Predictive Projections

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

Abstract

This paper addresses the problem of learning control policies in very high dimensional state spaces. We propose a linear dimensionality reduction algorithm that discovers predictive projections: projections in which accurate predictions of future states can be made using simple nearest neighbor style learning. The goal of this work is to extend the reach of existing reinforcement learning algorithms to domains where they would otherwise be inapplicable without extensive engineering of features. The approach is demonstrated on a synthetic pendulum balancing domain, as well as on a robot domain requiring visually guided control.

1 Introduction

Scaling reinforcement learning algorithms to handle problems with high dimensional state spaces has been a longstanding challenge. The problem is particularly acute for the case of visually guided control, where the raw sensor data may easily have thousands of dimensions. In such cases, a typical approach would be to provide the reinforcement learning algorithm with a front-end that converts raw sensor data to a low-dimensional state representation that is suitable for learning. The problem with this approach is that developing the visual front-end may be more difficult than solving the learning problem itself. Worse, the process generally needs to be repeated for each new learning task.

The goal of the work presented here is to develop a linear dimensionality reduction algorithm that greatly simplifies the process of finding state representations for control problems in high dimensional continuous spaces. An appropriate state representation for reinforcement learning should have at least two properties: First, the dimensionality of the space should not be too high. Otherwise the algorithm will be undone by the curse of dimensionality. Second, when the same action is taken in similar states, the outcome should be similar. This makes it possible to safely generalize from observed to unobserved states.

The proposed algorithm, predictive projections (PP), addresses both of these considerations. It builds on a body of recent work in the area of distance metric learning (e.g. [Goldberger *et al.*, 2004; Weinberger and Tesauro, 2007]) in which nearest neighbor learning is recast in a probabilistic framework that allows for gradient based optimization of the distance metric. These existing algorithms discover projections of the training data under which nearby points are likely to have the same class label or similar regression targets. The algorithm described in this paper makes use of the same machinery but attempts to find low-dimensional projections under which current state vectors accurately predict future states in the projected space. The intuition is that projections which capture the state dynamics in this way are likely to contain information that will be useful for control.

The remainder of this paper will introduce the predictive projections algorithm and demonstrate its use on two problems. The first is a simulated pendulum balancing task, and the second is a visual control problem on a real robot. In both cases the algorithm is able to discover projections that enable successful learning of the target task.

1.1 Metric Learning

The work presented here has its roots in the Neighborhood Components Analysis (NCA) algorithm [Goldberger *et al.*, 2004]. The goal of the NCA algorithm is to discover a distance metric that minimizes error for nearest neighbor classification. The algorithm begins with a set of input vectors $x_1, ..., x_n$ in $\mathbb{R}^D$, along with their class labels, $c_1, ..., c_n$. It then searches for a linear transformation of the input space that minimizes leave-one-out classification error in the transformed space. This linear transformation, defined by the matrix A, implicitly defines a Mahalanobis distance metric in the original space, $d(x_i, x_j) = (Ax_i - Ax_j)^\top (Ax_i - Ax_j) = (x_i - x_j)^\top C(x_i - x_j)$, where $C = A^\top A$ is a symmetric positive semi-definite matrix. Since x_i and x_j are D-dimensional vectors, A is restricted to be in $\mathbb{R}^{d \times D}$. In this paper we will be interested in the case where $d < D$; the transformed space has lower dimensionality than the input space.

It would be difficult to directly find an A that minimizes the k-nearest-neighbor classification error rate, because the number of errors will be a highly discontinuous function of A; very small changes in A may change the set of nearest neighbors for some points. The innovation behind the NCA algorithm is to recast nearest neighbor learning in a probabilistic framework. In this framework, expected error is a continuous, differentiable function of A and thus may be minimized using gradient based techniques.

Under NCA, prediction is performed by choosing a single 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.

While NCA was originally developed in the context of classification, several authors [Keller *et al.*, 2006; Weinberger and Tesauro, 2007; Sprague, 2007] have explored extensions of the NCA framework to regression problems. In the classification formulation, the goal is to minimize the expected number of misclassified points. In the regression formulation, the goal is to minimize the expected squared prediction error. The expected squared error for point i can be expressed as:

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

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

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

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

2 Predictive Projections

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.

The first step in applying the predictive projections algorithm is to collect a set of samples of the form (s, a, r, s') where s is a state vector, a is the action chosen in state s, r is the resulting scalar reward value, and s' is the resulting next state. For all of the examples below, samples are collected by choosing among the possible actions uniformly at random.

The predictive projections algorithm is motivated by the observation that in many real world domains the raw state vectors are noisy and high dimensional. The goal is to find a low dimensional projection that adequately captures the relevant task dynamics. This is accomplished by searching for projections that allow for accurate prediction of future states.

It is straightforward to modify Equation (3) so that our regression targets are vector valued future states instead of scalars:

$$f(A) = \sum_i \delta_i = \sum_i \sum_j p_{ij} \left|\left|s'_i - s'_j\right|\right|^2 \qquad (4)$$

Here p_{ij} is calculated by substituting s_i and s_j for x_i and x_j in Equation (1).

The drawback of applying this objective is that it penalizes inaccurate prediction in the original state space, even though our assumption is that the raw state information is noisy and may contain many uninformative dimensions. The resulting gradient may easily be dominated by errors in predicting the irrelevant information. Preliminary experiments using this objective (not shown) were not successful.

The proposed solution is to make state predictions in the same projected space used for neighborhood calculations. In other words, the predictive projections algorithm searches for a transformation matrix that renders the transformed state vectors both predict*ive* and predict*able*. This can be expressed by updating Equation (4) as follows:

$$f(A) = \sum_i \delta_i = \sum_i \sum_j p_{ij} \left|\left|As'_i - As'_j\right|\right|^2 \qquad (5)$$

The problem with this modified objective function is that it can be trivially minimized by setting all of the entries of A to 0.

We remedy this by dividing the squared error term by the squared Frobenius norm of A:

$$f(A) = \sum_i \delta_i = \sum_i \sum_j p_{ij} \frac{\left|\left|As'_i - As'_j\right|\right|^2}{||A||_{\mathrm{F}}^2} \qquad (6)$$

This has the effect of rendering the error term invariant to changes in the scale of A. (Although p_{ij} does remain sensitive to the scale of A.)

Differentiating Equation (6) with respect to A results in the following gradient:

$$\frac{\partial f}{\partial A} = 2A \sum_i \sum_j p_{ij} \left(\delta_i - \frac{\left|\left|As'_i - As'_j\right|\right|^2}{||A||_{\mathrm{F}}^2} \right) s_{ij} s_{ij}^\top + \frac{p_{ij}}{||A||_{\mathrm{F}}^2} \left(s'_{ij} s'^{\top}_{ij} - I \frac{\left|\left|As'_i - As'_j\right|\right|^2}{||A||_{\mathrm{F}}^2} \right) \qquad (7)$$

For the examples presented in this paper Equation (6) is minimized through the method of conjugate gradients [1]. In order to more efficiently evaluate the gradient, we truncate the sums in (7) as suggested in [Goldberger *et al.*, 2004]; 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. For the examples below, the minimization generally converges in less than 15 minutes. Weinberger and Tesauro have shown that algorithms of this type can be scaled up to handle as many as 60,000 training points [2007].

Following this gradient acts to reduce the objective in two different ways. It adjusts A to be more predictive by increasing the probability that a neighbor will be chosen if it successfully predicts the next state. It also adjusts A to be more predictable, by moving target states together whenever there is a high probability they will be chosen to predict each other.

The fact that the algorithm looks for predictable projections makes it necessary to pre-whiten the state vectors. Otherwise the algorithm will preferentially preserve dimensions

[1] In particular, we use Carl Rasmussen's "minimize" package: http://www.kyb.tuebingen.mpg.de/bs/people/carl/code/minimize/.

with low variance because they are comparatively easy to predict whether or not they contain useful structure. The idea is that the original scaling of the data is not important while finding predictive structure is.

So far, our discussion has not accounted for the fact that there will generally be multiple possible actions. This is easily addressed by partitioning the samples according to which action was taken in each and by treating each set of samples as an independent learning problem, with the constraint that they share a common A. It is straightforward to adapt the algorithm described above to this scenario: The optimization objective in Equation (6) is modified to be a sum of objectives of the same form, and the gradient in Equation (7) is modified to be the sum of the corresponding gradients.

In summary, the predictive projections algorithm proceeds as follows:

- Collect samples under an arbitrary policy.
- Pre-whiten the state data.
- Partition samples according to which action was taken in each.
- Select an initial A.
- Minimize the multiple-action version of Equation (6) with respect to A.

The only parameters that need to be selected are the number of rows of A and A's initial value.

The output of the algorithm is the final A matrix. This matrix can then be used to project state data into a space that is more suitable for task learning. Although the objective in (6) minimizes prediction error, the ultimate goal is not to make state predictions, but to discover a projection of the state data that captures the dynamics of the task. Predictions could be used directly to provide a generative model of the task, but we will not do so in this paper.

The predictive projections algorithm as described above may not perform well in cases where the effects of different actions are restricted to specific state dimensions. Since there is no explicit penalty for failing to predict some dimensions, the algorithm may minimize the objective function by finding an A which is not full rank, thus accurately predicting some dimensions while discarding others. This did not prove to be an issue in the domains explored in this paper. In domains where it is an issue, the problem could be addressed by constraining A to be orthonormal and introducing a width parameter in Equation (1) as is done in [Sprague, 2007]. This would force the algorithm to retain exactly d dimensions in the projected space when A is in $\mathbb{R}^{d \times D}$.

3 Examples

For the examples in this paper, we will use the least squares policy iteration (LSPI) algorithm to handle task learning. LSPI has some attractive properties: It can be applied to continuous space problems, and it is an off-policy algorithm, meaning that the same set of samples used for the predictive projections algorithm can be used to find a policy.

LSPI attempts to find a good approximation of the optimal value function $Q^*(s, a)$, which expresses the expected amount of discounted return that the agent will receive if it takes action a in state s and acts optimally thereafter. If this function is known, then the deterministic optimal policy can be specified as:

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

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

$$\hat{Q}(s, a; w) = \sum_{i=1}^{k} \phi_i(s, a) w_i = \phi(s, a)^\top w \tag{8}$$

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 $Q^*(s, a)$.

A major challenge in applying LSPI is selecting appropriate basis functions. The predictive projections algorithm greatly simplifies this process for high dimensional tasks. Rather than specifying basis functions in the original state space, we specify them in the projected space discovered by the algorithm. This is likely to be easier both because the projected space has lower dimensionality than the original space and because it is explicitly constructed to capture the dynamics of the task. All of the examples presented in this paper will make use of radial basis functions (RBFs) for the ϕ_i in Equation (8).

3.1 Pendulum

The first example is a variation of the inverted pendulum balancing task described in [Lagoudakis and Parr, 2003] . In this task the goal is to balance a pendulum by applying forces to an attached cart. The state space of the problem consists of the vertical angle θ and the angular velocity $\dot{\theta}$ of the pendulum. The state dynamics are described by the equation:

$$\ddot{\theta} = \frac{g \sin(\theta) - \alpha m l \dot{\theta}^2 \sin(2\theta)/2 - \alpha \cos(\theta) u}{4l/3 - \alpha m l \cos^2(\theta)}$$

where g is the gravitational constant, m is the mass of the pendulum, and l is its length. The constant α is equal to $1/(m + M)$, where M is the mass of the cart. The variable u represents the amount of force that is applied to the cart. For the experiments presented here, u will take one of three possible values: -50 Newtons (left), $+50$ Newtons (right), or 0 Newtons. These actions are noisy, with uniform noise in the range $[-10, 10]$ added to each action. New actions are selected at .1 second intervals, and the specified force is applied until the next action is chosen. If the angle of the pendulum ever falls below the horizontal, the trial ends, and there is a reward of -1. Each time step that the pendulum stays above the horizontal results in a reward of 0. All constants are the same as those described in [Lagoudakis and Parr, 2003][2].

In the original version of this task, the learning algorithm is given direct access to the state variables θ and $\dot{\theta}$. In the modified version, six dimensions containing normally distributed

[2]The LSPI implementation used in this paper, as well as an implementation of the pendulum simulator can be downloaded from http://www.cs.duke.edu/research/AI/LSPI/.

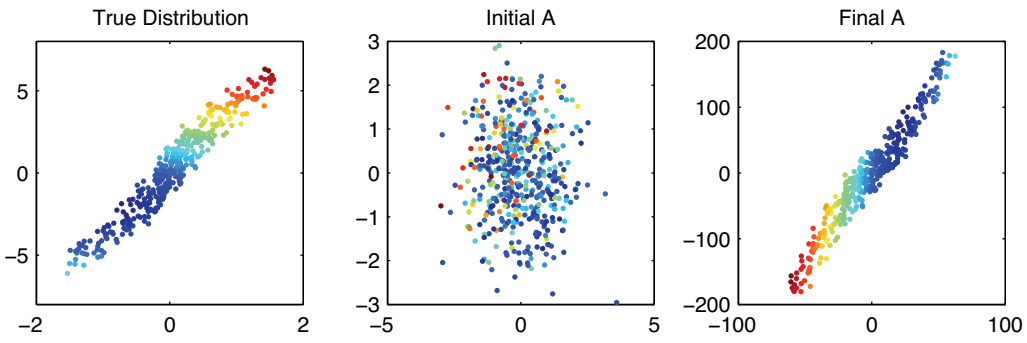

Figure 1: State projections before and after applying the PP algorithm. Corresponding points have the same color in all three figures.

random values are appended to the original state variables, and the resulting vector is projected through a randomly generated mixing matrix:

$$s = X * [\theta, \dot{\theta}, z_1, z_2, z_3, z_4, z_5, z_6]^\top.$$

Here s is the resulting state vector, and X is a randomly generated mixing matrix in $\mathbb{R}^{8\times 8}$ where each entry is uniformly selected from the range $[-1, 1]$. The z_i are selected according to $z_i \sim \mathcal{N}(0, 10)$. For comparison, the variance of θ and $\dot{\theta}$ are approximately 0.35 and 5.97 respectively.

The point of this task is to illustrate that the predictive projections algorithm is able to discover the two relevant state dimensions without direct access to the mixing matrix X. The task proceeds as follows: First, a random X is created. Next, a set of samples is generated during 500 trials, where each trial is initialized with the pendulum upright and stationary, and is limited to a maximum of 100 steps. This generally results in between 4300 and 4500 total samples.

Once the mixed samples are generated, they are prewhitened and passed to the PP algorithm. The A matrix is arbitrarily initialized to:

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

This constrains the metric space discovered by PP to be two dimensional.

Figure 1 illustrates a representative example of the results obtained by the PP algorithm. Figure 1a shows a set of samples in the original space. Figure 1b shows the location of the same samples in the projected space defined by the initial A. Figure 1c shows the samples in the projected space defined by A after the PP algorithm has converged. Although the points in 1c have been rotated and rescaled relative to the "true" distribution illustrated in 1a, it is clear that the PP algorithm has successfully extracted the two state dimensions while suppressing the noise dimensions.

In order to investigate the appropriateness of the of the projections discovered by PP for learning, we repeated the process described above 100 times, generating 100 different projections and 100 corresponding policies. For each run, a new X is generated along with a new set of samples. After running the PP algorithm, the samples are projected through the

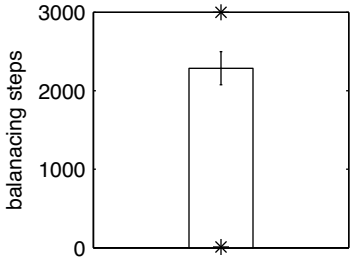

Figure 2: Balancing steps for the pendulum task.

A matrix and finally rescaled to be contained inside the unit square. This rescaling is done to simplify the specification of the radial basis functions that are used by the LSPI algorithm.

For a particular state and action value, all basis functions are zero except for those corresponding to the current action, which have the form:

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

Here the μ_i's are the positions of the radial basis functions in the projected and scaled space, and $\sigma = .02$ is a width parameter in that same space.

In [Lagoudakis and Parr, 2003] radial basis functions are positioned in a 3×3 grid centered at the origin. They are spaced to provide good coverage of state values that correspond to states where the pendulum has not already fallen past a point where it can be recovered. In this modified version of the task, it is not possible to predict in advance the position or orientation of that recoverable region in the projected space. Therefore, rather than positioning the RBFs in a fixed grid, we randomly select a subset of training points in the projected and scaled space to serve as RBF centers. The selection of samples is accomplished by randomly ordering all of the sample points, and sequentially selecting points as long as no other point is within a distance of .05. Once the end of the list of points is reached, all candidate points that are within .05 of any of the previously selected points are removed, and the process is repeated until no candidates re-

Figure 3: The robot and a sample of its visual input (inset).

main. This ensures that no sample point will be farther than .1 from an RBF center. The resulting number, n, of RBF centers ranged from 16 to 43 depending on the selection process and the distribution of points.

Once LSPI has discovered a policy for each of the 100 runs, those policies are tested by executing them for 100 trials each. Each trial is terminated after 3000 successful balancing steps. The results are presented in Figure 2. The error bars show 95% confidence intervals for the mean number of steps for each run. Error bars for trials within runs are not displayed. The stars illustrate the maximum and minimum mean number of balancing steps across the 100 runs. The average number of balancing steps is more than 2000, or more than 3 minutes of real time. These results are comparable to, though somewhat lower than, those presented in [Lagoudakis and Parr, 2003] where the learning takes place directly in the original state space using radial basis functions that are hand positioned. Looking over the discovered projections suggests that those runs that do not result in successful policies do not fail because the predictive projections algorithm failed to find an appropriate projection. Instead, the samples were not well distributed, or the randomly positioned RBFs ended up in inappropriate locations.

3.2 Robot

The second task is a visual navigation problem on a mobile robot. The robot is an iRobot Create with a small laptop mounted on top. An inexpensive USB webcam provides visual input. The task involves moving within a 132cm $\times$ 132cm enclosure without colliding with the walls. Figure 3 shows the robot and the enclosure as well as an example of the input from the camera[3]

At each time step the robot chooses from one of three possible actions: Left (turn in place -18^o), Right (turn in place 18^o), and Forward (move ahead 7.5cm). A reward of 1.0 is received each time the robot moves forward without triggering its bump sensor. If the robot ever collides with a wall, it receives a reward of 0, and the trial is terminated.

The robot collects training data by following a random policy for 6000 steps. After each collision, the robot turns in place 180^o and begins a new trial from the resulting position.

The images captured by the camera consist of 160 $\times$ 120 color pixels. These images are first cropped down to 160 $\times$ 100 (cropping out the front of the robot) and then rescaled to 20 $\times$ 20 pixels. This results in a 1200 dimensional input vector: three color channels with 400 entries each. In the interest of computational efficiency, these 1200 dimensional vectors are reduced to 25 dimensions by applying principal components analysis (PCA) and keeping only the 25 dimensions with the highest variance. For the results presented below, these 25 dimensions account for approximately 91% of the variance in the data set.

In this case, the goal of the predictive projections algorithm will be to reduce the resulting 25 dimensional input vectors to 2 state dimensions that are appropriate for learning the navigation task. The 25 dimensional input vectors are whitened, and the 2×25 A matrix is initialized to

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & \dots \\ 0 & 1 & 0 & 0 & \dots \end{bmatrix}.$$

Thus the initial projection corresponds to the first two principal components. We also experimented with randomly initializing the A matrix, with comparable results.

After the predictive projections algorithm has converged, the final A matrix is orthonormalized using the Gram-Schmidt process. This is not necessary, but it makes comparing the projection discovered by PP to those discovered by PCA more straightforward, and it leads to a slightly better policy. As in the pendulum task, the two dimensional state vectors are scaled to fit in the unit square after they are projected through the matrix A. The resulting two-dimensional samples are passed to LSPI for policy learning. Once again we use RBF basis functions. However, for this task the RBF centers are positioned at the 16 points $\{.2, .4, .6, .8\} \times \{.2, .4, .6, .8\}$ rather than being randomly positioned and $\sigma^2 = .03$. For the sake of comparison, we also use LSPI to learn a policy in the space defined by the first two principal components. The same training data and the same set of basis functions is used.

Figure 4 shows the projection defined by the first two principal components, as well as the projection discovered by the predictive projections algorithm. The color of the samples in this figure represent the action that is selected at that point under the corresponding policy. Green represents Forward, red represents Left and blue represents Right.

The projection discovered by the PP algorithm clearly uncovers the intrinsic two dimensional structure of this task. The horizontal axis corresponds to translation while the vertical axis corresponds to rotation. In contrast, the projection discovered by PCA is not relevant to the task. The first principal component seems to capture the amount of wall present in the image while the second captures the overall lighting level. These different projections result in substantially different policies. It is difficult to define an effective policy in the space recovered by PCA because states that are functionally very different (wall on the left versus wall on the right) end up near each other in the projected space. In contrast, LSPI is able to discover a simple policy in the space discov-

[3]Robot control is handled through the COIL library(http://code.google.com/p/libcreateoi/). Visual processing is handled by the OpenCV library (http://sourceforge.net/projects/opencvlibrary/).

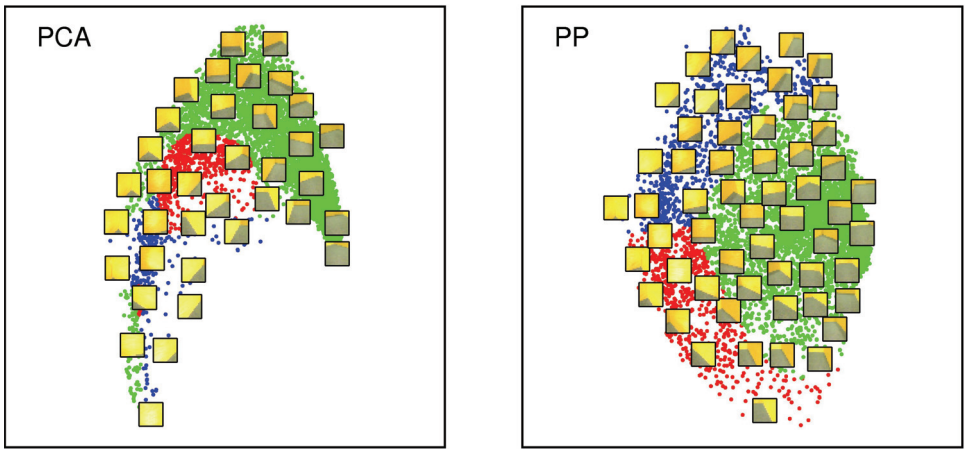

Figure 4: State projections under PCA and PP. A subset of samples are labeled with their corresponding visual input.

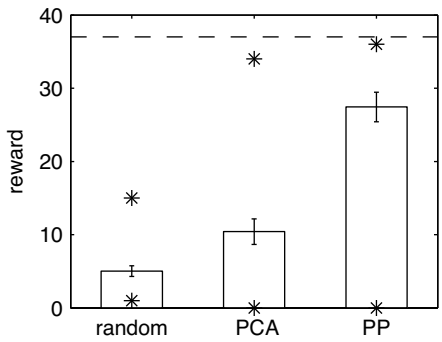

Figure 5: Comparison of reward for three different policies.

ered by the PP algorithm. The robot moves forward when the wall is sufficiently far away and turns left or right as appropriate when it approaches a wall. This figure would look essentially the same if we had not orthonormalized A, except that the distribution of points on the right would be somewhat elongated along the "translation" axis.

The two policies were tested by executing them for 80 trials each. Each trial begins at a random position and orientation in the enclosure and continues for 50 steps or until a collision occurs. Figure 5 compares the amount of reward received under the two policies, as well as the amount of reward received by a random policy under the same conditions. The stars indicate maximum and minimum reward values across all trials. The dotted horizontal line corresponds to the maximum amount of reward received by a human operator during three trials of steering the robot from an optimal starting position. As expected, the policy derived from the PP projection dramatically outperforms the random policy, as well as the policy derived from PCA.

The fact that the PP policy does not consistently perform at the same level as the human operator results from two main factors. First, the starting position is random, so it may be necessary to make several turns at the beginning of a trial. Second, this task suffers from some perceptual aliasing. Since the robot has a limited field of view, there are functionally distinct states that appear identical. For example, if the robot is facing a wall near a corner, it has no way of knowing that the corner is there. This can cause the robot to dither; It turns left until the corner disappears out of its field of view to the right, then it turns right until the corner disappears out of its field of view to the left. This is a separate issue from that addressed by the predictive projections algorithm.

4 Related Work

A project that is close in spirit to the work presented here is the action respecting embedding (ARE) algorithm described in [Bowling *et al.*, 2005]. That algorithm uses semidefinite programming to find a non-linear embedding of state data in which actions represent distance preserving transformations. The ARE algorithm is less restrictive in the class of transforms it is able to discover but more restrictive in its conception of the effects that actions must have in the recovered space. In its original formulation, ARE would be impossible to apply to the tasks presented in this paper because it does not provide a mechanism for mapping new points into the discovered manifold. However, the authors have recently described several variants of the algorithm that scale to larger data sets and make it possible to map out-of-sample points [Biggs *et al.*, 2008]. It would be worthwhile to compare the performance of these algorithms to predictive projections on the tasks explored in this paper.

There have been a number of recent algorithms designed to automatically generate an appropriate set of basis functions for learning Markov decision problems. The most popular of these is the proto-value functions (PVF) approach described in [Mahadevan and Maggioni, 2007] In the proto-value function framework, sample state trajectories are used to learn a diffusion model of the state dynamics. For discrete state systems, this is a weighted graph where connections are inferred from observed transitions. For a system with continuous state values, a subset of observed states serves as the nodes in the graph, and nodes are connected to their k nearest neighbors.

In either case, basis functions are found by calculating the eigenvectors of the graph Laplacian of the learned diffusion matrix. The eigenvectors with the smallest eigenvalues form a compact representation that captures the large-scale temporal features of the transition process. Empirically, these basis functions have shown good performance on a number of challenging test problems.

To our knowledge, the proto-value function framework has not been applied to the type of noisy, high dimensional control problems addressed in this paper. It seems likely that the neighborhood calculations required for constructing the diffusion model could be dominated by noise dimensions, particularly in very noisy tasks such as the modified pendulum domain described above. In that case, the PVF approach and predictive projections would be complementary: The PP algorithm could find a low dimensional state projection that contains relevant state information, and the PVF algorithm could then be used to discover a set of appropriate basis functions in that space.

Another closely related project is the basis iteration algorithm described in [Sprague, 2007]. This algorithm also uses gradient based metric learning to discover an appropriate projection, but it focuses directly on finding a metric that allows for accurate estimation of the optimal value function. It accomplishes this by iterating value function estimation with updates to the projection matrix. This algorithm has the advantage of incorporating reward information, but it depends on starting with an initial projection that enables a reasonable estimate of the optimal value function. This can be difficult, especially when reward is absent or very intermittent. The work presented here was motivated, in part, by the fact that the basis iteration algorithm fails to find appropriate projections for the tasks described above.

The work presented here was directly inspired by the analysis presented in [Parr *et al.*, 2008]. That paper demonstrates that the Bellman error in algorithms, such as LSPI, that make use of linear methods, can be expressed as a combination of errors in the prediction of next feature values and errors in predicting reward. The PP algorithm attempts to find projections that make it possible to minimize feature error.

5 Discussion and Future Work

One direction for future work is to explore ways to incorporate reward information in the process of finding projections. Ultimately, a good projection is one that makes it possible to maximize reward on the task. It is not difficult to envision cases in which there are predictive projections that are not actually relevant to a task, as well as cases where relevant projections are predictive of reward but not of future states. The most straightforward approach would be to add a term to Equation (6) that penalizes error in reward prediction.

6 Conclusion

For real world control problems, the state description provided by sensor data is often not well suited for specifying controllers. In these cases it is necessary to project raw state data onto some feature space that captures the essential elements of the state dynamics. The simple premise of the predictive projections algorithm is that we should search for a feature space such that features of the current state are maximally predictive of features of the next state. The effectiveness of that approach has been demonstrated on a visually guided navigation task. For that task, appropriate visual features are learned almost entirely from scratch, and as a result it is possible to discover a nearly optimal control policy using a standard reinforcement learning algorithm.

Acknowledgments

I wish to thank the anonymous reviewers for their helpful comments and suggestions.

References

[Biggs *et al.*, 2008] Michael Biggs, Ali Ghodsi, Dana Wilkinson, and Michael Bowling. Scalable action respecting embedding. In *Proceedings of the Tenth International Symposium on Artificial Intelligence and Mathematics (ISAIM)*, 2008.

[Bowling *et al.*, 2005] Michael Bowling, Ali Ghodsi, and Dana Wilkinson. Action respecting embedding. In *Proceedings of the 22nd International Conference on Machine Learning*, Bonn, Germany, 2005.

[Goldberger *et al.*, 2004] Jacob Goldberger, Sam Roweis, Geoff Hinton, and Ruslan Salakhutdinov. Neighbourhood component analysis. In *Neural Information Processing Systems*, 2004.

[Keller *et al.*, 2006] 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.

[Lagoudakis and Parr, 2003] Michail G. Lagoudakis and Ronald Parr. Least-squares policy iteration. *Journal of Machine Learning Research*, 4:1107–1149, 2003.

[Mahadevan and Maggioni, 2007] Sridhar Mahadevan and Mauro Maggioni. Proto-value functions: A Laplacian framework for learning representation and control in markov decision processes. *Journal of Machine Learning Research*, 8:2169–2231, 2007.

[Parr *et al.*, 2008] Ronald Parr, Lihong Li, Gavin Taylor, Christopher Painter-Wakefield, and Michael L. Littman. An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2008.

[Sprague, 2007] Nathan Sprague. Basis iteration for reward based dimensionality reduction. In *Proceedings of the 6th IEEE International Conference on Development and Learning (ICDL)*, London, 2007.

[Weinberger and Tesauro, 2007] Kilian Weinberger and Gerald Tesauro. Metric learning for kernel regression. In *Eleventh International Conference on Artificial Intelligence and Statistics*. Puerto Rico, 2007.