, while the second represents a relatively large discrete MDP.

6.1.1 Chain Walk MDP

The chain walk MDP is considered by Lagoudakis & Parr [12] and by Parr *et al.* [16]. The version considered here consists of 50 states indexed 1 through 50, with two actions denoted LEFT and RIGHT. The LEFT action moves to the previous state with probability 0.9 and the next state with probability 0.1 while the RIGHT action does the reverse, except in states 1 and 50 where either actions moves to state 2 or 49 respectively. A reward of $+1$ is obtained on a transition out of states 10 and 41, otherwise the reward is 0. The discount factor is 0.9. Clearly, the optimal policy is to move toward the nearer of the states 10 or 41.

For the purpose of testing the algorithm, the states are represented as 20-dimensional vectors obtained by

$$x_t = M \begin{bmatrix} Ps_t \\ w_t \end{bmatrix}, \qquad w_t \in \mathbb{R}^{n/2}, \qquad (w_t)_i \sim N(0, \sigma^2), \tag{6.1}$$

where M is a random but fixed orthogonal matrix, P maps the low-dimensional state to the set $[0, 1]^{n/2}$, and w_t is Gaussian noise. This scheme is used again later, and serves to provide a high-dimensional, noisy state representation of an underlying MDP. The parameter n controls the dimension of the representation, and the parameter σ controls the noise. This scheme allows the underlying state to be recovered by applying the linear transformation $(P^\top P)^{-1} P^\top M^\top$, thus ensuring that NCA can at least recover the underlying state representation (though this may not be the best NCA solution).

For the chain walk problem discussed here, $n = 20$, $\sigma = 0.01$ and P is a 10 dimensional vector with all entries set to $\frac{1}{49}$. The LEFT and RIGHT actions are represented by the scalars -1 and $+1$ respectively. Thus the action value function $Q(x, a)$ is a mapping from the 21-dimensional vector obtained by concatenating the state and action representation, to the reals. This is a somewhat unusual approach, since a seperate set of bases is often used for each action. However, using a single approximator may be more economical when the action-value functions for different actions are similar.

The LSPI algorithm is applied with samples of 5000 random transitions out of uniformly selected random states, with an exploration rate of 0.4 (the optimal action is chosen with probability 0.8). At most 100 basis functions are kept in total, and at most

25 are added per iteration. Figures 6–1 and 6–2 show the action-value functions obtained after the first six policy evaluation and policy iterations steps respectively.

In Figure 6–1, we see that on the first iteration the value of each state is already fairly well approximated (Recall that the initial policy is random). However, there is no difference between the values for the LEFT and RIGHT actions since NCA has identified the more important position dimension first. Subsequent iterations result in the value function seen in the first panel of figure 6–2. This should be contrasted with a similar experiment presented in [16] using exact Bellman error basis functions. With the latter approach, the support of the early basis functions is a single state, since the Bellman error for a constant value function is zero everywhere except at the states with positive reward. The difference is due to the fact that the correction MDP is solved approximately here. The difference between the estimated and optimal action-value function is plotted in figure 6–4.

We see that the algorithm has essentially converged to the optimal policy after the fourth policy iteration, though it does not achieve the optimum do to the positive exploration rate. Figure 6–3 shows the final action value function obtained. The action-value function plot is generated by mapping sampled states according to (6.1) without noise, and evaluating the approximator at the resulting points. Note that the controller only evaluates the function at the points shown, so the value at other points is irrelevant.

6.1.2 Three-room MDP

The algorithm is evaluated on a three-room MDP similar to one described by Mahadevan [13]. The discrete state space consists of three 11-by-10 rooms, connected by doors of width 1. The goal of the agent is to reach the upper right corner of the final

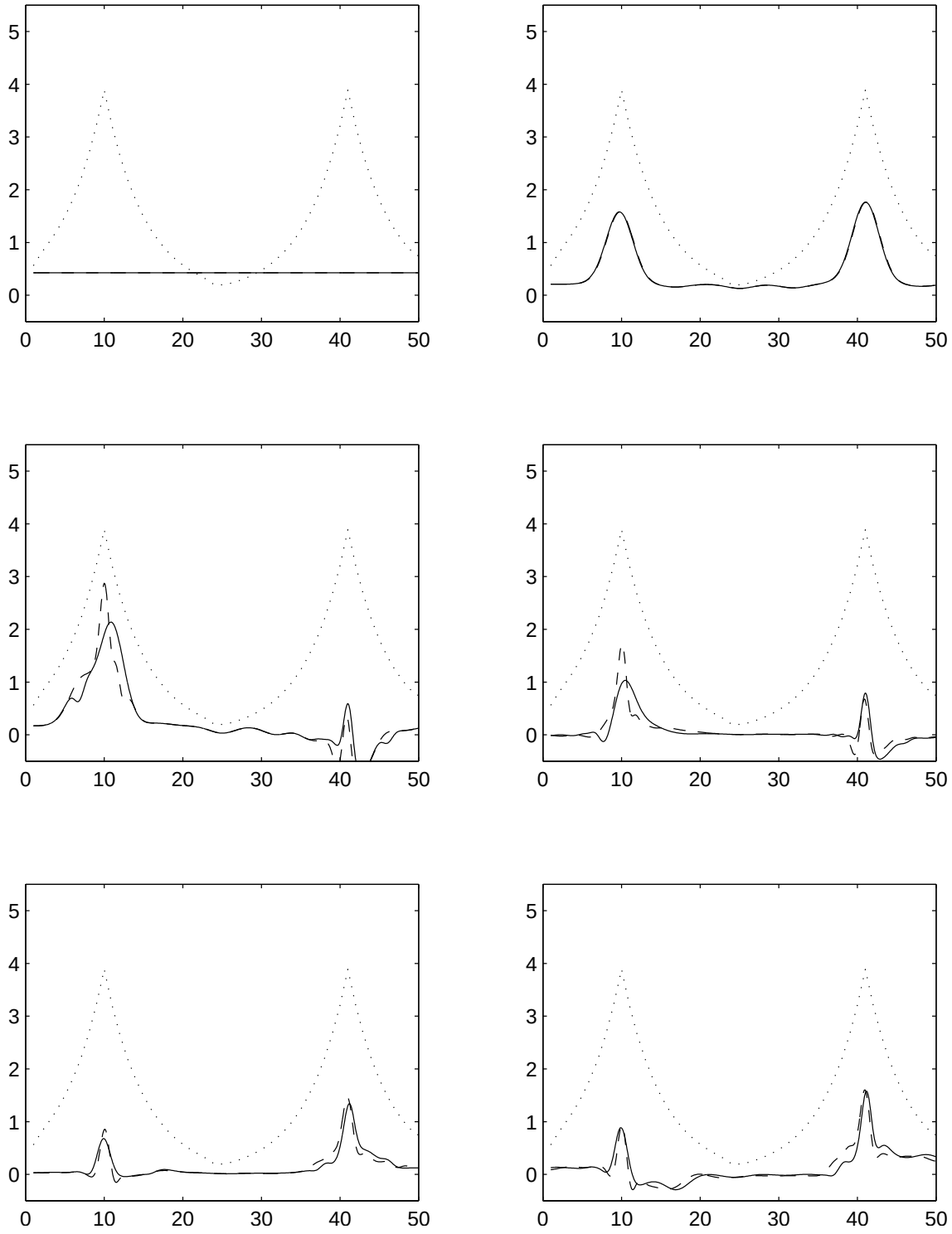

Figure 6–1: Action-value functions computed for the chain walk in the first six iterations of the algorithm. Panels progress from left to right and then top to bottom. The solid curve represents the estimated value of the LEFT action and the dashed curve that of the RIGHT action. The dotted curve shows the optimal value function.

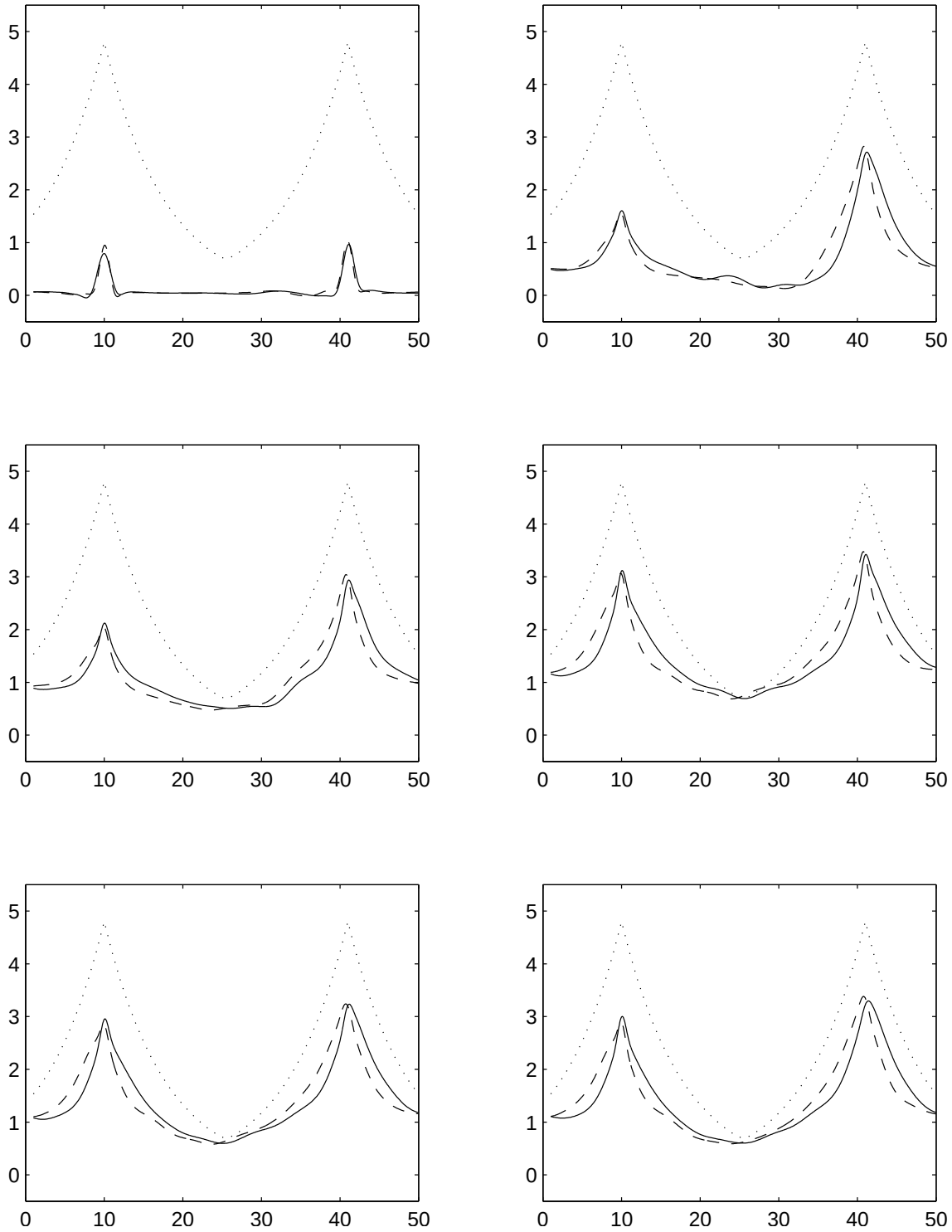

Figure 6–2: Action-value functions computed for the chain walk for the first six policies. Ten iterations of the algorithm are executed for each policy, starting with the basis functions from the previous policy. Panels progress from left to right and then top to bottom. The solid curve represents the estimated value of the LEFT action and the dashed curve that of the RIGHT action. The dotted curve shows the optimal value function. Furter policy iterations do not significantly change the value funtion from the last panel, where the true value function is nearly attained. It is less than the optimal value function due to the exploration policy, but induces the optimal policy.

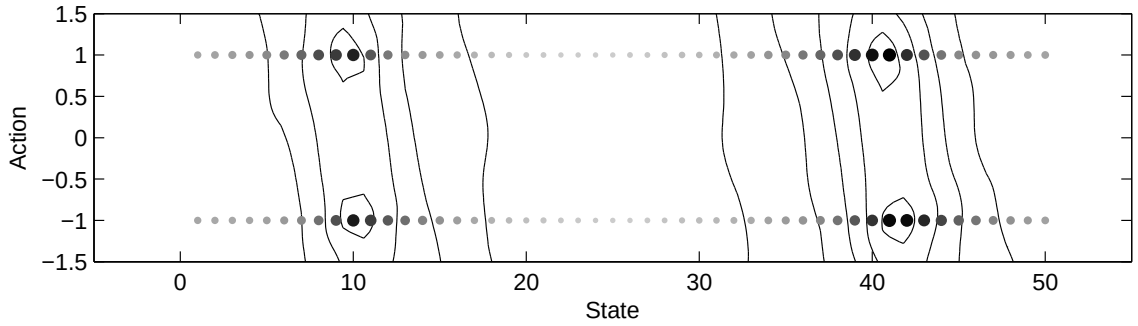

Figure 6–3: Action-value function for the chain walk after 10 policy iterations, each interleaved with 10 iterations of the algorithm. The size of the dots is proportional to the estimated value for the 100 state-action pairs.

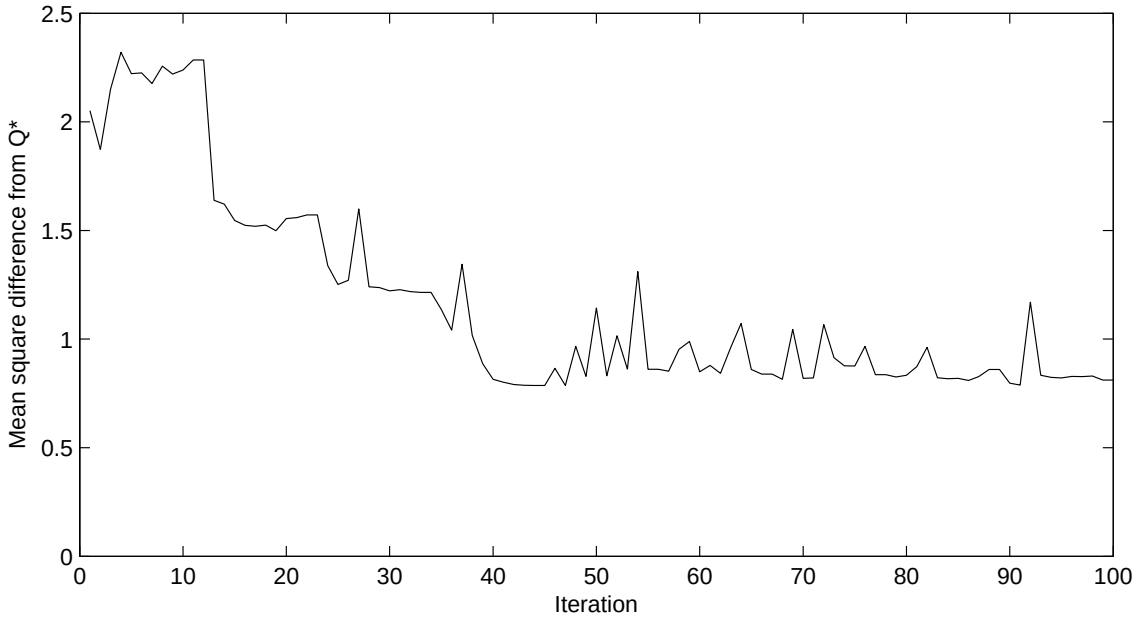

Figure 6–4: Difference between the action-value function approximation for the chain walk and the optimal action-value function.

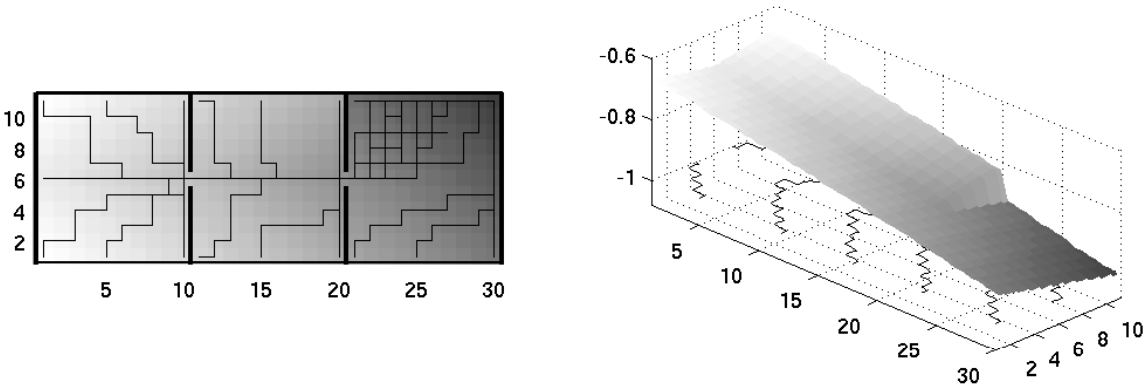

Figure 6–5: Optimal value function for the Three-Room problem, with a sampling of optimal trajectories. The goal is in the upper right corner at coordinates $(30, 11)$. Note the discontinuity due to the wall between the second and third rooms.

room, where a reward of +1 (or negative cost of -1, here) is obtained. There are four possible actions for the agent: it may move UP, DOWN, LEFT or RIGHT. The state space and optimal value function are shown in Figure 6–5. The formula 6.1 is used with $\sigma = 0.01$, $n = 20$ and P mapping the two-dimensional state space to a 10-dimensional vector containing five copies of each coordinate normalized to an interval of unit length.

The problem poses a challenge for two reasons. First, the walls induce a discontinuity in the value function. This issue is handled well by manifold-based techniques or with proto-value functions, but causes difficulties when smooth basis functions are used as is the case here. Second, the doorways are bottleneck states, since any transition between rooms passes through them, and thus the accuracy at theses states has a large impact on the entire value function estimate. Thus it may be desired to place many local basis functions near the doors.

Fixed training and validation samples consisting of 840 trajectories starting at randomly selected positions and following the optimal policy are used. This yields

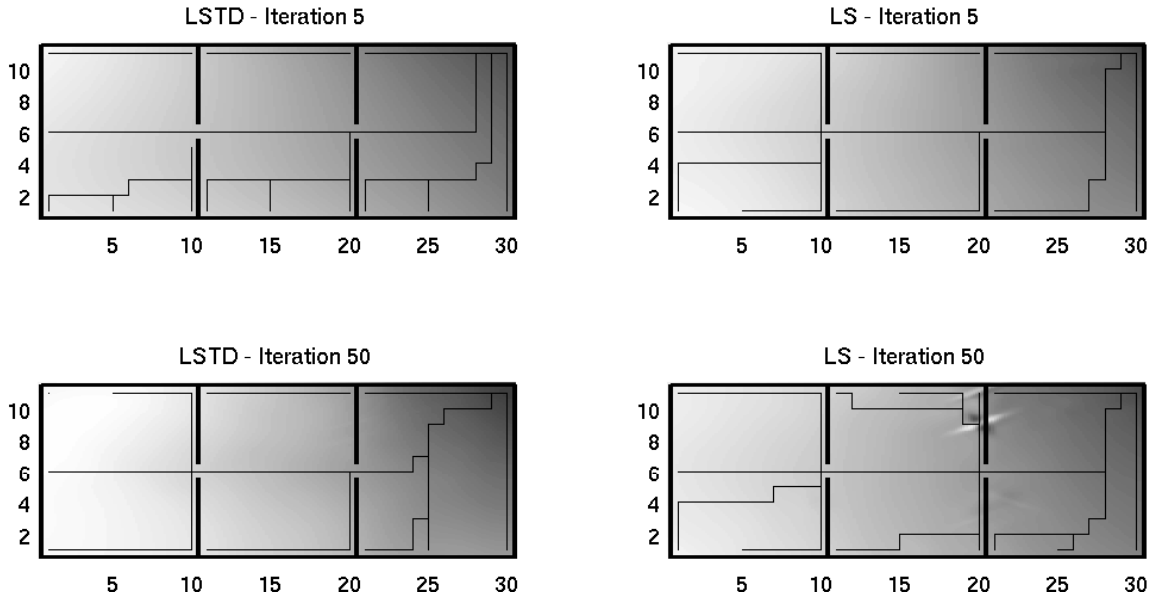

Figure 6–6: Value function approximation for the Three Room problem using the least-squares and LSTD algorithms, along with induced trajectories. After only a few iterations, the induced policies result in the agent getting "stuck" in the top half of the middle room. This effect is due to difficulty in representing the discontinuity in the value function. The LS algorithm eventually places small, local basis functions around the wall, resulting in a near-optimal policy. The LSTD algorithm does not resolve the problem.

approximately 35000 transitions in total. The algorithm is applied using both Bellman residual minimizing fits (denoted LS) and LSTD fits. Figure 6–6 illustrates the value function estimates and policies obtained in either case. While both methods yield good approximations of the value function after few iterations, only LS achieves an optimal policy. This is due to the fact that LSTD fails to place local basis functions near the discontinuity in the value function, and thus underestimates the value function in states near the top of the middle room near the wall.

Figure 6–7 shows the performance of the two versions of the algorithm, along with the case where Bellman residual minimizing fits are used but the underlying MDP is not solved. That is, the Bellman error is approximated directly when constructing new basis functions. As may be expected, minimizing the Bellman error results in lower Bellman errors in the approximation. However, LSTD provides a more accurate appximation to the value function. There is little difference whether or not the correction MDP is solved. Yet only the Bellman residual minimizing algorithm which solves the correction MDP achieves the optimal policy. This is surprising since LSTD is generally assumed to provide better policies in practice, and it does indeed yield a better approximation of the true value function.

Under the conjecture that this effect may be due to the particular state representation, the experiment is repeated with a different encoding. In Figure 6–8, the state is represented as a 330-dimensional vector with all elements set to 0 except the index corresponding to the current state, which is set to 1. Thus there is no notion of distance between states intrinsic to the state representation (besides the ordering of state variables, which has no impact on the algorithm). The use of sparse matrices where possible limits

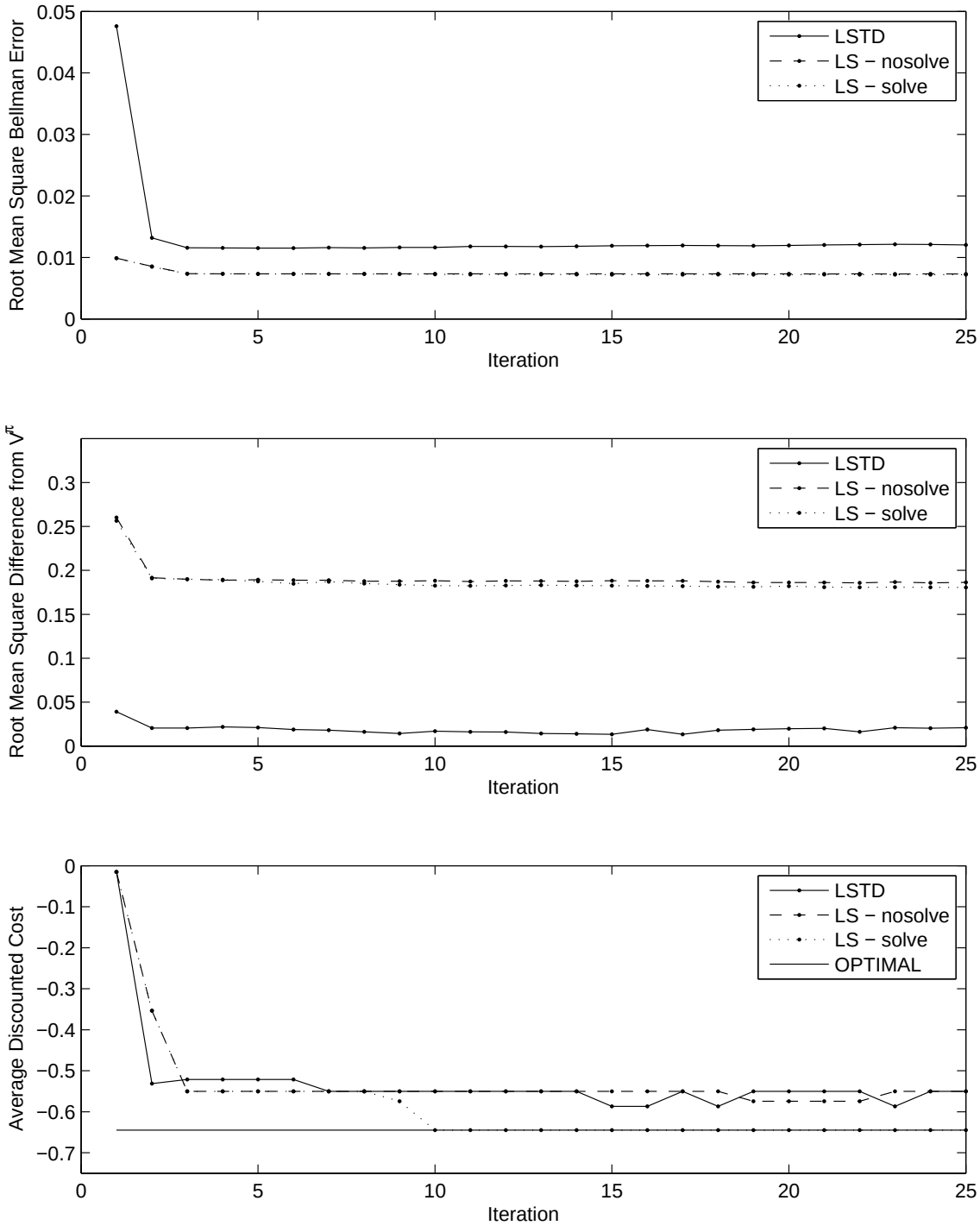

Figure 6–7: Bellman error, value function error, and induced policy performance for the three-room problem. The LS algorithms achieve a lower Bellman error but a worse approximation of the optimal value function. Only the LS algorithm achieves the optimal performance, and only when solving the correction MDP. The average costs are computed as the mean discounted cost over 440 trajectories starting at random but fixed locations.

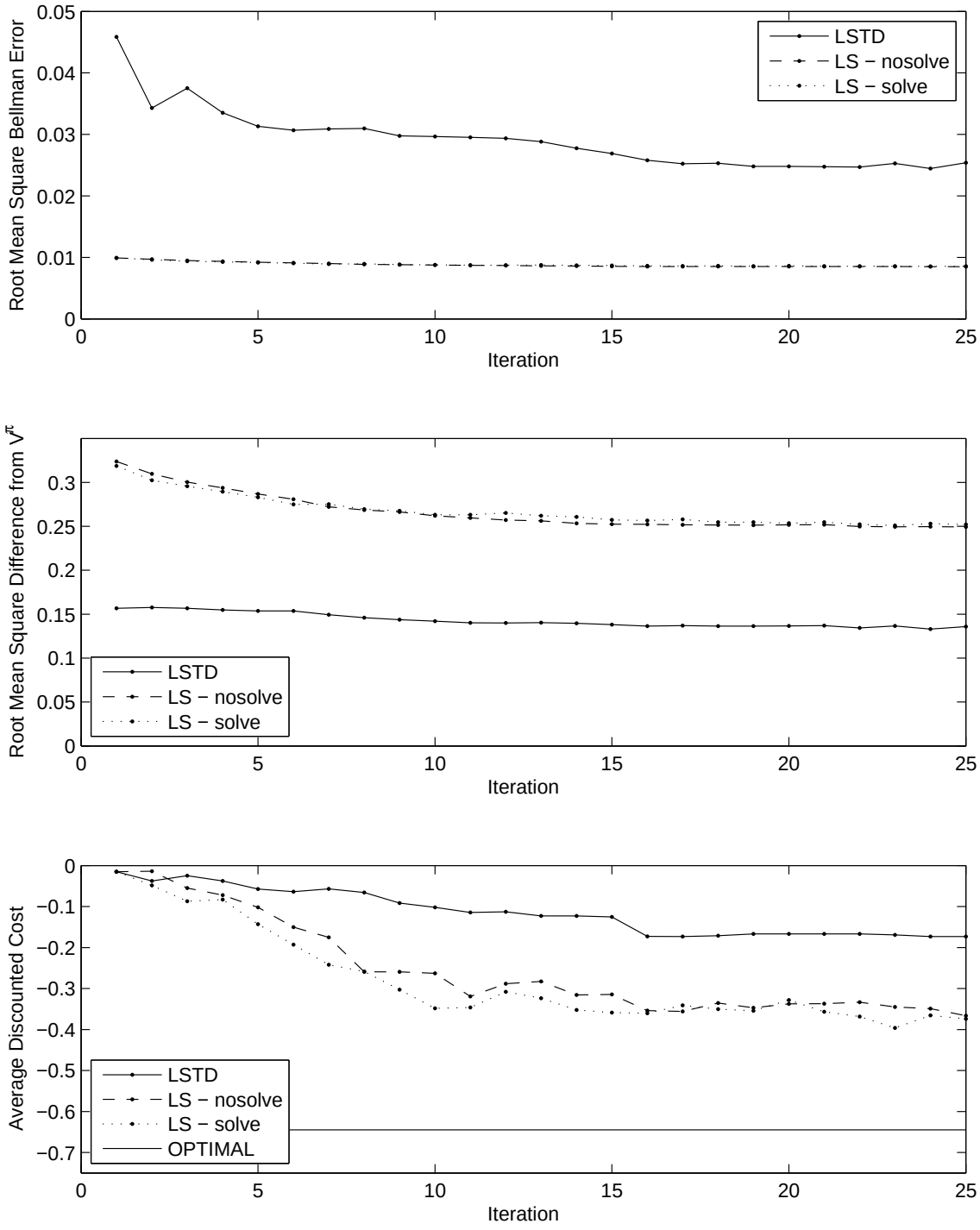

Figure 6–8: Same experiment as in Figure 6–7 when using the sparse state representation. The LS algorithm again achieves a lower Bellman error but higher approximation error. The LS algorithm yields a significantly better policy, but learning is very slow for both algorithms.

memory and computation requirements close to those when using the original representation. Learning transformations from this representations to a low dimensional space is thus closely related to the concept of state aggregation considered by Bertsekas and Castañon and discussed in Section 3.1. Though the performance of the algorithm in this case is much worse, the effect remains, and LSTD yields significantly worse policies than Bellman residual minimization. It should be noted that the perfomance levels of both algorithms corresponds to cases where the agent only reaches the goal when starting in certain states of the rightmost room, or near the middle of the other rooms. The poor performance is due to the removal of important state connectivity information from the representation.

6.2 Mountain Car

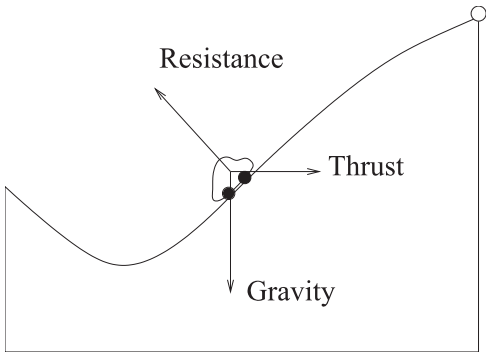

Figure 6–9: In the mountain car problem, the goal is for the car to reach the top of the hill with zero velocity. At each time step, the thrust may be either negative or positive. However, the car is not powerful enough to climb the hill if it starts partway up at a low speed. In such a case, it must first back up and then accelarate to gain enough momentum to reach the goal. A cost of 1 is incurred at each time step, except when the car reaches the goal or exceeds the minimum position or maximum velocity. If the car reaches the goal, the final cost varies from 0 to 100 proportionally to the final velocity. In the other cases, a final cost of 100 is incurred. A discount factor of 0.99 is used.

The mountain car problem, illustrated in figure 6–9 is a well-studied reinforcement learning benchmark [24, 18, 17]. Despite having only a two-dimensional state-space, it presents many of the difficulties commonly encountered in reinforcemenet learning. The algorithm is tested on the mountain car problem simulator included with the *CLSquare* simulation system [19], with the state space dimension increased to $n = 20$ using equation (6.1), and with the matrix P mapping the position and velocity state variables to a 10-dimensional vector containing 5 normalized copies of each. Unless otherwise mentioned, the noise parameter is set to $\sigma = 0.01$.

A sample of approximately 10,000 transitions is drawn according the near-optimal policy obtained by policy iteration using a tile coding approximator, with five 9-by-9 uniform tilings of the state space. At most 25 new basis functions are added at each iteration, and at most 100 basis functions are maintained in total. The LSTD or the Bellman residual minimizing algorithm is used to fit the value function, and the simulator's model is used to select actions based on the value function estimate.

Figure 6–10 shows a contour plot of the final value function approximations obtained when building features to approximate the Bellman error directly and when creating features to solve the correction MDP. Figure 6–11 shows the corresponding trajectories. The latter version of the algorithm produces a smoother approximation. Indeed Figures 6–12 and 6–13 show the evolution of the approximations. When the Bellman error is approximated direcly, the first features are localised in the areas of the state space with large cost differences. In this case, most sample trajectories reach the goal with low velocity, thus the Bellman error is identical almost everywhere except at

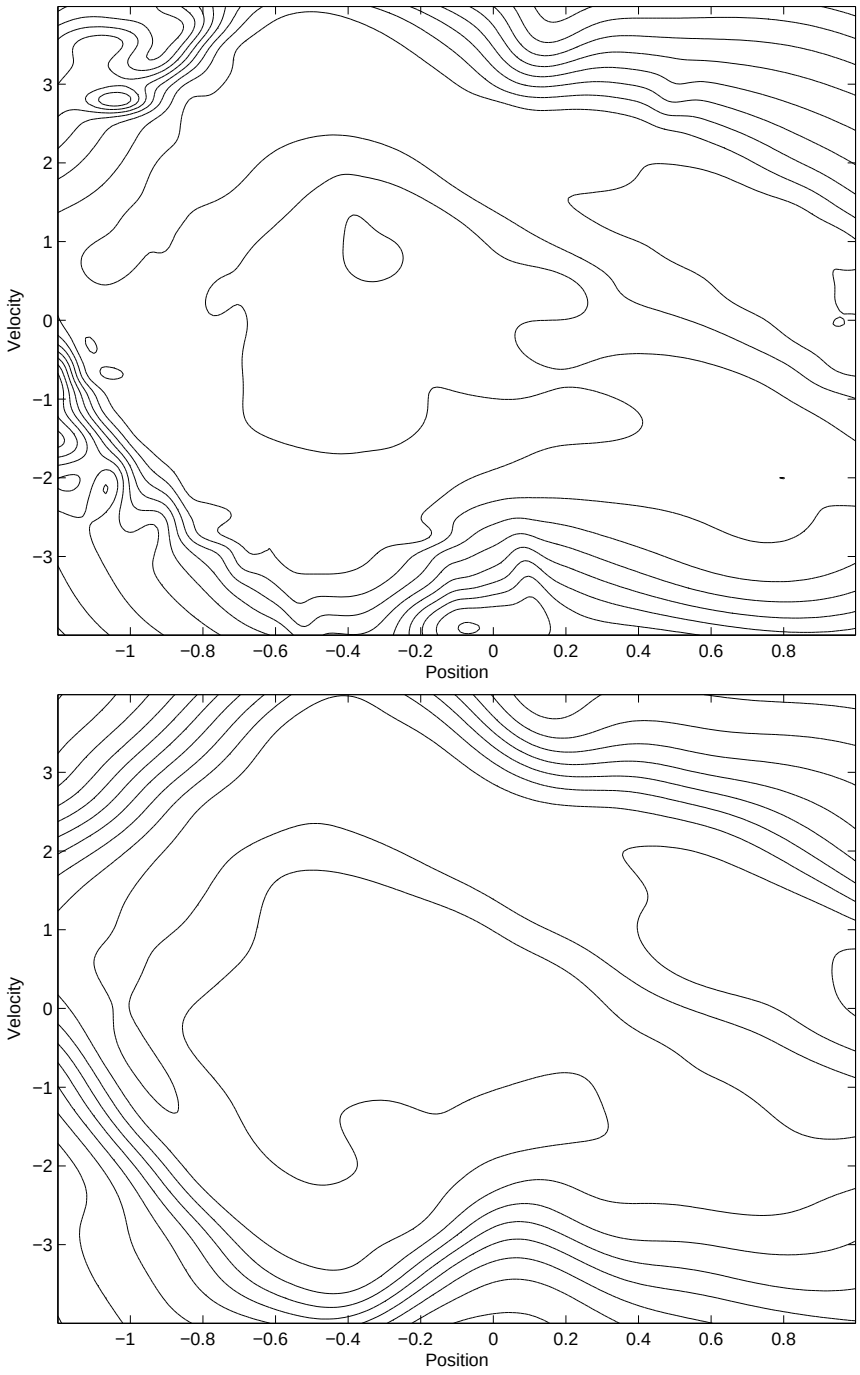

Figure 6–10: Mountain Car problem value function after 12 iterations when fitting only the Bellman error (above), and when solving the correction MDP with $K_2 = 5$ (below). Both algorithms yield approximations of similar quality, but the solution is much smoother when solving the correction MDP since the basis functions tend to be less local.

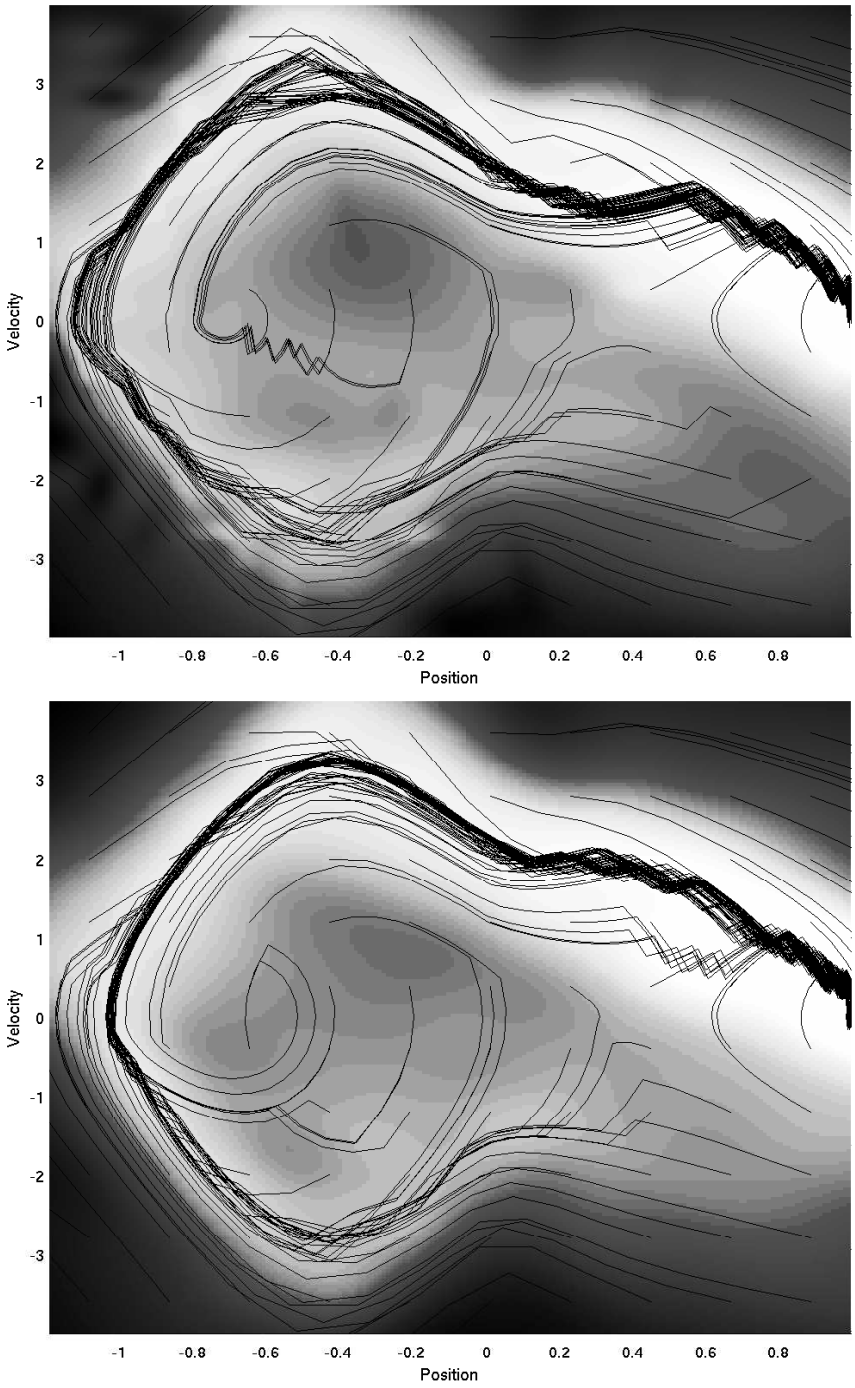

Figure 6–11: Mountain Car problem trajectories induced by the value function after 12 iterations when fitting Bellmand error directly (above) and when solving the correction MDP with $K_2 = 5$ (below).

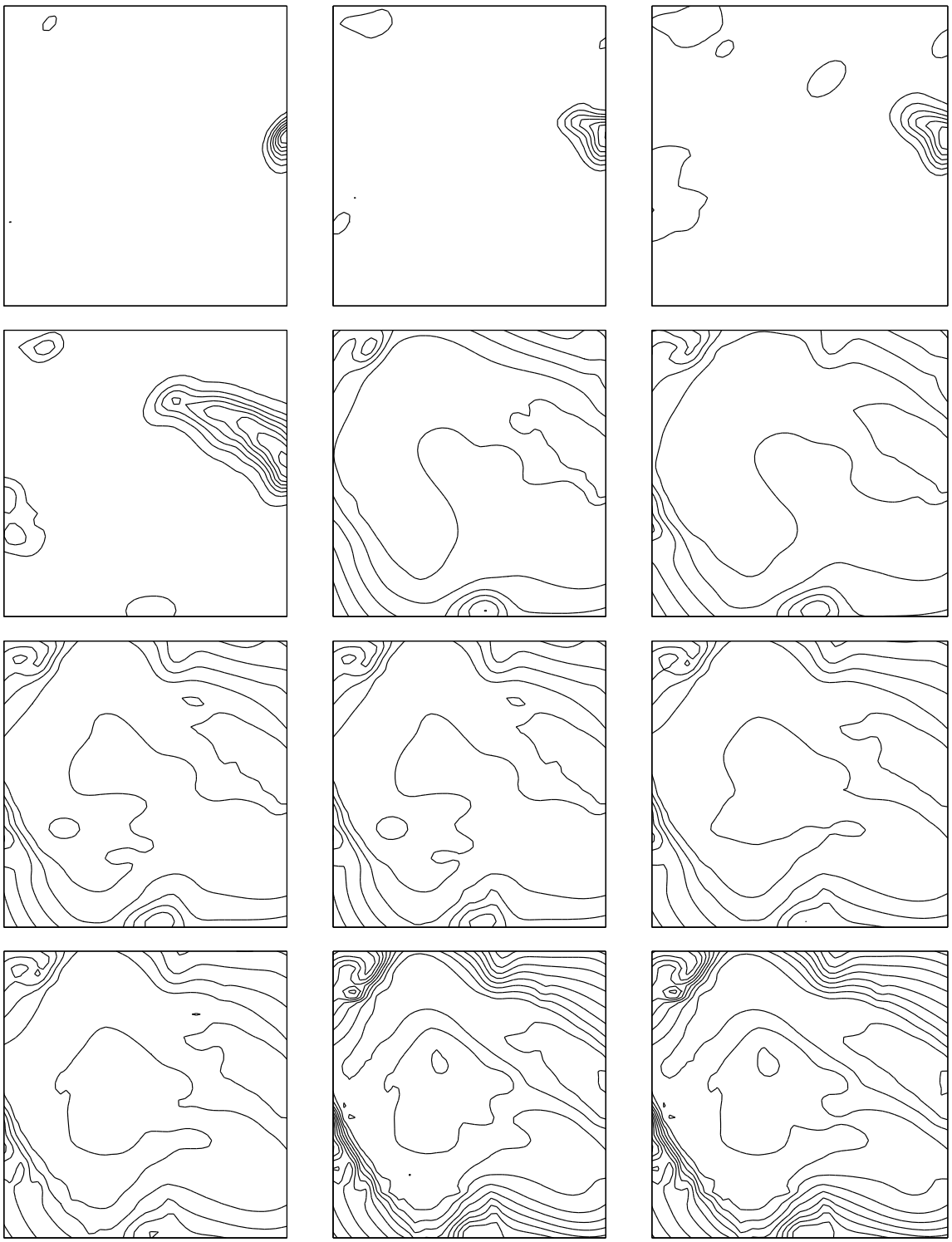

Figure 6–12: Mountain Car value function after each of the first twelve iterations when fitting the Bellman error directly. One can observe that local features are added where the Bellman error is significan, and then “spread” accross the state space in a manner reminiscent of table-based dynamic programming.

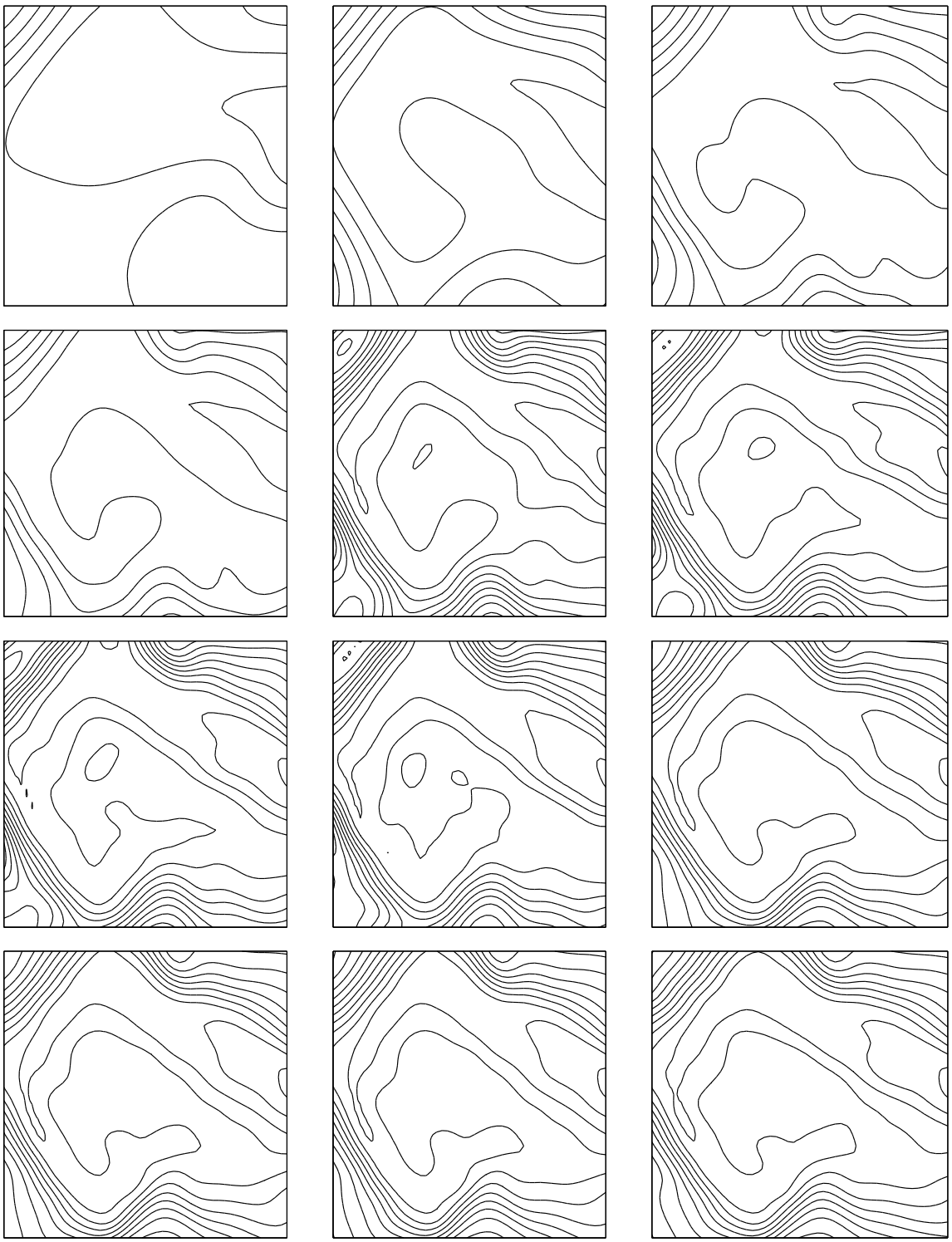

Figure 6–13: Mountain Car value function after each of the first twelve iterations when solving the correction MDP with $K_2 = 5$. In contrast to Figure 6–12, the initial features allow a rough approximation of the entire state space. Subsequent iterations refine the approximation.

the states terminating a trajectory. Subsequent iterations propagate the value across the state space in a manner akin to classic dynamic programming in finite MDPs.

In contrast, when the correction MDP is solved approximately, the early features are much less localised, and better able to represent the actual value function. Figure 6–14 shows that the second algorithm learns features much faster, in terms of the reduction in Bellman error, as well as in terms of minimizing the cost. This holds whether using Bellman residual minimizing fits (LS) or least-squares TD (LSTD). Both LSTD algorithms achieve a performace comparable to the tile-coding approximator, while the LS algorithms yield much less stable performance, but lower Bellman error. This last point contrasts with the results for the three room problem, but confirms the accepted wisdom that LSTD provides better policies in practice.

6.2.1 Robustness to Noise

To explore the effect of the noise in the state representation, the mountain car problem is solved using different values for the parameter σ when mapping the state space to higher dimensions. Figure 6–15 shows the degradation of the approximation's Bellman error and the performance of the resulting policy as the noise increases. Recall that the elements of the state vector are normalized to the range $[0, 1]$, thus setting the standard deviation to $\sigma = 0.5$ for instance, represents a significant amount of noise. Though the Bellman error increases with noise, the performance loss is small for $\sigma \leq 0.5$.

6.3 Inventory Control

The algorithm is applied to the inventory control problem in order to improved an existing value function approximation, rather than learn one from scratch. Unlike

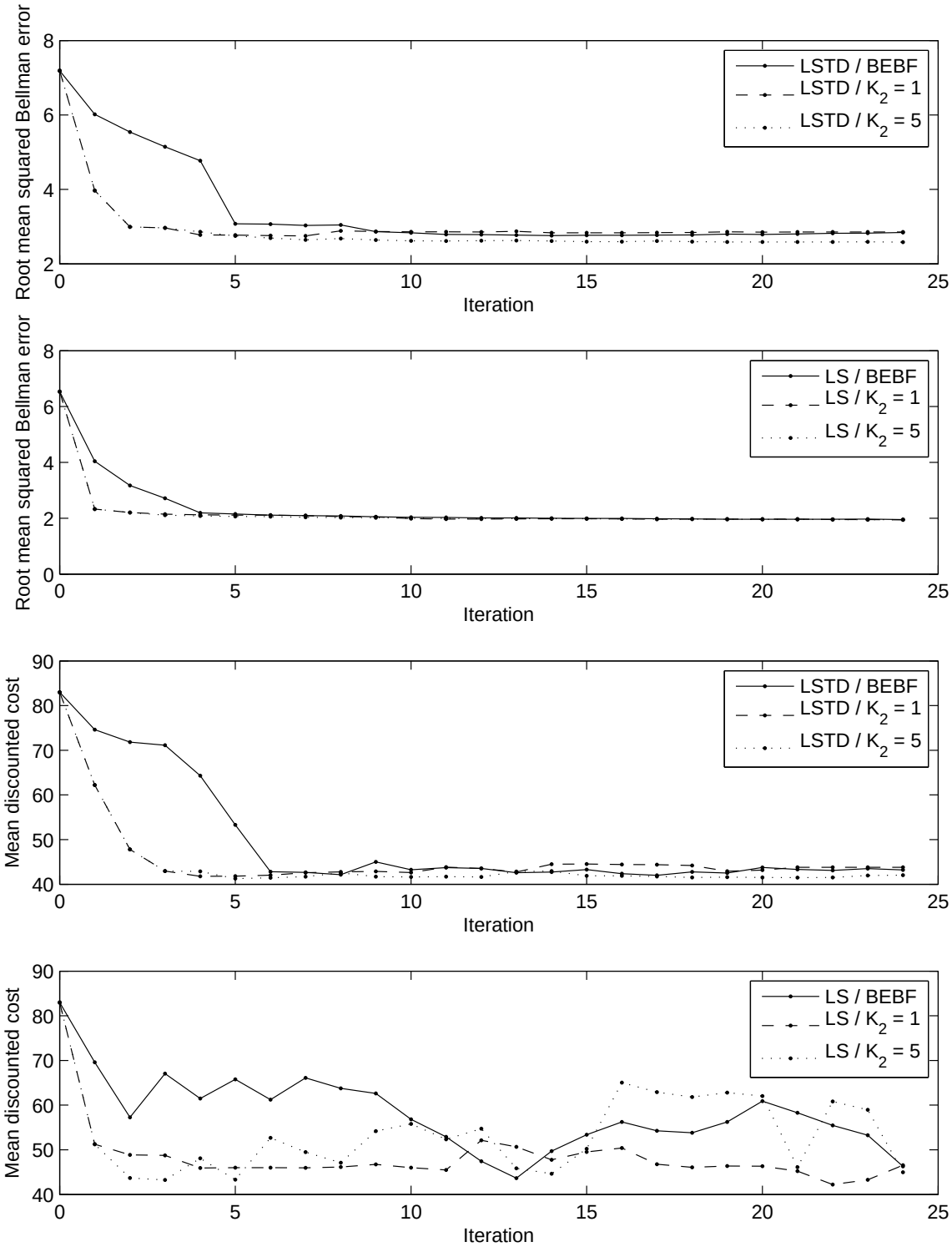

Figure 6–14: Mountain Car problem Bellman error when using the Bellman residual minimizing algorithm (LS) and LSTD. For the LS algorithm, the Bellman error is smaller, but the performance is worse and less stable. In either case, learning is much faster when solving the correction MDP, but the effect is more pronounced for LSTD. The two bottom panels show the mean discounted costs over 10000 steps with identical random initial positions.

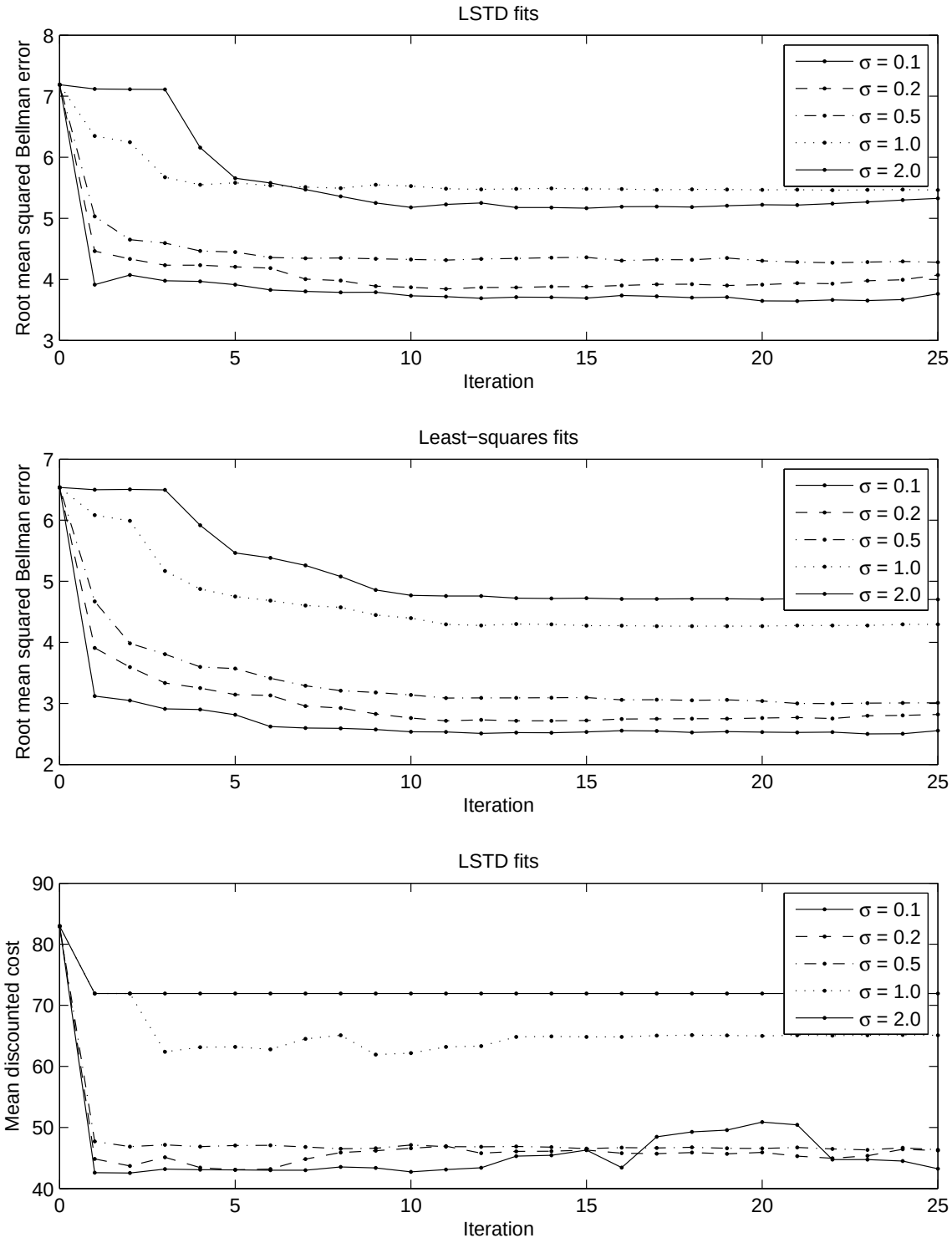

Figure 6–15: Mountain Car problem Bellman error for varying amounts of noise in the state representation. The top panel shows the Bellman error of the LSTD fit, the center panel shows the Bellman error of the residual minimizing fit when using the basis functions yielded by LSTD (in contrast to using LS to generate the basis function also), and the bottom panel shows the mean discounted costs over 10000 steps with identical random initial positions.

the other problems considered, the state space intrinsically has a hign dimension. Two instances are considered, the first where the state consists of the inventory of 16 products with correlated demand, and the second deals with 64 products.

6.3.1 Problem Description

In the inventory control problem, the goal is to maintain an inventory of products such as to minimize a cost function. The state is denoted by $x_t \in \mathbb{R}^n$ where element $(x_t)_i$ represents the inventory of the i-th product at the beginning of period t. The state is updated at each time period according to

$$x^{t+1} = \min(\max(x_t + u_t, x^{min}), x^{max}) - w_t,$$

where u_t is the quantity of each good ordered (or salvaged, if negative) at the beginning of the period, and w_t is a random variable representing the demand in time period t. The demand w_t is normally distributed according to $N(\mu, \Sigma), \Sigma_{i,j} = \rho_{ij}\sigma_i\sigma_j$, thus the demand may be negative in a given period. At the beginning of time period t, the decision maker is presented with the inventory x_t and decides on order quantities u_t. The bounds on the achievable inventory levels serve to limit the domain of the value function. The actual end of period inventory level may exceed the bounds. However, the order quantities are constrained to be in the range

$$\max(-x_t^+, x^{min} - x_t) \leq u_t \leq x^{max} - x_t$$

to ensure the order achieves at least the minimum inventory level, salvages at most the current inventory level, and salvages at least the excess inventory. The shorthand x^+ and

x^- denote $\max(x, 0)$ and $\max(-x, 0)$ respectively. Subject to these constraints, the state transitions simplify to $x^{t+1} = x_t + u_t - w_t$.

The cost at each time period is given by

$$f_t = f(x_t, u_t) = k^\top \text{sign}(KI_{u_t>0}) + c^\top u_t^+ + d^\top u_t^- + h^\top x_{t+1}^+ + p^\top x_{t+1}^- + q\left(\sum_i (x_t)_i^+ - R\right)^+,$$

where the parameter vectors are as defined in Table 6–16, $\text{sign}(\cdot)$ returns a vector with positive, zero and negative entries replaced by $1, 0$ and -1 respectively, and $I_{u_t>0}$ is a binary vector with the i^{th} entry indicating whether $(u_t)_i$ is positive. The product group matrix K partitions the products into sets which share a fixed order cost incurred whenever a positive amount one of the grouped products is ordered. For instance, the parameters

$$K = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \qquad k = \begin{bmatrix} 1 \\ 7 \\ 3 \end{bmatrix}$$

would specify that a fixed cost of 4 is incurred when ordering a positive amounts of products 1,3 and 6 since these belong to the first and third groups.

6.3.2 Inventory Control Policies

A class of policies known as (s, S) policies is well studied studied in inventory control litterature. Here $S \in \mathbb{R}^n$ is a vector of target inventory level, and $s \leq S$ is a vector of re-order levels. The order quantities are determined as

$$(u_t)_i = \begin{cases} S - (x_t)_i & \text{if } (x_t)_i \leq s_i \\ 0 & \text{otherwise} \end{cases}.$$

Parameter	Description	Example value
n	Number of products	64
m	Number of product groups	2
c_i	Unit order cost	1
d_i	Unit salvage cost	2
h_i	Unit holding cost	0.5
p_i	Unit backlog penalty	10
q	Unit excess inventory penalty	10
R	Total Inventory capacity	$10n$
μ_i	Product demand mean	1
σ_i^2	Product demand variance	1
ρ_{ij}	Product demand correlation	$[-1, 1]$
k_ℓ	Product group ℓ fixed order cost	1
K	Product group matrix ($m \times n$)	
x_i^{min}	Minimum inventory level	-3
x_i^{max}	Maximum inventory level	20

Figure 6–16: Inventory control problem parameters. The indices i and j range over the set of products $\{1, 2, \ldots, n\}$. The index ℓ ranges over the product groups $\{1, 2, \ldots, m\}$. The quantities q and R apply to all products.

That is, the quantity $(S_i - (x_t)_i)$ is ordered whenever the inventory of the i^{th} product has fallen below the re-order level. Otherwise no order is placed for that product.

For the single-product case, it is known that policies of this form are optimal. A policy iteration algorithm for computing optimal (S, s) policies with continuous demands without relying on discretizations was developped in 1985 [6]. An efficient algorithm for the case of discrete demands is known [27], and an efficient algorithm for the continuous demands has been developped [6]. However, these methods apply only to the single-product case, and an extensive review is beyond the scope of this thesis.

In this work, (s, S) policies are used to determine an initial value function. A smooth approximation to the value functions of (s, S) policies is used as a starting point for the policy iteration algorithm applied to the general multiple product case.

If the demands are assumed to be deterministic and there is a single product with fixed order cost $\bar{k}$, the value function under an (s, S) policy is given by the following equation, where $T = \lfloor \frac{x-s}{\mu} + 1 \rfloor^+$ is the number of time periods remaining until the inventory falls below the re-order level when starting at $x_0 = x$.

$$\begin{aligned}
V_0(x) &= \sum_{t=0}^{T-1} \gamma^t f(x - t\mu, 0) + \gamma^T f(x - T\mu, S - x + T\mu) + \gamma^{T+1} V_0(S - \mu) \\
&= \sum_{t=0}^{T-1} \gamma^t h(x - T\mu - \mu) + \gamma^T \left(\bar{k} + c(S - x + T\mu) + h(S - \mu)\right) + \gamma^{T+1} V_0(S - \mu) \\
&= h(x - \mu)\frac{1 - \gamma^T}{1 - \gamma} - h\mu\frac{1}{(1 - \gamma)^2}\left(\gamma - \gamma^T - (T - 1)\gamma^T(1 - \gamma)\right) + \\
&\qquad \gamma^T \left(\bar{k} + c(S - x + T\mu) + h(S - \mu)\right) + \gamma^{T+1} V_0(S - \mu),
\end{aligned}$$

where one uses the fact that $\sum_{t=1}^{T} t\gamma^t = \gamma(1 - \gamma)^{-2}\left(1 - \gamma^T - T\gamma^T(1 - \gamma)\right)$ to obtain the final expression. The penalty terms of the cost function are ignored since an

appropriate (s, S) policy avoids any shortages and excesses under deterministic demand. The quantity $V_0(S-\mu)$ does not depend on x. Furthermore, since the inventory will be replenished to $S-\mu$ followind the next re-order point, the constant $V_0(S-\mu)$ in the last term can be expressed as

$$V_0(S-\mu) = \sum_{t=0}^{\infty} \left(\gamma^{T^{(S,s,\mu)}+1}\right)^t \cdot \left(\sum_{t=0}^{T^{(S,s,\mu)}-1} \gamma^t f(S-\mu-t\mu, 0) + \gamma^{T^{(S,s,\mu)}} f(S-\mu-T^{(S,s,\mu)}\mu, \mu+T^{(S,s,\mu)}\mu) \right),$$

where $T^{(S,s,\mu)} = \left\lfloor \frac{S-\mu-s}{\mu} + 1 \right\rfloor^+$ is the number of time periods between orders. This quantity can be evaluated by taking the constant factor outside of the summation and computing it in a manner similar to the closed-form terms of $V_0(x)$ above. Thus $V_0(x)$ itself can be expressed in closed form, and is differentiable everywhere $T \neq 0$. A smooth approximation $\hat{V}_0$ to V_0 is obtained by replacing T with $\left(\frac{x-s}{\mu} - \frac{1}{2}\right)^+$ to approximate the floor function. A value function with fewer breakpoints facilitates optimization, and since the final aim is to model a problem with stochastic state transitions, it is reasonable to locally smooth the value function.

The choice of the parameters (s, S) nonetheless determines the initial policy. The inventory levels and bounds are set to

$$s_i = \mu + \sigma_i \qquad S_i = \frac{\mu_i}{\sum_{i=1}^n \mu_i} R + (\mu_i - \sigma_i)$$

$$x_i^{min} = s_i - 2\mu_i \qquad x_i^{max} = S_i + 2\mu_i$$

The target inventories and re-order levels are chosen to avoid excess inventory and backlog penalties when the next demand falls within one standard deviation of the mean.

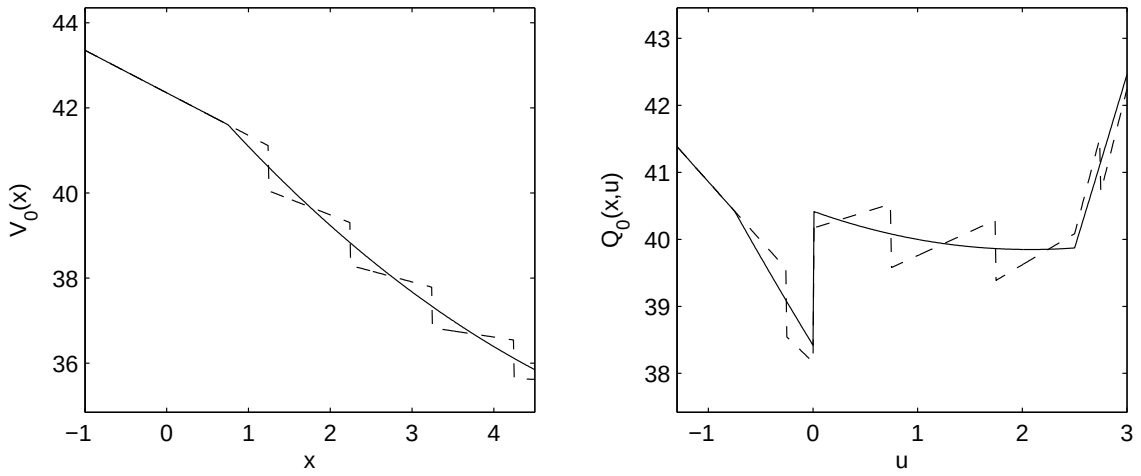

Figure 6–17: Initial value function approximation for $\mu = 1, c = d = 1, \bar{k} = 2, h = 0.2, p = q = 5, R = 4, s = 0.25, S = 4.75$ and $\gamma = 0.95$. The dashed lines represent the exact value functions under deterministic demand. The right panel shows the action-value function when the current inventory level is $x = 2.5$. The optimal order quantity is zero.

The target inventory S_i is somewhat conservative since the variance of the total inventory will be equal to the sum of the variances of the individual inventories. The choice of S_i is motivated by the fact that, if the holding cost is sufficiently low, it is optimal to fill the entire inventory capacity. The choice of inventory bounds is somewhat arbitrary and assume that the inventory levels achieved under the optimal policy will not deviate much from those achieved under the initial policy.

When considering multiple products, the value function is taken to be the sum of the value functions for the individual products, using $\bar{k} = K^\top (KK^\top)^{-1} k$ as individual fixed ordering costs. This provides an initial guess for the value function of the problem with stochastic demands. An (s, S) policy for the multiple product case would ignore the joint fixed order costs k and the joint inventory excess penalty. There would be no

coordination between orders for different products. Instead the policy induced by the initial value function is used for sampling.

Given any value function V, the action-value function Q is approximated by the action-value function $\hat{Q}$ under deterministic demands

$$\hat{Q}(x, u) = f(x, u) + \gamma V(x - \mu),$$

which is differentiable where its two terms are. Thus the policy induced by a value function V may be approximated by selecting order quantities u_t minimizing $\hat{Q}(x_t, u)$ given the current state x_t. In practice, a conjugate-gradient algorithm with multiple starting points is used to find approximate minima of the action-value function. Neither f nor V is assumed to be convex, though both are well-behaved enough to allow effective optimization if appropriate safeguards are taken.

In addition to any breakpoints of V, the cost function f has discontinuities due to the fixed ordering costs. For this reasons a seperate search is done where the order quantity is constrainted to be zero for each possible subset of products groups. Under this scheme, the choice of product groups is actually made from a discrete set of possible actions, and the order quantities within a group are continuous decision variables. The approach results in satisfactory practical performance.

In order to ensure that a relevant subset of the state space is visited during sampling it may be necessary to add an exploration term to the policy. Thus the order levels are

perturbed as

$$u_i = \begin{cases} (u_i^* + \epsilon_i)^+ & \text{if } u_i^* > 0 \\ (u_i^* + \epsilon_i)^- & \text{if } u_i^* < 0 \\ 0 & \text{if } u_i^* = 0 \end{cases}, \qquad \epsilon_i \sim N\left(0, \left(\left(\frac{\mu_i}{4}\right)^2 - \sigma_i^2\right)^+\right),$$

where u^* contains the estimated optimal order quantities. The variance of the noise term is chosen so that the conditional variance of the inventory $x_{t+1} = x_t + u_t + w_t$ is at least $(\mu_i/4)^2$ when u replenishes the inventory. When the demand variance is sufficiently large, there is no need for an exploration term and $\epsilon_i = 0$.

6.3.3 Basis Function Selection for the Inventory Control Problem

Two sets of 100 trajectories of 20 time steps with uniformly distributed initial inventory levels are sampled to serve as the training and validation samples, resulting in 2000 state-action pairs for each sample. The model is used again to sample the transition resulting from each pair 10 times to improve the accuracy of the next-state value-function estimate and immediate rewards estimates, as described by equation (2.4). Thus each sample is computed from 20,000 transitions.

The LSTD version of the algorithm is used to create basis functions and fit the value function starting with th initial value function V_0. The policy is updated on every tenth iteration, for a total of 60 iterations with 5 policy improvement steps. The total number of bases is limited to 100, and at most 25 new bases are added on each iteration. The pruning procedure of Algorithm 5 is used to prune redundant or excessive features at each step, with parameters $tol = 0.005$ and $N = 100$.

6.3.4 Experiments

A problem with 16 products where the first and last eight product demands have respective correlation matrices

$$\rho_A = \begin{bmatrix} 1 & 0.9571 & 0.9143 & 0.8714 & 0.8286 & 0.7857 & 0.7429 & 0.7 \\ 0.9571 & 1 & 0.9571 & 0.9143 & 0.8714 & 0.8286 & 0.7857 & 0.7429 \\ 0.9143 & 0.9571 & 1 & 0.9571 & 0.9143 & 0.8714 & 0.8286 & 0.7857 \\ 0.8714 & 0.9143 & 0.9571 & 1 & 0.9571 & 0.9143 & 0.8714 & 0.8286 \\ 0.8286 & 0.8714 & 0.9143 & 0.9571 & 1 & 0.9571 & 0.9143 & 0.8714 \\ 0.7857 & 0.8286 & 0.8714 & 0.9143 & 0.9571 & 1 & 0.9571 & 0.9143 \\ 0.7429 & 0.7857 & 0.8286 & 0.8714 & 0.9143 & 0.9571 & 1 & 0.9571 \\ 0.7 & 0.7429 & 0.7857 & 0.8286 & 0.8714 & 0.9143 & 0.9571 & 1 \end{bmatrix}$$

and

$$\rho_B = \begin{bmatrix} 1 & 0.9857 & 0.9714 & 0.9571 & 0.9429 & 0.9286 & 0.9143 & 0.9 \\ 0.9857 & 1 & 0.9857 & 0.9714 & 0.9571 & 0.9429 & 0.9286 & 0.9143 \\ 0.9714 & 0.9857 & 1 & 0.9857 & 0.9714 & 0.9571 & 0.9429 & 0.9286 \\ 0.9571 & 0.9714 & 0.9857 & 1 & 0.9857 & 0.9714 & 0.9571 & 0.9429 \\ 0.9429 & 0.9571 & 0.9714 & 0.9857 & 1 & 0.9857 & 0.9714 & 0.9571 \\ 0.9286 & 0.9429 & 0.9571 & 0.9714 & 0.9857 & 1 & 0.9857 & 0.9714 \\ 0.9143 & 0.9286 & 0.9429 & 0.9571 & 0.9714 & 0.9857 & 1 & 0.9857 \\ 0.9 & 0.9143 & 0.9286 & 0.9429 & 0.9571 & 0.9714 & 0.9857 & 1 \end{bmatrix},$$

while demands are uncorrelated between the two subsets, yielding

$$\rho = \begin{bmatrix} \rho_A & 0 \\ 0 & \rho_B \end{bmatrix}.$$

Two product groups are used, with any orders of products $\{1, 2, 3, 4, 9, 10, 11, 12\}$ incurring a fixed cost of 20, and orders of products $\{5, 6, 7, 8, 13, 14, 15, 16\}$ incurring a fixed cost of 40. Even numbered products have mean demand $\mu_i = 1$, while odd numbered products have mean demand $\mu_i = 2$. The demand standard deviation is set to $\sigma = 0.25$ for products $\{1, 2, 5, 6, 9, 10, 13, 14\}$ and to $\sigma = 0.5$ for products $\{3, 4, 7, 8, 11, 12, 15, 16\}$. The other parameters are set to $p_i = 10, c_i = 1, d_i = 2$ and $h_i = 0.2$ for all products, with $q = 10$ and $R = 160$. The discount factor is set to $\gamma = 0.95$.

The resulting Bellman error and discounted costs are shown in Figure 6–18. There is a significant decrease in the Bellman error over the early iterations as new basis functions are added, but little change after each of the policy iterations. Between policy iterations, there is a large change in Bellman error because a new sample is taken using the updated policy. There is a large increase in cost after the second iteration, but the average cost returns to the level of the initial value function estimate by the 11^{th} iteration. There is a small improvement in cost over the initial policy by iteration 40.

Figure 6–19 shows the result on a larger problem. There are 64 products where demand in four groups of 16 products have correlation greater than 0.8, 0.9, 0.95, and 0.95. Products in different groups have uncorrelated demand. Half of the products from each group are obtained from one supplier with a fixed cost of 40, and the other half share a fixed cost of 80. Penalties for an equal number of products in each group are set

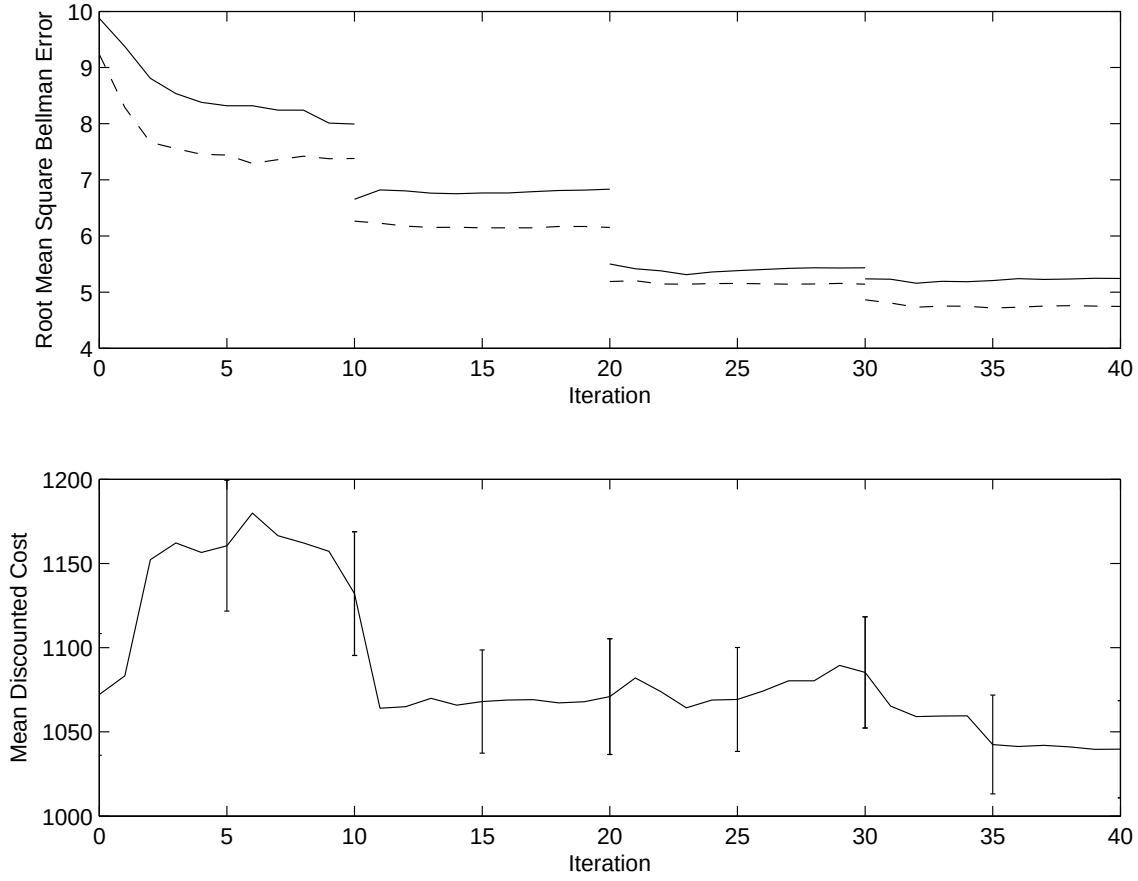

Figure 6–18: Bellman error and mean discounted cost as the algorithm is run on the 16-product Inventory Control problem. In the top panel, the solid line shows the root mean square Bellman error for the transitions in the validation sample, while the dashed line shows the Bellman error resulting from a Bellman error minimizing fit (rather than LSTD). The average shown in the bottom panel is computed for each iteration over 100 20-step trajectories with fixed but randomly chosen initial inventory levels. The error bars show one standard deviation (the variance is largely due to the effect of the initial inventory levels on the discounted cost, as evidenced by the smoothness of the curve). There is a small improvement in the mean discounted cost.

to each of $p = 10$ or $p = 20$. Demands have either mean $\mu_i = 1$ or $\mu_i = 2$, and standard deviation $\sigma_i = 0.5$ or $sigma_i = 0.25$, with each combination of parameters appearing twice in each group. The inventory capacity is set to $R = 525$.

The experiment is repeated using both the LS and LSTD variants of the algorithm, and solving the correction MDP or not. A policy improvement step is taken after every 30 iterations. A similar pattern as with the smaller problem is observed when using the Bellman residual minimizing fits, with the plain BEBFs offering the best Bellman error reduction except at the beginning. The is some reduction of cost after an initial increase, but the performance level of the initial value function is not achieved despite the reduction in Bellman error.

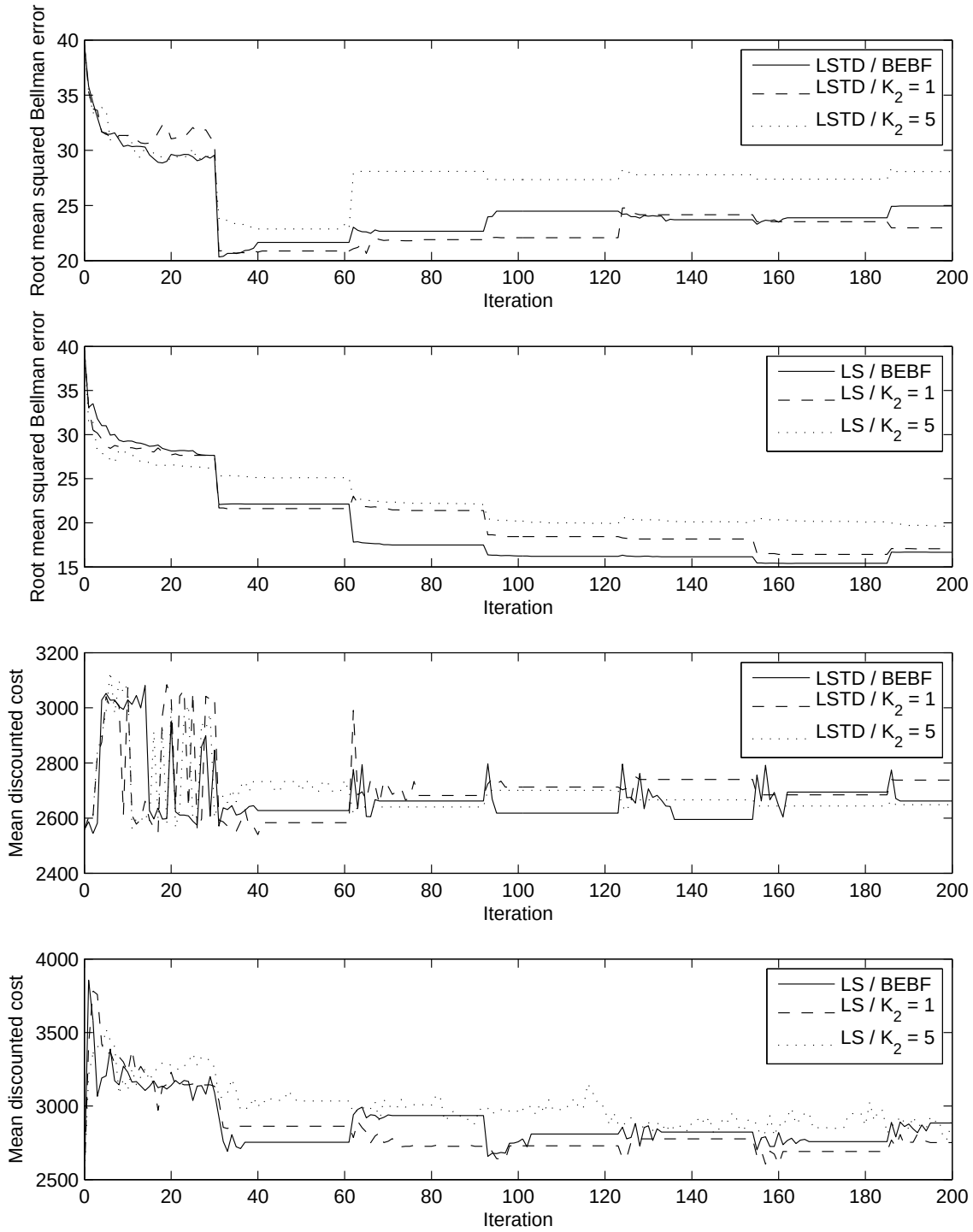

Figure 6–19: Bellman error and cost for the 64-product inventory control problem. A policy improvement step is taken on every 30^{th} iteration. With the initial sample (policy), there is a clear improvement of Bellman error with all the algorithms, but all yield a worse discounted cost. There is little change in the Bellman error or cost after each policy iteration. Solving the underlying MDP does not appear to yield better approximations, indeed it appears to be detrimental.

CHAPTER 7
Conclusions and Future Work

7.1 Contribution

The proposed algorithm was successfully applied to augmented versions of benchmark reinforcement learning problems, and shows some promise in solving the multiple product inventory control problem. It can be applied to problems with natural high-dimensional state representations, and can be used to improve an existing value function approximation, as shown with the inventory control problem.

7.2 Discussion

A major innovation in the current algorithm is the solution of the correction MDP in order to obtain more appropriate basis function. This was shown to have a significant effect in the mountain car domain in particular. This observation suggests that rather than simply creating approximate Bellman error basis functions as in [11, 16], the information contained in sampled transitions Bellman error should be exploited more fully. However, as evidenced by the three room problem and inventory control results, the approach taken here does not always outperform the algorithm which simply fits the Bellman error. Yet, the positive effect of solving the correction MDP is dramatic for the mountain car problem.

On the other hand, the proposed algorithm does not require as much pre-processing as the approximate transition model built to generate proto-value functions or the connectivity information gathered to create an explicit manifold representation. This may

prove particularly useful when available sample trajectories are limited, or simulation is costly.

The particular choice of dimentionality reduction and pruning algorithms used allows an efficient implementation, but other methods may be substituded to better suit a particular problem. Indeed, NCA performed well on the artificially augmented benchmark problems, even in the presence of significant noise, and succeeded in providing transformation suitable for reducing the Bellman error of the value function approximation in the inventory control problem. The method as implemented is efficient and can deal with high-dimensional inputs, as it was applied to 64-coordinate dense vector, and 300-coordinate sparse vector state representation.

The pruning algorithm exhibited stable performance in that it did not prune useful basis functions, as was observed with the direct application of the optimal brain surgeon procedure. Pruning is necessary to keep the set of features from growing too large.

The method of selecting basis function in the low dimensional space is a heuristic bases on minimizing the Bellman error on a validation set. Many other approaches are possible, but the use of radial basis functions provides a simple and suitably efficient class of approximators. Since the transformation from the state representation is linear, the radial basis functions added on a given iteration can be considered as RBFs in the original state space with a common rank-deficient width matrix. The choice of radial basis functions as features is common in reinforcement learning, and allows the basis functions generated on each iteration to be partially reused. This differs from proto-value functions or BEBFs, where a smaller number of global basis functions are generated.

The Bellman error norm is used as an evaluation criterion of the algorithm, a pruning criterion, and a feature selection criterion. There may be better objectives which may be used for each of these tasks. This is especially important when one considers the differences between LSTD and Bellman residual minimizing fits.

7.3 Future Directions

A possible next step in this line of research is to find bounds on the improvement due to new basis function, as in the work of Parr et al. [16], but when using more elaborate ways to generate basis functions than simply fitting the Bellman error. Such work would likely require an exploration of the relation between the LSTD and residual minimizing solutions to the value function fit.

The application of the algorithm to realistic large-scale problems is nontrivial, as evidenced by the inventory control results. It may be possible to taylor the various aspects of the algorithm in order to solve a particular class of problems.

References

[1] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control, Two Volume Set*. Athena Scientific, 2001.

[2] D.P Bertsekas and D.A. Castanon. Adaptive aggregation methods for infinite horizon dynamic programming. *IEEE Transactions on Automatic Control*, 34:589–598, 1989.

[3] Justin A. Boyan. Technical update: Least-squares temporal difference learning. *Machine Learning*, 49(2-3):233–246, 2002.

[4] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, March 2004.

[5] Steven J. Bradtke and Andrew G. Barto. Linear least-squares algorithms for temporal difference learning. *Machine Learning*, 22(1-3):33–57, 1996.

[6] Federgruen, A. and Zipkin, P. Computing optimal (s, s) policies in inventory models with continuous demands. *Advances in Applied Probability*, 17(2):424–442, jun 1985.

[7] Jacob Goldberger, Sam T. Roweis, Geoffrey E. Hinton, and Ruslan Salakhutdinov. Neighbourhood components analysis. In *NIPS*, 2004.

[8] Gene H. Golub and Charles F. Van Loan. *Matrix Computation*. John Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, Baltimore, Maryland, third edition, 1996.

[9] Babak Hassibi and David G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. In Stephen José Hanson, Jack D. Cowan, and C. Lee Giles, editors, *Advances in Neural Information Processing Systems*, volume 5, pages 164–171. Morgan Kaufmann, San Mateo, CA, 1993.

[10] Leemon C. Baird III. Residual algorithms: Reinforcement learning with function approximation. In *International Conference on Machine Learning*, pages 30–37, 1995.

[11] Philipp W. Keller, Shie Mannor, and Doina Precup. Automatic basis function construction for approximate dynamic programming and reinforcement learning. In William W. Cohen and Andrew Moore, editors, *ICML*, pages 449–456. ACM, 2006.

[12] Michail G. Lagoudakis and Ronald Parr. Least-squares policy iteration. *J. Mach. Learn. Res.*, 4:1107–1149, 2003.

[13] S. Mahadevan and M. Maggioni. Proto-value functions: A laplacian framework for learning representation and control. *Journ. Mach. Learn. Res.*, September 2007.

[14] Ishai Menache, Shie Mannor, and Nahum Shimkin. Basis function adaptation in temporal difference reinforcement learning. *Annals of Operations Research*, 134:215–238(24), February 2005.

[15] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer Series in Operations Research. Springer, New York, 1999.

[16] Ronald Parr, Christopher Painter-Wakefield, Lihong Li, and Michael Littman. Analyzing feature generation for value-function approximation. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 737–744, New York, NY, USA, 2007. ACM.

[17] Bohdana Ratitch and Doina Precup. Sparse distributed memories for on-line value-based reinforcement learning. In Jean-François Boulicaut, Floriana Esposito, Fosca Giannotti, and Dino Pedreschi, editors, *ECML*, volume 3201 of *Lecture Notes in Computer Science*, pages 347–358. Springer, 2004.

[18] Andrew Moore Remi Munos. Variable resolution discretization in optimal control. *Machine Learning*, 49, Numbers 2/3:291–323, November/December 2002.

[19] Martin Riedmiller, Sascha Lange, Stephan Timmer, and Roland Hafner. Clsquare: Closed loop simulation system, September 2005.

[20] Ralf Schoknecht. Optimality of reinforcement learning algorithms with linear function approximation. In S. Thrun S. Becker and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, pages 1555–1562. MIT Press, Cambridge, MA, 2003.

[21] Satinder P. Singh, Tommi Jaakkola, and Michael I. Jordan. Reinforcement learning with soft state aggregation. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 361–368. The MIT Press, 1995.

[22] William D. Smart. Explicit manifold representations for value-functions in reinforcement learning. In *Proceedings of the Eighth International Symposium on Artificial Intelligence and Mathematics*, January 2004. Paper number AI&M 25-2004.

[23] Nathan Sprague. Basis iteration for reward based dimensionality reduction. *Development and Learning, 2007. ICDL 2007. IEEE 6th International Conference on*, pages 187–192, 11-13 July 2007.

[24] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction*. MIT Press, 1998.

[25] Vladislav Tadic. On the convergence of temporal-difference learning with linear function approximation. *Mach. Learn.*, 42(3):241–267, 2001.

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

[27] Zheng, Yu-Sheng and Federgruen, A. Finding optimal (s, S) policies is about as simple as evaluating a single policy. *Operations Research*, 39(4):654–665, July 1991.

[28] O. Ziv and N. Shimkin. Multigrid algorithms for temporal difference reinforcement learning. In *Proceedings of the ICML workshop on rich representations for RL*, 2005.