Action Respecting Embedding

Michael Bowling BOWLING@CS.UALBERTA.CA
Department of Computing Science, University of Alberta, Edmonton AB, Canada

Ali Ghodsi AGHODSIB@UWATERLOO.CA
Department of Statistics and Actuarial Science, University of Waterloo, Waterloo, ON, Canada

Dana Wilkinson D3WILKIN@UWATERLOO.CA
School of Computer Science, University of Waterloo, Waterloo, ON, Canada

Abstract

Dimensionality reduction is the problem of finding a low-dimensional representation of high-dimensional input data. This paper examines the case where additional information is known about the data. In particular, we assume the data are given in a sequence with action labels associated with adjacent data points, such as might come from a mobile robot. The goal is a variation on dimensionality reduction, where the output should be a representation of the input data that is both low-dimensional and respects the actions (*i.e.*, actions correspond to simple transformations in the output representation). We show how this variation can be solved with a semidefinite program. We evaluate the technique in a synthetic, robot-inspired domain, demonstrating qualitatively superior representations and quantitative improvements on a data prediction task.

1. Introduction

Dimensionality reduction and manifold learning are popular topics in machine learning. Traditionally, linear dimensionality-reduction techniques, such as principle components analysis, have been used to find low-dimensional linear subspaces in high-dimensional data. Manifolds in natural data are rarely linear, however, leading to a variety of research in discovering non-linear manifolds.

Historically, the two main ideas for discovering low-dimensional manifolds in high-dimensional data have been to find a mapping from the original space to a lower-dimensional space that: (i) preserves pairwise distances, *e.g.*, multidimensional scaling (Cox & Cox, 2001); or (ii) preserves mutual linear reconstruction ability, *e.g.*, principle components analysis (Jolliffe, 1986). In each case, globally optimal solutions are linear manifolds. The more recent techniques for manifold discovery, *e.g.*, Isomap (Tenenbaum et al., 2000), LLE (Saul & Roweis, 2003), and SDE (Weinberger & Saul, 2004b), are based on these same two principles, with the generalization that the new methods only seek low-dimensional representations that *locally* preserve distances or linear reconstructions. In this way, they avoid recovering globally linear solutions.

Although these techniques produce non-linear manifolds in different ways, they all share one feature. All knowledge about the input data, and therefore the desired low-dimensional manifold, must be encoded in the similarity function. Not all such knowledge can be so easily encoded. Consider sensor readings, such as images, taken from a mobile robot. The most natural representation of the observations would be the robot's pose (*e.g.*, for a wheeled robot: x, y and θ describing the robot's position and orientation), which allows the high-dimensional sensor data to be described with only a few dimensions. This representation is desirable not only because it is low-dimensional, but because within it the robot's actions (*e.g.*, forward and rotation) correspond to simple transformations. This objective pose is an ideal representation for robot planning and localization. There is no natural way, though, to encode either the robot's actions nor the desire that the representation respect these actions through a simple similarity function.

This paper introduces a new algorithm, Action Respecting Embedding (ARE), to address this variation on traditional manifold learning. Specifically, we examine situations where the input data are given in sequence, along with uninterpreted[1] action labels that are associated with adja-

Appearing in *Proceedings of the 22^{nd} International Conference on Machine Learning*, Bonn, Germany, 2005. Copyright 2005 by the author(s)/owner(s).

[1]This means that the action labels themselves have no implied meaning. We may refer to actions as 'go left' or 'go right' but the

cent pairs of data points. ARE learns a low-dimensional representation of the data, in which the actions are simple transformations. For ARE to extract such a representation it exploits the knowledge of action labels in two key ways:

1. It uses the action-labeled pairs of data points to build a *non-uniform neighborhood graph*. The graph is constructed using the assumption that pairs of data points that can be reached in a small number of actions should be nearby in the learned representation. Other non-linear manifold-learning techniques use a k-nearest neighbor graph with a globally uniform k which can create overly dense neighborhood graphs.

2. The action labels individually have no implied meaning. However, every time an action is repeated it provides more implicit information about the data. From these repetitions we can build *action-respecting constraints* that ensure that each action corresponds to a simple transformation in the learned representation.

Using non-uniform neighborhoods and action-respecting constraints, ARE constructs a semidefinite program to learn a kernel that describes the desired low-dimensional representation. The result is a very natural representation of the original high-dimensional data, with a strong correspondence to the actual low-dimensional process that generated the data. Although manifold-learning techniques often rely on qualitative evaluation, our knowledge of the actions involved in generating the data allows for a more objective evaluation. Therefore, along with traditional qualitative comparisons we introduce a data-prediction task as a quantitative measure of the success of a learned representation.

In Section 2 of this paper, we review previous relevant manifold-learning techniques. The focus is on Semidefinite Embedding, which is the foundation for our new algorithm. The Action Respecting Embedding algorithm is introduced in Section 3. We extract a non-uniform neighborhood graph based on the fact that the data are connected by actions, and we create additional manifold constraints which respect the action labeling. We also introduce the task of data prediction and show how ARE can solve this problem. Experimental results of the proposed algorithm are presented in Section 4 before we conclude in Section 5.

2. Background

Dimensionality reduction or manifold learning can be seen as the process of deriving a set of degrees of freedom which can be used to reproduce most of the variability of a data set. For example, consider a set of images produced by rotating a camera through different angles. Clearly only one degree of freedom is being altered, and thus the images lie along a continuous curve through image space.

algorithm gets the actions as simply 'Action 1' and 'Action 2'.

Many algorithms for dimensionality reduction have been developed, beginning with PCA. Principal components analysis (PCA) (Jolliffe, 1986) is a classical method which provides a sequence of best linear approximations to a given high-dimensional observation. It is a popular technique for dimensionality reduction, but its effectiveness is limited by its global linearity. Multidimensional scaling (MDS) (Cox & Cox, 2001), closely related to PCA, suffers from the same drawback. In order to resolve the problem of dimensionality reduction in non-linear cases, many techniques including Kernel PCA (Schölkopf et al., 1998; Mika et al., 1999; Schölkopf & Smola, 2002), locally linear embedding (LLE) (Roweis & Saul, 2000; Saul & Roweis, 2003), Isomap (Tenenbaum, 1998; Tenenbaum et al., 2000), and Semidefinite Embedding (Weinberger & Saul, 2004b) have been proposed. Motivating our algorithm requires a brief overview of Kernel PCA and SDE.

Kernel PCA is a non-linear generalization of PCA. In Kernel PCA, using kernels, principle components are computed efficiently in high-dimensional feature spaces that relate to the input space by some non-linear mapping. PCA finds an orthogonal transformation of the coordinate system which describes the data. Kernel PCA finds principal components which are non-linearly related to the input space. The key observation is that PCA can be formulated entirely in terms of dot products between data points. In Kernel PCA, this dot product is replaced by the inner product of a Hilbert space—equivalent to performing PCA in the space produced by the non-linear mapping, where the low-dimensional latent structure is easier to discover.

Consider a feature space $\mathcal{H}$ such that $\Phi : X \to \mathcal{H}$. Let $\sum_{i=1}^{n} \Phi(x_i) = 0$ (since a simple transformation on X can center the data). The solution for PCA could be found by taking the singular value decomposition:

$$\Phi(X) \quad = \quad U \Sigma V^T \tag{1}$$

where U contains the eigenvectors of $\Phi(X)\Phi(X)^T$, Σ is a diagonal matrix containing the square roots of the eigenvalues of $\Phi(X)\Phi(X)^T$ and $\Phi(X)^T\Phi(X)$, and V contains the eigenvectors of $\Phi(X)^T\Phi(X)$. The primal PCA solution for encoding the data is $Y = U^T\Phi(X)$. Since $\Phi(X)$ might be very high-dimensional, simply applying PCA might be impractical. From equation 1, $U^T\Phi(X) = \Sigma V^T$. This is the dual form of PCA which allows us to employ the kernel function $k(\cdot,\cdot)$ to compute the kernel matrix $K = \Phi(X)^T\Phi(X)$ where $K_{ij} = k(x_i, x_j)$. Note that this matrix does not depend on the dimensionality of the feature space. The Kernel PCA procedure is summarized in Table 1. The choice of kernel plays an important role—linear, polynomial and Gaussian kernels are widely used kernels which reveal different types of low-dimensional structure.

Algorithm: Kernel PCA

Recover basis: Calculate $\Phi(X)^\top\Phi(X) = K$ and let V be the eigenvectors of K corresponding to the top d eigenvalues. Let $\Sigma =$ diagonal matrix of *square roots* of the top d eigenvalues.

Encode training data: $Y = U^\top\Phi(X) = \Sigma V^\top$ where Y is a $d \times n$ matrix of encodings of the original data.

Table 1. Kernel PCA Algorithm.

In 2004 Weinberger and Saul introduced SDE (Weinberger & Saul, 2004b; Weinberger & Saul, 2004a), which learns a kernel matrix instead of choosing a kernel function a priori. They formulated the problem of learning the kernel matrix as an instance of semidefinite programming. Since the kernel matrix K represents inner products of vectors in a Hilbert space it must be positive semidefinite. Also the kernel should be centered, *i.e.*, $\sum_{ij} K_{ij} = 0$. Lastly, SDE imposes constraints on the kernel matrix to ensure that the distances and angles between points and their neighbors are preserved under the neighborhood graph η. That is, if both x_i and x_j are neighbors (*i.e.*, $\eta_{ij} = 1$) or are common neighbors of another input (*i.e.*, $[\eta^T\eta]_{ij} > 0$), then:

$$||\Phi(x_i) - \Phi(x_j)||^2 = ||x_i - x_j||^2.$$

In terms of the kernel matrix, this can be written as:

$$K_{ii} - 2K_{ij} + K_{jj} = ||x_i - x_j||^2.$$

By adding an objective function to maximize $\text{Tr}(K)$, which represents the variance of the data points in the learned feature space, SDE constructs a semidefinite program for learning the kernel matrix, K. The last detail of SDE is the neighborhood graph, η_{ij}, constructed by connecting the k nearest neighbors using a similarity function over the data, $||x_i - x_j||$. The algorithm is summarized in Table 2.

```
Algorithm: SDE

Construct neighbors, $\eta$, using $k$-nearest neighbors.

Maximize $\text{Tr}(K)$ subject to $K \succeq 0$, $\sum_{ij} K_{ij} = 0$, and
  $\forall ij$   $\eta_{ij} > 0 \vee [\eta^T\eta]_{ij} > 0 \Rightarrow$
          $K_{ii} - 2K_{ij} + K_{jj} = ||x_i - x_j||^2$     we    Run

Kernel PCA with learned kernel, $K$.
```

Table 2. SDE Algorithm.

The manifolds learned by SDE are comparable to those of other non-linear dimensionality-reduction methods. Also, at its core is a semidefinite optimization. The next section demonstrates that our variant on dimensionality reduction can be solved by adding appropriate constraints to this core.

3. Action Respecting Embedding

Action respecting embedding takes a sequence of high-dimensional data $x_1, \ldots, x_n$, along with associated discrete actions $a_1, \ldots, a_{n-1}$. The data are assumed to be in some order, where action a_i was taken between data points x_i and x_{i+1}. The final piece of input is a similarity function, $||x_i - x_j||$, defining a distance over the high-dimensional data points. For vector data, Euclidean distance is often sufficient, but other similarities can be used.

The overall structure of the algorithm follows the same three steps of SDE: (i) construct a neighborhood graph, (ii) solve a semidefinite program to find the maximum variance embedding subject to constraints, (iii) extract a low-dimensional embedding from the dominant eigenvectors of the learned kernel matrix. ARE, though, seeks to exploit the additional information provided by the action labels of the data. We exploit this information through two key insights. The first modifies step (i) by constructing non-uniform neighborhoods based on action-labeled pairs of data points. The second modifies step (ii) by adding action-respecting constraints into the semidefinite program.

3.1. Non-Uniform Neighborhoods

Many of the current non-linear manifold-learning techniques seek to preserve local properties of the original data. They often require a neighborhood graph over the original data points to define a notion of locality. As we've seen, SDE creates this graph by connecting each data point to its k-nearest neighbors for some chosen value of k. Since the neighborhood graph must be fully connected for SDE to have a bounded solution, this choice of k can be forced to be quite large and may over-constrain the learned manifold. Another possibility would be to choose a distance threshold δ and connect any two data points within that threshold as neighbors. Again, this may result in an over-constrained manifold as δ must be set large enough to make the graph fully connected. The key drawback in these techniques is that they require a globally uniform k or δ.

Since we are given additional information relating the points in our set, *i.e.*, that certain pairs of data points are connected by an action, we can build a more intuitive, non-uniform neighborhood graph. The idea is based on the assumption that data points connected by an action are nearby and should be considered neighbors. We use these assumed neighbors to define a neighborhood ball around each data point, whose radius is large enough to encompass all data points connected by an action. We then include an edge in the neighborhood graph between two images if they are both in each other's neighborhood ball. We can increase the connectivity of the neighborhood graph by increasing the action window, *i.e.*, requiring data points within T actions of each other to be neighbors. Since our data is generated

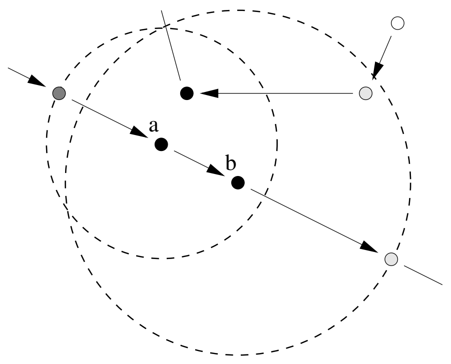

Figure 1. An example of the use of action labels to find non-uniform neighborhoods. The arrows show the points that are connected by an action. The circles show the neighborhood for the points labeled ‘a’ and ‘b’ with $T = 1$. Black points are in both, white points in neither. Shaded points are in ’b’ but not ’a’.

from a sequence of actions, we can define the neighborhood graph as follows. Let η_{ij} be the adjacency matrix of the neighborhood graph. Given an action window of T,

$$\begin{aligned} \eta_{ij} = 1 \quad \Leftrightarrow \quad & \exists k, l \quad \text{such that} \\ & |k - i| < T, \quad |l - j| < T, \\ & ||x_i - x_k|| > ||x_i - x_j|| \quad \text{and} \\ & ||x_j - x_l|| > ||x_i - x_j||. \end{aligned} \qquad (2)$$

Figure 1 shows some two-dimensional points connected by actions, and the resulting neighborhood balls when $T = 1$.

Note that since the data come from a sequence of actions, the neighborhood graph ($T \geq 1$) is fully connected. This satisfies a critical requirement that the semidefinite optimization be bounded (or a solution may not exist).

3.2. Action-Respecting Constraints

The second, and more important, contribution of ARE is the addition of action-respecting constraints. The evaluation of learned manifolds is often subjective and usually amounts to demonstrating that a manifold corresponds to a known data generator’s own underlying degrees of freedom. Action labels, even without interpretation or implied meaning, provide more information about the underlying generation of the data. It is natural to expect that the actions correspond to some simple operator on the generator’s own degrees of freedom. For example, a camera that is being panned left and then right, has actions that correspond to a simple translation in the camera’s actuator space. We therefore want to constrain the learned representation so that labeled actions correspond to simple transformations in it. In particular, an action should correspond to a rotation-plus-translation [2] in the low-dimensional representation.

[2] The subset of linear transforms that don’t involve scaling.

This constraint can be formalized by first observing that rotation-plus-translation is exactly the space of *distance-preserving* transformations. Transformation f is distance preserving, thus a rotation-plus-translation, if and only if:

$$\forall x, x' \quad ||f(x) - f(x')|| = ||x - x'||.$$

Consider this in the context of an action-labeled data sequence. All actions must be distance-preserving transformations in the learned representation. Therefore, for any two data points, x_i and x_j, the same action taken at each data point must preserve the distance between them. Let $\Phi(x_i)$ denote data point x_i in the the learned space, then action a’s transformation, f_a, must satisfy:

$$\begin{aligned} \forall i, j \qquad & ||f_a(\Phi(x_i)) - f_a(\Phi(x_j))|| = \\ & ||\Phi(x_i) - \Phi(x_j)||. \end{aligned} \qquad (3)$$

Now, let $a = a_i$ and consider the case where $a_j = a_i$. Then, $f_a(\Phi(x_i)) = \Phi(x_{i+1})$ and $f_a(\Phi(x_j)) = \Phi(x_{j+1})$, so Constraint 3 becomes:

$$||\Phi(x_{i+1}) - \Phi(x_{j+1})|| = ||\Phi(x_i) - \Phi(x_j)||. \qquad (4)$$

We don’t want to pose this as a constraint on distances, but rather as a constraint on inner products (*i.e.*, on the learned kernel matrix, K). Squaring both sides of the equation and rewriting in terms of K results in the following constraints:

$$\begin{aligned} \forall i, j \quad & a_i = a_j \Rightarrow \\ & K_{(i+1)(i+1)} - 2K_{(i+1)(j+1)} + K_{(j+1)(j+1)} = \\ & K_{ii} - 2K_{ij} + K_{jj} \end{aligned} \qquad (5)$$

We can add Constraint 5 into SDE’s usual constraints to arrive at the optimization and algorithm shown in Table 3. There is a slight modification to SDE’s usual neighbor constraint, changing strict equality into an upper bound. This modification insures that the constraints are feasible by allowing the zero matrix to be a feasible solution. Notice that the additional action-respecting constraints are still linear in the optimization variables, K_{ij}, and so the optimization remains a semidefinite program. Since the neighborhood graph η_{ij} is fully connected, the optimization is bounded, convex, and feasible, and therefore can be solved efficiently with various general-purpose toolboxes. The results in this paper were obtained using CSDP (Borchers, 1999) in MATLAB. Our results also used highly-penalized slack variables in SDE’s neighborhood constraint to help improve solution stability. This was recommended by Weinberger *et al.* in the original SDE paper (Weinberger & Saul, 2004b).

3.3. Data Prediction

As manifold learning is an unsupervised learning problem, evaluation of algorithms is often qualitative. We now introduce the task of *data prediction*, which (i) can be measured

```
Algorithm: ARE
Construct neighbors, $\eta$, according to Equation 2.
Maximize $\mathrm{Tr}(K)$ subject to $K \succeq 0$: $\sum_{ij} K_{ij} = 0$,
  $\forall ij$  $\eta_{ij} > 0 \vee [\eta^T \eta]_{ij} > 0 \Rightarrow$
        $K_{ii} - 2K_{ij} + K_{jj} \leq ||x_i - x_j||^2$ , and
  $\forall ij$  $a_i = a_j \Rightarrow$
        $K_{(i+1)(i+1)} - 2K_{(i+1)(j+1)} + K_{(j+1)(j+1)} =$
        $K_{ii} - 2K_{ij} + K_{jj}$
Run Kernel PCA with learned kernel, $K$.
```

Table 3. ARE Algorithm.

quantitatively and (ii) seeks to evaluate how well a low-dimensional representation has captured the actions. Data prediction is: given a data point and an action, predict the resulting data point. In general, this is a very challenging task. Manifolds learned with ARE can be used to tackle a partial version of this task: given a data point and action from the training set, x_i and a (where a is not necessarily a_i), predict the next data point assuming it is also in the training set. Here we describe how ARE can be used to solve this task, and in Section 4 we present results of this quantitative evaluation of accuracy of ARE's predictions.

ARE learns a space where actions correspond to distance-preserving operators. By Constraint 3, this implies:

$$\forall i, j \quad ||f_a(\Phi(x_i)) - f_a(\Phi(x_j))|| = ||\Phi(x_i) - \Phi(x_j)||.$$

Considering only j's such that $a_j = a$, results in the following constraint on the result of the action's transformation:

$$\forall j \quad a_j = a \Rightarrow \quad ||f_a(\Phi(x_i)) - \Phi(x_{j+1})|| = ||\Phi(x_i) - \Phi(x_j)||. \tag{6}$$

If action a appears in the training set m times, then this gives m constraints on $f_a(\Phi(x_i))$'s distance to other known points, $\Phi(x_{j+1})$. In fact, if the learned manifold has dimensionality d, $d+1$ independent distance constraints uniquely determine $f_a(\Phi(x_i))$. In this case, it is a simple matter to find point $\Phi(x_p)$ nearest the constrained point $f_a(\Phi(x_i))$, and use x_p as our prediction. If a point is under-constrained ($m <= d$), then the index, p, is selected by:

$$p = \operatorname*{argmax}_{k=1\ldots n} \sum_{j:a_j=a} \left(\begin{array}{c} ||f_a(\Phi(x_i)) - \Phi(x_{j+1})|| - \\ ||\Phi(x_k) - \Phi(x_{j+1})|| \end{array} \right)^2. \tag{7}$$

In other words, $\Phi(x_p)$ is the embedded point whose distances to other points most closely agrees with $f_a(\Phi(x_i))$'s distance constraints. We then use x_p as our prediction.

4. Results

We now examine the effect of ARE's non-uniform neighborhoods and action-respecting constraints on learning low-dimensional action-respecting representations. Our results are in a synthetic, robot-inspired, image manipulation domain called IMAGEBOT. We first present this domain. We then show manifolds produced by ARE and SDE from data generated in this domain. In addition to the compelling qualitative comparisons, we also present quantitative evaluation using the data prediction task described in Section 3.3

4.1. IMAGEBOT Domain

Given an image, one can imagine a virtual robot that observes a small patch on that image and can take actions which move that observable patch around on the image. This "image robot" provides an excellent domain for testing ARE, with obvious connections to robotic applications.

For these experiments, IMAGEBOT is always viewing a 200 by 200 patch of a 2048 by 1536 image. IMAGEBOT is restricted to eight distinct actions: four translation actions, two rotation actions and two zoom actions. The translations are 'forward', 'backward', 'left' and 'right', each by 25 pixels. The rotation actions are 'turn left' and 'turn right', each by $22\frac{1}{2}^\circ$. The zoom actions are 'zoom in' and 'zoom out', each changing the scale by a factor of $\sqrt[8]{2}$ (*i.e.*, eight zoom actions double the image scale).

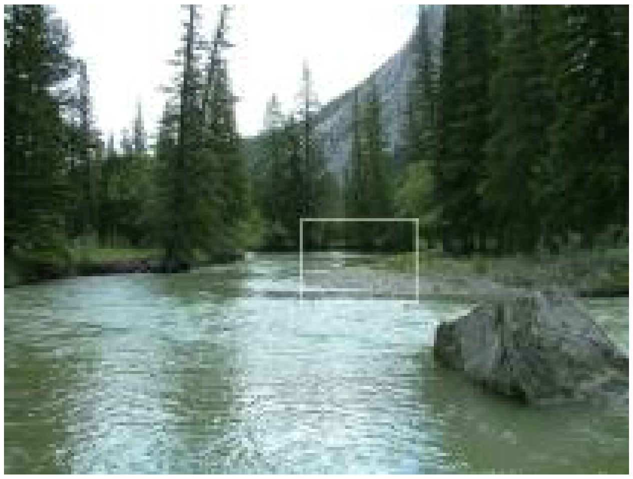

Figure 2. IMAGEBOT's world.

Figure 2 shows the image used for the experiments, while Figure 3 shows an example trajectory from IMAGEBOT (Figure 3 is an enlargement of the long, thin highlighted rectangular section in Figure 2.) The trajectory starts on the far left with IMAGEBOT facing right. IMAGEBOT then takes 40 steps forward (to the right) and then 20 steps backward. Figure 4 shows a more complicated 'A'-shaped trajectory that IMAGEBOT followed (Figure 4 is a blow up of

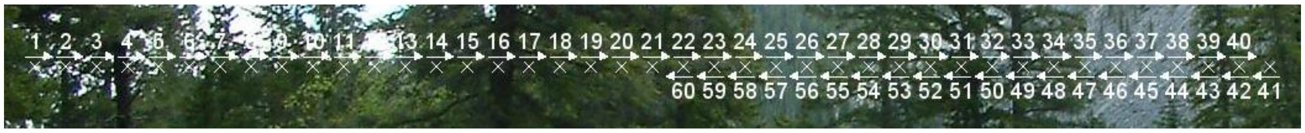

Figure 3. A sample 60-action trajectory from IMAGEBOT.

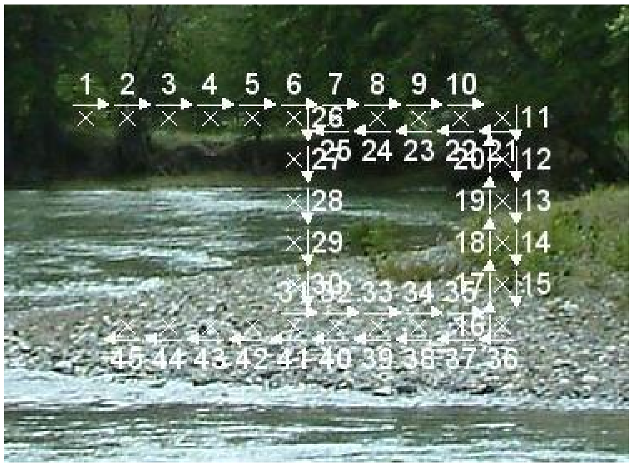

Figure 4. A more complicated 45-action trajectory from IMAGEBOT.

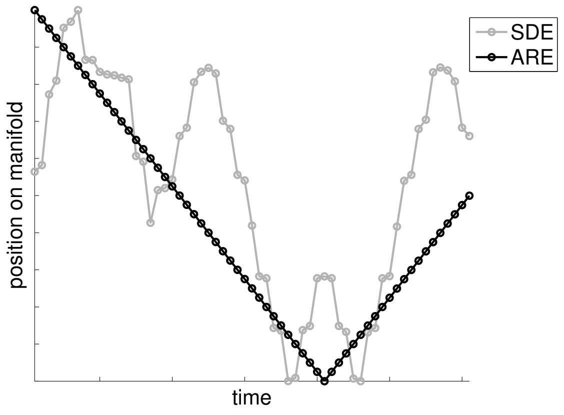

Figure 5. Manifolds from trajectory shown in Figure 3. Lines show the distance along the manifold over time.

the other highlighted rectangular section in Figure 2.)

IMAGEBOT's observations as it follows these paths, along with the actions associated with the paths, gives a perfect domain for testing ARE—ordered high-dimensional data with each consecutive pair related by an action. Note that while IMAGEBOT knows what action it takes at every step there is no semantic information associated with that knowledge, *i.e.*, the labels are uninterpreted.

4.2. Manifold Learning

Both SDE and ARE were applied to the IMAGEBOT data from the trajectory in Figure 3. As might be expected, the resulting manifold for both algorithms is not surprising—essentially one-dimensional as the first eigenvalue of the resulting kernel dominates the others. Of interest, however, is a plot of the trajectory on this manifold over time, which is shown in Figure 5. Note that the result from SDE indicates that IMAGEBOT doubled back on itself seven times. The result from ARE is markedly smoother and corresponds almost exactly to IMAGEBOT's actual manifold. Despite not having any meaning attached to the actions, ARE has clearly managed to learn a representation which captures the essential properties of the actual actions. Namely, that the two different actions are opposites of each other in terms of direction and have the same magnitude.

We can subtly change the actions which generate the data, making the backward action move twice as far as the forward one. Figure 6 demonstrates that ARE is capable of learning a manifold that can capture this property as well.

ARE can correctly handle periodic actions, such as rotation, as well. Figure 7 shows the first two dimensions of a manifold corresponding to a trajectory consisting of sixteen 'turn right' and eight 'turn left' actions. ARE essentially discovers the representation, $(\sin(\theta), \cos(\theta))$, as well as discovering that the actions are opposites and are periodic.

ARE continues to yield good results in the face of more complicated collections of transformations. ARE and SDE were both run with the more complex example shown in Figure 4. The resulting manifolds are displayed in Figure 8. SDE, as with the previous example, fails to generate a manifold in which the actions have a simple interpretation. Notice that again, ARE's manifold has a strong correspondence with IMAGEBOT's actual trajectory. ARE again captures the expected relationships between the 'forward' and 'back' actions, as well as the 'right' and 'left' actions. Even more impressive, the manifold has captured the 'forward'/'back' actions independence and orthogonality from the 'right'/'left' actions—despite the fact that none of this meaning was explicitly coded in the problem input.

In the final example, IMAGEBOT follows a variation of the 'A' trajectory. Instead of the actions 'left', 'right', 'forward' and 'backward' IMAGEBOT uses the actions 'zoom in', 'zoom out', 'forward' and 'backward'. In this case it is no longer true that the two pairs of actions—

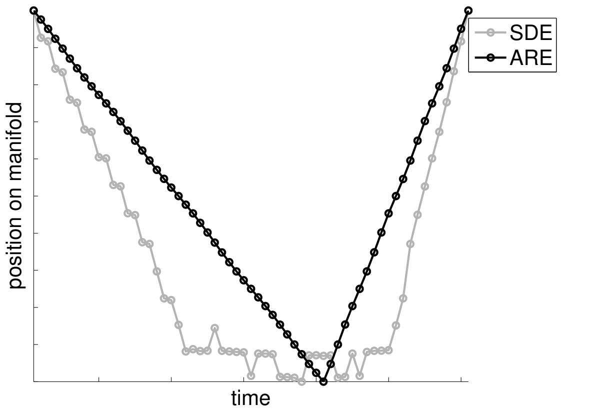

Figure 6. Manifolds from a trajectory similar to that from Figure 3 but with slightly different actions. Lines show the distance along the manifold over time.

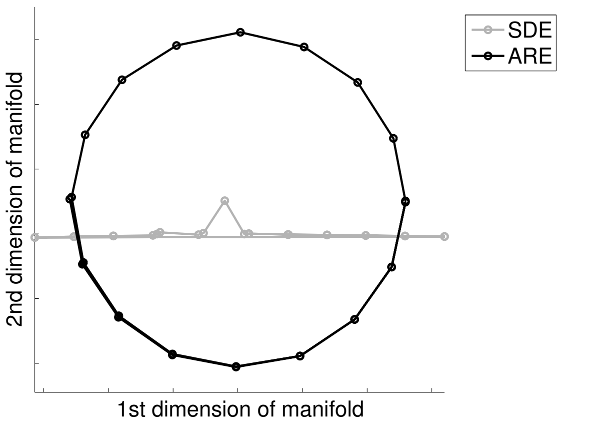

Figure 7. Manifolds learned on data generated by rotation.

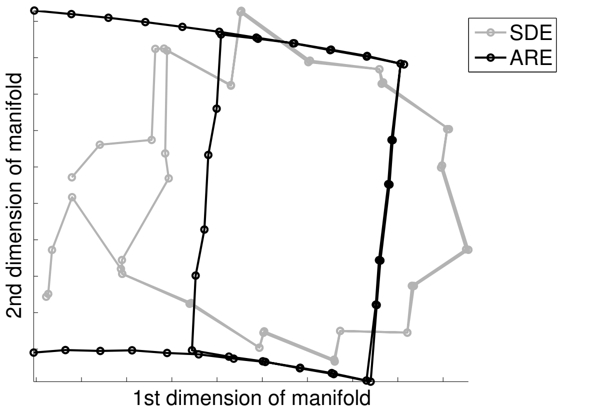

Figure 8. Manifolds corresponding to Figure 4.

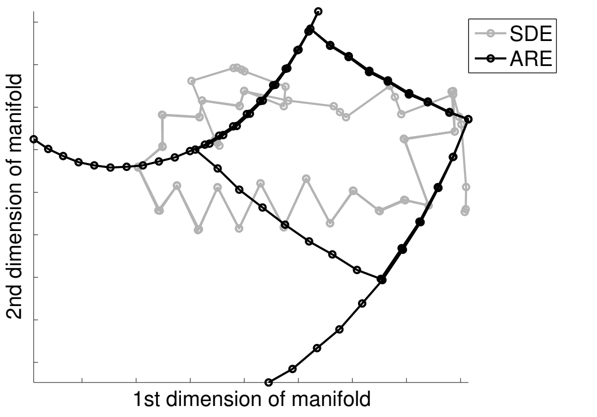

Figure 9. Manifolds learned on data generated with zoom actions.

'forward'/'backward' and 'zoom in'/'zoom out'—are independent, as the distance IMAGEBOT moves when implementing the first pair is dependent on IMAGEBOT's zoom level. Nonetheless, as Figure 9 demonstrates, ARE again learns a manifold that captures this relationship. The left leg of the 'A' corresponds to images gathered when IMAGEBOT was zoomed in, the right leg corresponds to images gathered when IMAGEBOT was zoomed out. Note that distance between consecutive points is less on the left leg than on the right. On this example ARE has successfully learned the radial relationship between the two sets of actions without knowing the relationship ahead of time.

Finally, ARE is flexible in choice of image-similarity function. All though not shown here, similar results can be obtained using distance metrics other than Euclidean distance.

4.3. Data Prediction

Section 3.3 introduced the task of data prediction and described how ARE could be used to solve this problem. We applied the data prediction algorithm to the four trajectories from the previous section. Since data prediction is a form of supervised learning, we wanted to only measure accuracy on queries outside of the training data. Queries of the form, "What training image would result from taking action a_1 from image x_1?", can easily be answered, (x_2), from the original data stream. Other queries, such as, "In Figure 4, what training image would result from taking action a_{11} from image x_{28}?", are not so easily answered. This query can only be answered by understanding that some actions are inverses of each other (*i.e.*, when the extracted representation appropriately respects the action labels).

We generated all possible image-action pairs resulting in an image in the training data, then excluded pairs of the form (x_i, a_i) as these are queries answered directly in the training data. The remaining queries were used to evaluate ARE's data prediction algorithm. For a comparison baseline, we also performed the same evaluation using SDE's learned manifolds. To be as fair as possible, we examined two prediction techniques for SDE. First, we used ARE's data-prediction algorithm with SDE's manifold. Second,

	Fig. 5	Fig. 6	Fig. 7	Fig. 8	Fig. 9
ARE	100.0%	100.0%	100.0%	100.0%	97.2%
SDE-d	10.2%	14.0%	28.0%	41.7%	25.0%
SDE-l	11.9%	29.8%	20.0%	39.6%	29.2%

Table 4. Prediction accuracy across the four trajectories.

we used regression on SDE's manifold to find the best linear transformation for each action, with the nearest training point to the transformed query point being the prediction.

Table 4 shows prediction accuracy for all three methods across all trajectories. "SDE-d" is SDE using ARE's data prediction and "SDE-l" is SDE using a linear transformation. ARE achieves near-perfect accuracy, quantitatively demonstrating ARE's ability to learn better manifolds.

5. Conclusion

We described a variant of dimensionality reduction where we are given action labels in addition to data points. Assuming these labels correspond to particular movements of a camera or other actuator, the goal becomes learning a manifold in which actions have meaningful representation.

Traditional dimensionality-reduction methods can be applied to this problem, but none of them make use of action labels. We therefore developed ARE: a semidefinite optimization for solving this problem inspired by SDE. ARE introduces two critical components. First, using the action labels to build a non-uniform neighborhood graph. Second, and more important, using the action labels to build constraints which force the learned manifold to be one in which the actions are represented as simple transformations.

We demonstrated the effectiveness of ARE in learning manifolds from the IMAGEBOT domain. We evaluated the results qualitatively and quantitatively. ARE was able to capture properties of the actions underlying the original data, despite the fact that none of these properties were explicitly coded in the input. Additionally, ARE greatly out-performed SDE in the provided data-prediction task.

As mentioned in the introduction, low-dimensional representations where actions can be defined as simple transformations are essential for many AI applications. Finding sequences of actions to achieve particular outcomes (planning) and maintaining a representation of one's location (localization) are two such tasks. We have demonstrated that ARE can *automatically* extract representations suited to these tasks from only a stream of experience. Although beyond the scope of this paper, we have successfully implemented planning (Wilkinson et al., 2005a) and localization (Wilkinson et al., 2005b) with ARE on small problems. Other AI tasks may also be able to benefit from ARE's ability to automatically extract good representations.

Acknowledgments

We would like to thank Michael Littman, Dan Lizotte, Dale Schuurmans, Finnegan Southey, and Tao Wang for discussions and insight. We would like to acknowledge Alberta Ingenuity Fund for their support of this research through the Alberta Ingenuity Centre for Machine Learning.

References

Borchers, B. (1999). CSDP, a C library for semidefinite programming. *Optimization Methods and Software*, *11*, 613–623.

Cox, T., & Cox, M. (2001). *Multidimensional scaling*. Boca Raton: Chapman Hall. 2nd edition.

Jolliffe, I. (1986). *Principal component analysis*. New York: Springer-Verlag.

Mika, S., Schölkopf, B., Smola, A., Müller, K.-R., Scholz, M., & Rätsch, G. (1999). Kernel PCA and de-noising in feature spaces. *Advances in Neural Information Processing Systems 11* (pp. 536–542). MIT Press.

Roweis, S., & Saul, L. (2000). Nonlinear dimensionality reduction by locally linear embedding. *Science*, *290*, 2323–2326.

Saul, L., & Roweis, S. (2003). Think globally, fit locally: Unsupervised learning of nonlinear manifolds. *Journal of Machine Learning Research*, *4*, 119–155.

Schölkopf, B., & Smola, A. (2002). *Learning with kernels*. Cambridge, Massachusetts: MIT Press.

Schölkopf, B., Smola, A., & Müller, K.-R. (1998). Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, *10*, 1299–1319.

Tenenbaum, J. (1998). Mapping a manifold of perceptual observations. *Advances in Neural Information Processing Systems 10* (pp. 682–687). MIT Press.

Tenenbaum, J., de Silva, V., & Langford, J. (2000). A global geometric framework for nonlinear dimensionality reduction. *Science*, *290*, 2319–2323.

Weinberger, K., & Saul, L. (2004a). Learning a kernel matrix for nonlinear dimensionality reduction. *Proceedings of the International Conference on Machine Learning* (pp. 839–846).

Weinberger, K., & Saul, L. (2004b). Unsupervised learning of image manifolds by semidefinite programing. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 988–995).

Wilkinson, D., Bowling, M., & Ghodsi, A. (2005a). Learning subjective representations for planning. *The 19th International Joint Conference on Artificial Intelligence*.

Wilkinson, D., Bowling, M., Ghodsi, A., & Milstein, A. (2005b). *Subjective localization with action respecting embedding* (Technical Report TR05-12). University of Alberta, Edmonton, Alberta, Canada.