Basis Iteration for Reward Based Dimensionality Reduction

Nathan Sprague
Kalamazoo College
1200 Academy St.
Kalamazoo, MI 49006

Abstract— **We propose a linear dimensionality reduction algorithm that selectively preserves task relevant state data for control problems modeled as Markov decision processes. The algorithm works by alternating value function estimation with basis vector adaptation. The approach is demonstrated on two tasks: a toy task designed to illustrate the key concepts, and a more complex three dimensional navigation task.**

Index Terms— **Reinforcement Learning, Dimensionality Reduction, Perceptual Development**

I. Introduction

There has been much research that attempts to explain the structure of biological receptive fields in terms of adapting basis vectors to capture the statistical structure of sensory input. These include principal components analysis [1], independent components analysis [2], non-negative matrix factorization [3], and predictive coding [4], among others. Typically, such approaches are based purely on the structure of the input; there is no consideration of the role that sensory information plays in the goal directed behavior of an organism. The motivation for the current work is to explore mechanisms of basis vector adaptation that are explicitly driven by the behavioral demands of a situated agent. The underlying hypothesis is that perceptual systems should be developmentally tuned to extract information that is relevant for survival, and to discard information that is not.

We approach this issue within a reinforcement learning (RL) framework. The goal is to discover basis vectors that preferentially extract information that can be used to predict and obtain future reward. Progress has been slow in developing RL algorithms that scale reliably from small discrete state spaces to high dimensional continuous spaces such as those that arise when dealing directly with real sensory modalities such as vision. One problem has been that scaling up traditional reinforcement learning algorithms, such as Q-Learning, requires replacing table based representations of the value function with function approximators. Unfortunately, the resulting algorithms lack convergence guarantees and often perform poorly in practice. One algorithm that addresses these issues is least squares policy iteration (LSPI) [5]. LSPI couples a linear approximation architecture with approximate policy iteration. The result is an algorithm that is guaranteed not to diverge, and performs well in practice.

Of course, LSPI is not a silver bullet for the curse of dimensionality. It can only be effective when the intrinsic dimensionality of the state information necessary for predicting reward is relatively small, and when basis vectors are provided that extract the appropriate low dimensional subspace. The goal of this work is to automate the discovery of appropriate basis vectors.

In the remainder of the paper, we briefly introduce the Markov decision process framework as well as the two algorithms that form the foundation for our approach: least squares policy iteration and neighborhood components analysis. We then describe the basis iteration algorithm in detail. The algorithm is demonstrated on two tasks, first a simple artificial task, then a more complex task involving navigation in a simulated 3d environment.

II. Algorithm

We assume that the control problems of interest can be described as Markov decision processes (MDP). An MDP is specified as a 5-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$ where $\mathcal{S}$ is the space of possible states; $\mathcal{A}$ is the space of possible actions; $\mathcal{P}$ is a transition function where $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ represents a probability distribution over state transitions; $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is a reward function indicating expected immediate reward; and $\gamma \in [0, 1)$ is a discount value applied to future rewards. A policy is specified by $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ indicating the probability of selecting each action in each state.

In reinforcement learning problems it is assumed that neither $\mathcal{R}$ or $\mathcal{P}$ are known in advance, and the goal is to find the policy π^* that results in the maximum expected discounted return. In finding an optimal policy it is helpful to define a state action value function $Q^\pi(s, a)$ that maps state action pairings to the expected discounted return that will be received if the agent starts in state s, takes action a, and acts according to π thereafter:

$$Q^\pi(s, a) = E_\pi \left\{ \sum_{t=0}^{t=\infty} \gamma^t r_t \right\}$$

The optimal value function $Q^*(s, a)$ represents the state action value function under the optimal policy. If this function is known, then the deterministic optimal policy can be specified as:

$$\pi(s) = \operatorname*{argmax}_a Q^*(s, a).$$

A. LSPI

LSPI represents an approximation of the value function as a linear combination of k basis functions $\phi_i(s,a)$:

$$\hat{Q}(s,a;w) = \sum_{i=1}^{k} \phi_i(s,a) w_i = \phi(s,a)^\top w$$

For the discrete state case, this can represent a large space savings over a tabular representation because, in general, k is chosen so that $k << |\mathcal{S}||\mathcal{A}|$. This approach is also attractive because it allows continuous state problems to be handled within the same framework. The objective of the LSPI algorithm is to find a setting of the w vector that results in a good approximation of the true optimal value function $Q^*(s,a)$.

The mechanism for adapting the vector w is based on approximate policy iteration. The algorithm proceeds by first generating an arbitrary policy, specified by w, and then iterating through the following steps until convergence:

1) Compute an estimate $\hat{Q}^\pi$ of the value function associated with the current policy.
2) Improve the policy according to $\pi(s) = \text{argmax}_a \hat{Q}(s,a;w)$.

In the tabular setting, if $\mathcal{P}$ and $\mathcal{R}$ are known, step 1) can be accomplished exactly by solving for the fixed point of the Bellman operator T_π:

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

We face a harder problem here. The result of applying the Bellman operator to $\hat{Q}^\pi$ is likely to be outside of the space of value functions that can be represented by the approximation architecture. The LSPI algorithm addresses this issue by finding a value function $\hat{Q}^\pi$ that is invariant after an application of the Bellman operator followed by a projection to the space spanned by the basis functions. Since $\mathcal{P}$ and $\mathcal{R}$ are not known, that approximate value function is learned from a finite set of samples:

$$D = \{(s_i, a_i, r_i, s'_i) \mid i = 1, 2, ..., L\}$$

where one sample indicates that action a_i was taken in state s_i, reward r_i was received, and the resulting state was s'_i. Refer to [5] for a complete description of the algorithm.

For this work, we are assuming a discrete set of actions and that states are represented by n dimensional real valued vectors: $\mathcal{S} \subset \mathbb{R}^n$. The basis functions are the composition of two steps. First, state vectors are linearly projected from the state space to a d dimensional space:

$$A : \mathbb{R}^n \rightarrow \mathbb{R}^d$$

where $A \in \mathbb{R}^{d \times n}$ and $d < n$. Next, radial basis functions are applied in the lower dimensional space.

Since it is unlikely that the true optimal value function can be represented exactly in the space spanned by the basis functions, the estimated optimal value function $\hat{Q}^*$ discovered by LSPI will generally not be identical to the true optimal value function Q^*. One key to finding good policies using LSPI is selecting basis functions that make this error as small as possible. The focus in this paper is on adapting A; the radial basis functions are constructed by hand, and remain fixed throughout training.

B. NCA

The basic outline of the algorithm we are proposing is to start out with an arbitrary A matrix, apply LSPI to find an estimate of the optimal value function, and then update A based on that estimate. This process is repeated until convergence, or for some fixed number of iterations. The problem of adapting the A matrix is addressed using neighborhood components analysis (NCA) [6].

Neighborhood components analysis is a probabilistic generalization of K-Nearest neighbor classifications that learns a linear transformation of the input space that minimizes leave-one-out cross validation error in the projected space. Specifically, NCA finds a metric $C = A^\top A$ such that $d(x,y) = (x-y)^\top C(x-y) = (Ax - Ay)^\top (Ax - Ay)$. If $x, y \in \mathbb{R}^n$ then A is restricted to be in $\mathbb{R}^{d \times n}$. We will consider the case where $d < n$; the projected space has lower dimensionality than the input space.

Under the NCA model, prediction is performed by choosing a neighbor according to a distance-based probability distribution:

$$p_{ij} = \frac{\exp(-\,||Ax_i - Ax_j||^2)}{\sum_{k \neq i} \exp(-\,||Ax_i - Ax_k||^2)}, \quad p_{ii} = 0 \qquad (1)$$

where p_{ij} indicates the probability of selecting point j to predict the class of point i.

The authors of [7] have adapted the NCA algorithm to regression problems. In the regression formulation, the expected squared error for point i can be expressed as:

$$\delta_i = \sum_j p_{ij}(y_j - y_i)^2 \qquad (2)$$

Where y_i indicates the target value associated with point x_i. The algorithm proceeds by minimizing the expected sum squared error across the entire data set:

$$f(A) = \sum_i \delta_i = \sum_i \sum_j p_{ij}(y_j - y_i)^2 \qquad (3)$$

This is minimized by differentiating with respect to A and using gradient descent.

As observed in [7] the resulting algorithm frequently does not converge. This can be understood by observing that, after any projection, points that are near the query point are likely to have values closer to the target value than those that are farther away. As a result, the elements of A tend to grow without bound, increasing the probability that points will be assigned to their single nearest neighbor. This is dealt with in [7] by adding the squared Frobenius norm of A to $f(A)$ as a regularization term, resulting in a new objective: $g(A) = f(A) + N\rho\,||A||_{\mathrm{F}}$, where N is the number of data points and ρ is a small constant. It has been our experience that it is difficult to find a value of ρ that consistently leads to good performance. As an alternative we propose the following formulation of the probability where σ is a width parameter:

$$p_{ij} = \frac{\exp\left(-\left(\frac{||Ax_i - Ax_j||^2}{||A||_{\mathrm{F}}}\right)/\sigma\right)}{\sum_{k\neq i}\exp\left(-\left(\frac{||Ax_i - Ax_k||^2}{||A||_{\mathrm{F}}}\right)/\sigma\right)}, \quad p_{ii} = 0 \tag{4}$$

Dividing by $||A||_{\mathrm{F}}$ in Equation (4) has the effect of making p_{ij} invariant to changes in the scale of A. Selecting a value for σ amounts to specifying a particular scale in advance. This has the dual advantages of leading to an algorithm that is numerically stable, and reducing the effective search space of the problem.

Substituting (4) into (3) and differentiating with respect to A results in the following gradient (where $x_{ij} = x_i - x_j$):

$$\frac{\partial f}{\partial A} = \frac{2A}{||A||_{\mathrm{F}}\sigma}\sum_{i,j}(\delta_i - y_{ij}^2)\left(p_{ij}\left(x_{ij}x_{ij}^{\top} - I\frac{||Ax_i - Ax_j||^2}{||A||_{\mathrm{F}}}\right)\right) \tag{5}$$

In our implementation, we minimize (3) through the method of conjugate gradients. In order to more efficiently evaluate the gradient, we truncate the sums in (5) as suggested in [6]; the inner sums are evaluated in descending order of probability and the sum is truncated when 99.9% of the probability mass is accounted for.

When we apply NCA to the problem of finding appropriate projections for value function estimation, it will be useful to consider the case of multiple simultaneous regression problems that each use the same projection matrix. In particular, value function estimation will be formulated as $|\mathcal{A}|$ separate regression problems, one for each possible action. It is straightforward to adapt the NCA algorithm to this scenario; the optimization objective in Equation (3) is modified to be a sum of objectives of the same form, and the gradient in Equation (5) is modified to be the sum of the corresponding gradients. The only restriction is that the dimensionality of each of the regression problems must be the same.

Note that NCA is only being used to find appropriate state projections. We will not be performing value function estimation using the the nearest neighbor approach.

C. Basis Iteration

We are now ready to describe the proposed algorithm. The premise is that, although the dimensionality of the state vectors is high, there may be a low dimensional linear subspace that contains most, or all, of the information necessary to represent the optimal value function. If the optimal value function were known, it would be straightforward to apply NCA to find this subspace: we could partition our set of samples D into $|\mathcal{A}|$ subsets $D_{a\in\mathcal{A}}$ according to which action was taken in each sample. Each of these subsets would have its own set of input/target value pairings of the form $\langle s, Q(s,a)\rangle$, and the multiple regression formulation of NCA could be applied to the resulting $|\mathcal{A}|$ regression problems. In principle, the projection matrix discovered by NCA would selectively preserve a linear subspace containing information useful for predicting the optimal value function. Of course, the true optimal value function is not available for use as a target, so an alternative is necessary.

The solution we propose is to alternate value function estimation with basis function adaptation. The process is outlined in Algorithm 1. An initial A matrix may be either generated randomly or initialized using some other algorithm such as Principal Components Analysis. The first step, labeled REDUCE in Algorithm 1 is to use A to project the state vectors to the lower dimensional space and rescale the resulting vectors so that they are contained in the unit hypercube. The rescaling is performed so that the width and spacing of the radial basis functions can remained fixed. Next, the LSPI algorithm is run on the rescaled samples. For the examples presented in this paper the actual basis functions used by the LSPI algorithm are radial basis functions defined in the d dimensional projected space. However, other choices for the basis functions are possible.

At this point we have an estimate of the optimal value function, but unless we were very lucky in selecting our initial A matrix, the estimate is probably not very good. We would like to use NCA to update A, but it is not immediately clear what the target values should be. Consider the Bellman optimality operator T_*:

$$(T_*Q)(s,a) = \mathcal{R}(s,a) + \gamma\sum_{s'\in\mathcal{S}}\mathcal{P}(s,a,s')\max_{a'\in\mathcal{A}}Q(s',a') \tag{6}$$

The optimal value function is the fixed point of this operator. A good choice of A would have the property that, for all states and actions, the resulting estimated value function $\hat{Q}(s,a)$ would be equal to the right hand side of Equation (6). Therefore, we will use an estimate of the right hand side of Equation (6) as the target for the NCA algorithm. For each sample (s_i, a_i, r_i, s_i') in our training set, we calculate the following target:

$$t_i = r_i + \gamma\max_{a'\in\mathcal{A}}\hat{Q}(s_i', a'; w) \tag{7}$$

The resulting training points of the form $\langle s_i, t_i\rangle$ are partitioned into $|\mathcal{A}|$ disjoint sets $\{\langle s_i, t_i\rangle \mid a_i = a\}_{a\in\mathcal{A}}$ and the multiple-objective formulation of the NCA algorithm is applied to finding an appropriate projection, using the previous A matrix as a starting point. The original set of samples is then re-projected using the new A matrix to the lower dimensional space, and the process is repeated for the desired number of iterations, or until there is no further change in A.

Algorithm 1 Basis Iteration

```
// INPUTS:
// $D$ - Samples of the form $(s, a, r, s')$
// $A$ - Initial $d \times n$ projection matrix.
// $\phi$ - Basis functions in $d$-dimensional space.
// $K$ - Desired number of iterations.

$X \leftarrow \mathrm{REDUCE}(D, A)$ //Project and scale state vectors.
for $k = 1, 2, ..., K$ do
    $w \leftarrow \mathrm{LSPI}(X, \phi)$
    //Calculate Bellman target values for all samples:
    $t_i = r_i + \gamma \max_{a' \in \mathcal{A}} \hat{Q}(s'_i, a'; w)$
    $A \leftarrow \mathrm{NCA}(\{\langle s_i, t_i \rangle \mid a_i = a\}_{a \in \mathcal{A}}, A)$
    $X \leftarrow \mathrm{REDUCE}(D, A)$
end for
return $A$, $w$
```

III. Example: Puck Centering

First we present a toy example designed to highlight the main features of the algorithm. The goal of this task is for the agent to move a simulated puck to the center of a linear region. The agent has a choice of three actions: $\mathcal{A} = \{-.2, 0, .2\}$. A reward of 1.0 is received on every time step that the position of the puck is between .45 and .55, and a reward of 0 is received otherwise. If the position of puck ever moves outside of the range [0,1], the trial ends. The puck is randomly positioned in the range [0,1] at the start of every trial.

The first dimension of the task's state space, p, represents the current position of the puck. The second dimension, j, represents a multiplier that is applied to the action value in determining the next position. The remaining four dimensions contain noise. State vectors have the form $s = [p, j, z_1, z_2, z_3, z_4]^\top$. The position variable is updated according to the following equation:

$$p_{t+1} = p_t + a_t j_t + v$$

where $v \sim \mathcal{N}(0, .0025)$. On each time step, the value of j, as well as the values for the noise dimensions, are drawn from a uniform distribution in the range [-.5, .5].

In order to demonstrate the proposed algorithm it will be used to find a two dimensional projection of the state vector that allows a good policy to be learned. Given that the state vector only contains two useful dimensions, such a projection should exist. In fact, this is a case where the "correct" projection is known; a 2×6 matrix with ones along the diagonal, and zeros for all other entries will retain the relevant state dimensions, while discarding the noise dimensions.

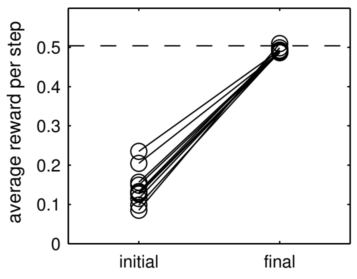

Fig. 1. Average reward in puck centering task before and after basis function adaptation. Data points labeled "initial" represent the average reward obtained by policies discovered using LSPI and a random dimensionality reduction matrix. Points labeled "final" represent the average reward obtained by policies discovered using LSPI and the dimensionality reduction matrix found by the basis iteration algorithm. The dotted line represents the amount of reward obtained by the policy trained using the "correct" projection matrix: ones along the diagonal and zeros for all other entries. All points represent 1000 trials with a maximum of 100 steps per trial. Bars representing 95% confidence intervals would be contained inside the plotted circles.

The interesting thing about this task is that the j dimension is, in most respects, indistinguishable from the dimensions that contain pure noise. There is no difference in their distributions, and neither j nor the noise dimensions carry any information that can be used to predict immediate reward.

The basis functions used in the two dimensional space are adapted from those used for the pendulum task in [5]. There are 10 basis functions for each of the three possible actions, for a total of 30. The basis functions for each action include one constant and nine radial basis functions equally spaced in a 3×3 grid. For a particular state and action value all basis functions are zero except for the 10 corresponding to the current action, which have the form:

$$\left(1, e^{-\frac{\|s-\mu_1\|^2}{2\sigma^2}}, e^{-\frac{\|s-\mu_2\|^2}{2\sigma^2}}, \cdots, e^{-\frac{\|s-\mu_9\|^2}{2\sigma^2}}\right)^\top$$

where the μ_i's are the nine points $\{.25, .5, .75\} \times \{.25, .5, .75\}$, and $\sigma^2 = .5$.

For all experiments a discount factor of .9 is used. The width parameter for the NCA algorithm is set to .01.

Tests were conducted by initially gathering data during 200 trials, each consisting of a maximum of 20 steps, resulting in a total of 2488 samples. During these initial trials actions were chosen at random. This same set of training samples was used for all of the results reported.

We performed ten test runs of the basis iteration algorithm, each starting with a different randomly generated 2×6 projection matrix with elements drawn from a uniform distribution in the range [-.5, .5]. Figure 1 illustrates the results. For all ten randomly generated matrices the algorithm is able to find a projection that results in a significantly better policy than is possible with the initial projection. Algorithm 1 converged within six iterations in all cases. Note that the matrices

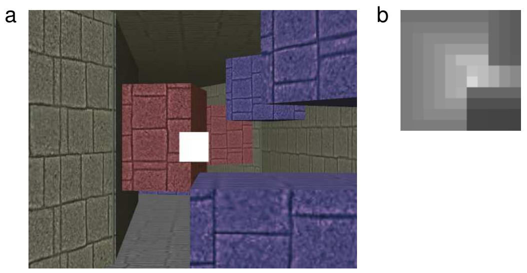

Fig. 2. The Navigation Task. a) A view of the environment from behind the agent (white square). b) Depth information from the agent's point of view. The two dark regions on the right represent looming obstacles.

discovered by the basis iteration algorithm generally do not have ones on the diagonal and zeros for the other entries; there are many projections of the first two dimensions that allow a good policy to be learned. What all of the discovered projections have in common is that they suppress the four noise dimensions.

IV. Example: Navigation

The algorithm has also been tested on a three dimensional collision avoidance task in which an agent uses depth information to fly through a twisting corridor containing randomly positioned obstacles (Figure 2a). On each time step the agent chooses between three actions: $\{\text{LEFT}, \text{FORWARD}, \text{RIGHT}\}$. LEFT and RIGHT correspond to a 15° turn in the appropriate direction. All three actions cause the agent to move forward .4 meters.

The agent receives a reward of 1.0 on every time step that he successfully avoids colliding with obstacles or the walls, and 0.0 otherwise. If the agent ever becomes stuck in the same spot for more than two time steps in a row, the trial is terminated. State information takes the form of an 11×11 depth map of the area ahead, where each pixel corresponds to the log of the distance to the nearest obstacle in the corresponding direction (Figure 2b). The agent has a $90^\circ \times 90^\circ$ field of view.

In order to highlight the fact that the algorithm is able to adapt basis vectors to task demands, two variations of the navigation task are explored. These variations involve vertically offsetting the agent's point of collision either one meter above or one meter below his point of view. This manipulation does not change the distribution of state information that the agent observes, but it does change which aspects of the state information are most task relevant. In the original formulation, the most important state data is in the central portion of the depth field; depth information for the top and bottom of the field are not as relevant, because the agent is incapable of colliding with obstacles in those regions. Moving the point of collision up or down results in a corresponding change in the relevant portion of the depth field.

As in the puck centering task, the first step is to gather a set of training samples under a completely random policy. Samples were gathered for each of the three conditions during 200 trials with a maximum of 100 steps per trial. The total number of samples for the 0m offset, +1m offset, and -1m offset tasks were 5515, 6165 and 5671 respectively.

The goal is to reduce the 122 dimensional state space down to two dimensions that are appropriate for finding a good policy. In the interest of computational efficiency, the state data is first reduced to 15 dimensions using principal components analysis. The basis iteration algorithm is applied in this 15 dimensional space. Rather than starting with a random set of basis vectors, we initialize Algorithm 1 using the first two principal components of the state vectors from the training sets.

There are 75 radial basis functions in the projected two dimensional space: 25 for each of the three possible actions. The 25 basis functions associated with each action have the form:

$$\left(e^{-\frac{\|s-\mu_1\|^2}{2\sigma^2}}, e^{-\frac{\|s-\mu_2\|^2}{2\sigma^2}}, \cdots, e^{-\frac{\|s-\mu_{25}\|^2}{2\sigma^2}}\right)^\top$$

where the μ_i's are the 25 points $\{\frac{1}{6}, \frac{1}{3}, \frac{1}{2}, \frac{2}{3}, \frac{5}{6}\} \times \{\frac{1}{6}, \frac{1}{3}, \frac{1}{2}, \frac{2}{3}, \frac{5}{6}\}$, and $\sigma^2 = \sqrt{.03}$.

For all experiments a discount factor of .9 is used. The width parameter for the NCA algorithm is set to .01.

For all three training sets, Algorithm 1 converged in fewer than 15 iterations. Figure 3 illustrates the initial basis vectors discovered using principal components analysis as well as the final basis vectors discovered using the basis iteration algorithm. Figure 3b illustrates the basis vectors discovered for the original version of the task. Relative to the principal components, these vectors are mostly uniform along the top and bottom of the depth field: regions of the environment that do not contain useful information for collision avoidance. For the other two variations of the task, the structured part of the basis vector moves to the part of the depth field that could contain potential colliders (Figure 3c-d).

Figure 4 shows more explicitly what information is being preserved by the basis vectors displayed in Figure 3. This figure was generated by orthonormalizing the projection matrices and using them to reconstruct the depth data from the training sets. What we can observe here is that the principal components result in the best overall reconstruction, while the adapted basis functions preferentially preserve regions of the depth images that are task relevant.

Figure 5 illustrates that, despite the fact that the principal components do a better job of capturing the variance in the state data, the adapted basis vectors result in superior policies.

V. Related Work

Our approach to basis function adaptation is built on a body of recent work that attempts to automatically construct

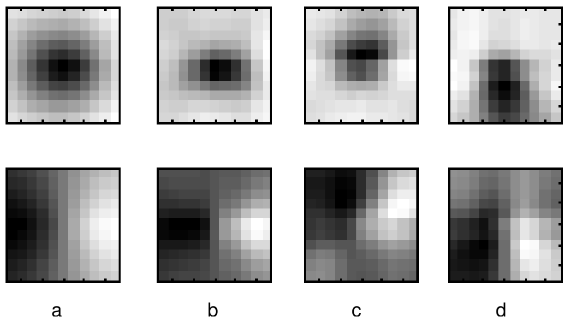

Fig. 3. Basis Vectors. a) The first two principle components of the depth data. b) The basis vectors discovered by the basis iteration algorithm. c,d) The basis vectors discovered by basis iteration when the agent's point of collision is one meter above (c) or below (d) the center of his perceptual field.

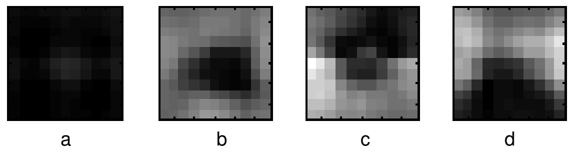

Fig. 4. Per-Pixel Reconstruction Error. For this figure, the corresponding basis vectors from Figure 3 are orthonormalized and used to reconstruct the depth data from the training sets. Darker regions indicate lower mean squared reconstruction error.

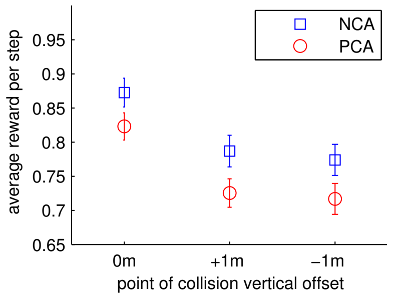

Fig. 5. Average Reward. Average reward across 300 trials of 100 steps each. The horizontal axis represents the three different experimental conditions described in Figure 3. In each condition the average per-step reward is reported for policies learned using PCA basis vectors as well as policies learned using learned basis vectors. One unit of reward is received for every step that does not result in collision. Therefore the y axis can be read as the percentage of steps that did not result in collision. Bars represent 95% confidence intervals.

appropriate basis functions for reinforcement learning in continuous or high dimensional spaces e.g. [7], [8], [9]. In particular, the work presented here is strongly influenced by the algorithm presented in [7], which also uses NCA to discover basis functions. The main differences are that rather than iteratively improving a fixed number of basis vectors, the algorithm in [7] adds additional basis vectors with every iteration. They also use a different formulation of the target value for the NCA algorithm, and focus only on policy evaluation, not on discovering an optimal policy.

Another area of related work is the action respecting embedding algorithm described in [10]. That algorithm attempts to reduce the dimensionality of state data in such a way that actions represent simple transformations in the low dimensional space. Such an approach is complementary to our work, in that it also results in a state representation that facilitates control, but it does not directly incorporate information about task goals.

VI. Discussion and Future Work

We have demonstrated an algorithm for basis function adaptation that successfully finds task relevant linear subspaces of high dimensional state data. There is nothing about the basic framework of the algorithm, value function estimation followed by basis function adaptation, that depends on the specific details of LSPI and NCA. There may be alternatives for one or both of those algorithms that would result in a more effective algorithm.

One possible weakness of the proposed algorithm is that the two stages are built on top of two entirely different frameworks for function approximation. The value function estimation is performed using weighted basis vectors, while the neighborhood components analysis algorithm is built on top of a nearest neighbor based approach. The assumption is that a projection that works well for the second is likely to work for the first. We are interested in finding a way to handle both stages of the algorithm within a unified function approximation framework.

References

[1] Peter J.B. Hancock, Roland J. Baddeley, and Leslie S. Smith. The principal components of natural images. *Network*, 3:61–70, 1992.
[2] A.J. Bell and T.J. Sejnowski. The "independent components" of natural scenes are edge filters. *Vision Research*, 37(23):3327–38, 1997.
[3] D.D. Lee and H.S. Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788–91, 1999.
[4] Rajesh P. N. Rao and Dana H. Ballard. Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects. *Nature Neuroscience*, 2(1):79 – 87, 1999.
[5] Michail G. Lagoudakis and Ronald Parr. Least-squares policy iteration. *Journal of Machine Learning Research*, 4:1107–1149, 2003.
[6] Jacob Goldberger, Sam Roweis, Geoff Hinton, and Ruslan Salakhutdinov. Neighbourhood component analysis. In *Neural Information Processing Systems*, 2004.
[7] Philipp W. Keller, Shie Mannor, and Doina Precup. Automatic basis function construction for approximate dynamic programming and reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, pages 449 – 456, Pittsburgh, PA, 2006.
[8] S. Mahadevan. Samuel meets amarel: Automating value function approximation using global state space analysis. In *Proceedings of the National Conference on Artificial Intelligence AAAI05*, Pittsburgh, PA, July 2005.
[9] W.D. Smart. Explicit manifold representations for value-function approximation in reinforcement learning. In *Proceedings of the 8th International Symposium on AI and Mathematics*, 2004.
[10] Michael Bowling, Ali Ghodsi, and Dana Wilkinson. Action respecting embedding. In *Proceedings of the 22nd International Conference on Machine Learning*, Bonn, Germany, 2005.