Action Respecting Embedding

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

Ali Ghodsi AGHODSIB@CS.UWATERLOO.CA
School of Computer Science, University of Waterloo, Waterloo, ON, Canada

Dana Wilkinson D3WILKIN@CS.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 on dimensionality reduction can be solved with a semidefinite program. We evaluate the technique in a synthetic, robot-inspired domain, demonstrating both 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 robot's 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 image 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) can be described as simple transformations. This is why the robot's objective pose is such 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.

In this paper, we introduce a new algorithm called Action Respecting Embedding (ARE) to address our variation on traditional manifold learning. Specifically, we examine situations where the input data are given in sequence, along

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

with uninterpreted[1] action labels that are associated with adjacent pairs of data points. ARE finds a low-dimensional representation of the input data where the actions are simple transformations in the learned representation. 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 themselves 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 allow for a more objective evaluation. Therefore, along with traditional qualitative comparisons we also introduce the task of data prediction as a quantitative measure of the success of our learned representations.

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.

[1]By uninterpreted we mean that the action labels themselves have no implied meaning. We may refer to actions as being 'move left' or 'move right' while the algorithm sees the actions as simply 'Action 1' and 'Action 2'.

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.

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 one of the most popular techniques for dimensionality reduction, but its effectiveness is limited by its global linearity. Multidimensional scaling (MDS) (Cox & Cox, 2001), which is 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 (Mika et al., 1999; Scholkopf & 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. In order to motivate our algorithm, we provide a brief overview of Kernel PCA and SDE.

Kernel PCA is a non-linear generalization of PCA. In Kernel PCA, through the use of kernels, principle components can be computed efficiently in high-dimensional feature spaces that are related to the input space by some non-linear mapping. PCA finds an orthogonal transformation of the coordinate system in which we describe our 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. This is equivalent to performing PCA in the space produced by the non-linear mapping, where the low-dimensional latent structure is, hopefully, 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 in kernel PCA. 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:

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

By adding an objective function to maximize 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 construction of the neighborhood graph η_{ij}. This graph is 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.

SDE's strength is not only that its manifolds are comparable to other non-linear dimensionality reduction methods, but at its core is a simple semidefinite optimization. We will see in the next section that variants on dimensionality reduction, such as ours, can be solved by adding appropriate constraints into this optimization.

Algorithm: SDE

Construct neighbors, η, using k-nearest neighbors.

Maximize Tr(K) **subject to** $K \succeq 0$, $\sum_{ij} K_{ij} = 0$, **and**

$$\forall ij \quad \eta_{ij} > 0 \vee [\eta^T\eta]_{ij} > 0 \Rightarrow K_{ii} - 2K_{ij} + K_{jj} = ||x_i - x_j||^2$$

Run Kernel PCA with learned kernel, K.

Table 2. SDE Algorithm.

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 data-specific similarities can be employed.

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 modifications. 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-finding 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

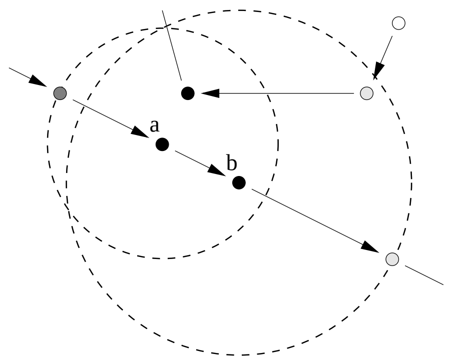

Figure 1. An example of the use of action labels to find non-uniform neighborhoods. The arrows show the points that are connected by an action. The circles show the resulting neighborhood for the points labeled 'a' and 'b' with $T = 1$. Black points are in both neighborhoods. White points in neither.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

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 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,

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

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

Notice that since our data points come from a sequence of actions, the resulting neighborhood graph ($T \geq 1$) must be fully connected. This satisfies the critical requirement that the semidefinite optimization be bounded (otherwise a solution would not exist).

3.2. Action Respecting Constraints

The second, and most 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 the manifold corresponds to the known data generator's own underlying degrees of freedom. Action labels, even with no 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 the labeled actions correspond to simple transformations in that space. In particular, we require all actions to be a simple rotation plus translation in the resulting low-dimensional representation.[2]

We can formalize this constraint by first observing rotation plus translation is exactly the space of *distance preserving* transformations. The transformation f is distance preserving and thus a rotation plus translation if and only if:

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

Let's 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 those data points must preserve their distance. Letting $\Phi(x_i)$ denote data point x_i in the the learned space, we know action a's transformation, f_a, must satisfy:

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

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

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

We want to pose this not as a constraint on distances, but rather as a constraint on inner products, *i.e.*, on the learned kernel matrix, K. We can square both sides of the equation and rewrite it in terms of K resulting in the following set of constraints:

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

We can add Constraint 5 into SDE's usual constraints to arrive at the optimization and algorithm shown in Table 3.

[2]These are the subset of linear transformations that don't involve any scaling component.

```
Algorithm: ARE
Construct neighbors, $\eta$, according to Equation 2.
Maximize $\text{Tr}(K)$ 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.

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 the stability of the solution. This was recommended by Weinberger and colleagues in their original SDE work (Weinberger & Saul, 2004b).

3.3. Data Prediction

As manifold learning is an unsupervised learning problem, evaluation of algorithms is usually qualitative. Data prediction, though, is a task we introduce, which (i) can be measured quantitatively and (ii) seeks to evaluate how well a low-dimensional representation has captured the actions. The problem of 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 a data point from the training set. Here we describe how ARE can be used to solve this task, and in Section 4 we present the results of this quantitative evaluation of the accuracy of ARE's predictions.

ARE learns an embedding where actions correspond to distance preserving operators. Recalling 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 appeared in our training set m times, then this gives us 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 determines $f_a(\Phi(x_i))$. In this case, it is a simple matter to find the point $\Phi(x_p)$ nearest to the constrained point $f_a(\Phi(x_i))$, and use x_p as our prediction. If the point is under-constrained ($m <= d$), then we select the index p according to:

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

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 can observe a small patch on that image and also take actions to move the observable patch around the larger image. This "image robot" provides us with an excellent domain in which we can test ARE, while having obvious connections to robotic applications.

For these experiments, IMAGEBOT is always viewing a 100 by 100 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 shows the image used for the experiments, while Figure 3 shows an example trajectory from IMAGEBOT

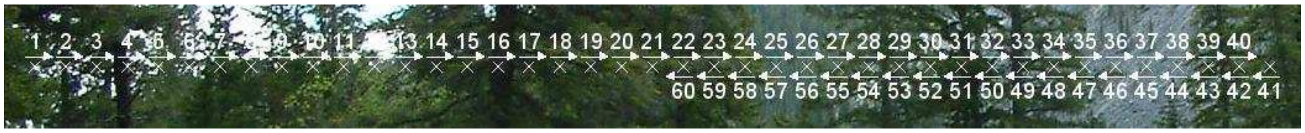
Figure 3. A sample 60-action trajectory from IMAGEBOT.

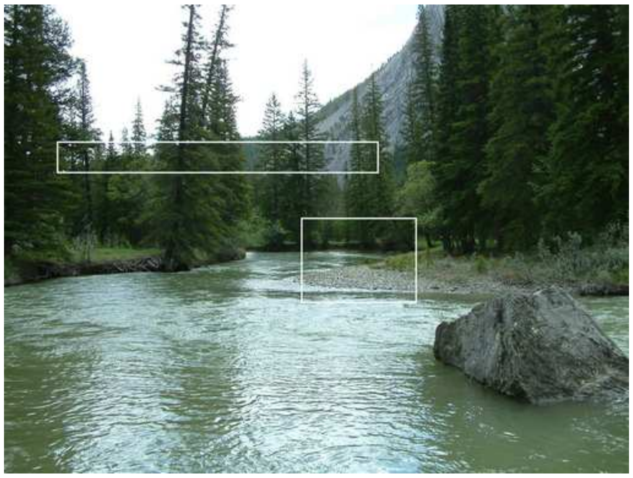
Figure 2. IMAGEBOT's world.

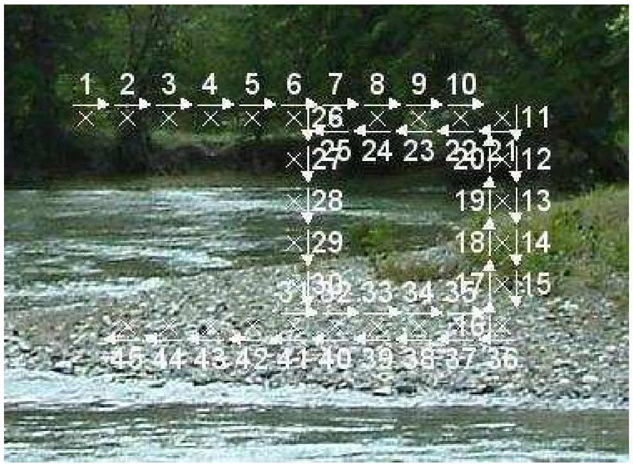
Figure 4. A more complicated 45-action trajectory from IMAGEBOT

(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 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 the manifold corresponding to the trajectory consisting of sixteen 'turn right' and eight 'turn left' actions. ARE again captures that the two actions are opposites, and that they are periodic.

ARE continues to yield good results in the face of more complicated collections of transformations. ARE and SDE were run with the more complex example shown in Figure 4, and 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. It again captures the expected relationships between the actions corresponding to forward and back, as well as the actions corresponding to right and left. Even more impressive, the manifold has captured that the forward/back actions are in-

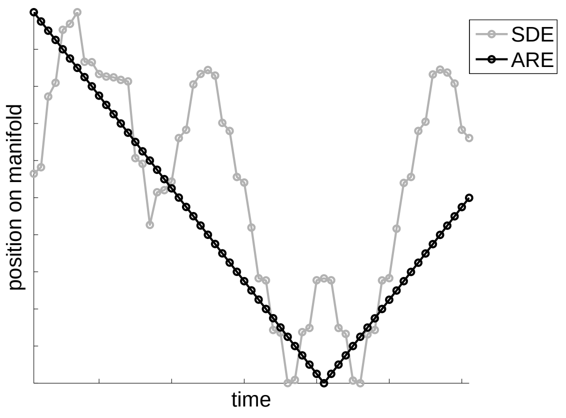

Figure 5. Manifolds from trajectory shown in Figure 3. Lines show the distance along the manifold over time.

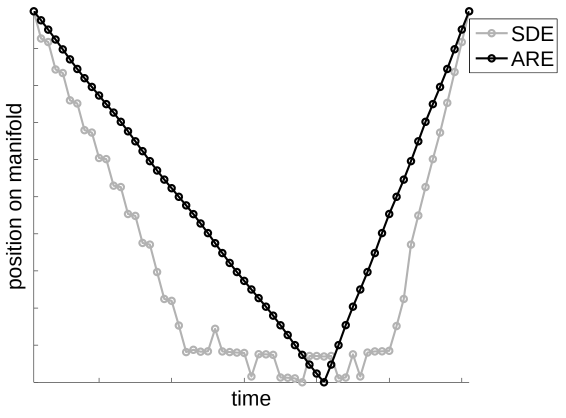

Figure 6. Manifolds from a trajectory similar to that from Figure 3 but with slightly different actions. Lines show the distance along the manifold over time.

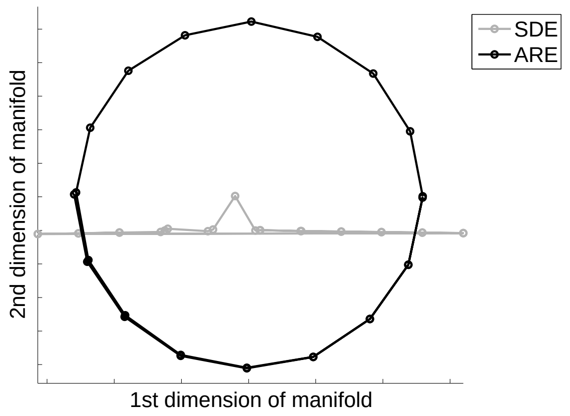

Figure 7. Manifolds learned on data generated with rotation actions.

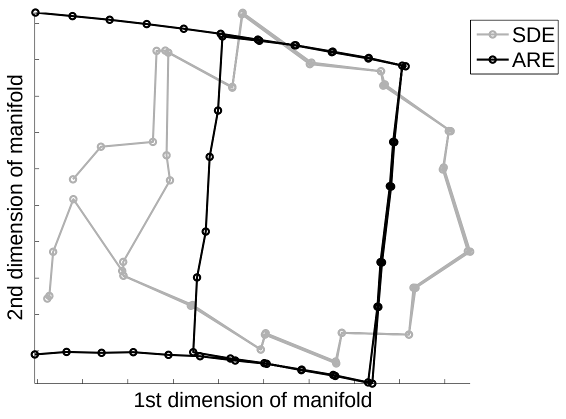

Figure 8. Manifolds corresponding to Figure 4

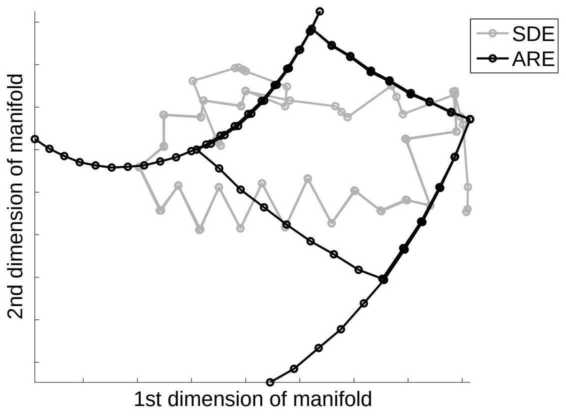

Figure 9. Manifolds learned on data generated with zoom actions.

dependent and orthogonal to 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—‘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. With this example ARE has successfully learned the radial relationship between the two sets of actions without requiring that the relationship be explicitly known ahead of time.

	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%	27.8%

Table 4. Prediction accuracy across the four trajectories.

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

4.3. Data Prediction

In Section 3.3 we introduced the task of data prediction, and described how ARE could be used to solve this problem. We applied our data prediction algorithm to the four trajectories from the previous section. As data prediction is a form of supervised learning, we want to be careful to measure accuracy only 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 be answered (x_2) easily from ARE's input data. 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.*, that the manifold extracts a representation that appropriately respects the action labels.

We generated all possible image-action pairs whose result is an image in the training data. We then excluded all 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 manifolds extracted with SDE. To be as accommodating as possible, we examined two prediction techniques for SDE. First, we used ARE's data prediction algorithm with SDE's learned manifold. Second, we used regression on SDE's representation 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 the prediction accuracy for all three methods across the five trajectories. In the table, "SDE-d" refers to SDE using ARE's data prediction, and "SDE-l" refers to using a linear transformation. ARE achieves near-perfect accuracy, quantitatively demonstrating ARE's ability to learn better manifolds.

5. Conclusion

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

Although traditional dimensionality reduction methods can be applied to this problem, none of them make effective use of the action labels. We therefore developed ARE—a semidefinite optimization for solving this problem inspired by SDE. ARE introduces two new critical components. First, ARE uses the action labels to build a non-uniform neighborhood graph. Second, and more important, we can use these action labels to build constraints which force the learned manifold to be one in which the actions can be 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 the foundation for many AI applications. Finding a sequence of actions to achieve a particular outcome (*i.e.*, planning) and maintaining a representation of one's location (*i.e.*, localization) are two such tasks. We have demonstrated that ARE can *automatically* extract representations especially suited to these tasks from only a stream of experience. Although beyond the scope of this paper, we have successfully implemented planning and localization with ARE on small problems. We expect that other standard AI tasks may 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 their discussions and insight. We also 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*. Chapman Hall. 2nd edition.

Jolliffe, I. (1986). *Principal component analysis*. 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).

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.

Scholkopf, B., & Smola, A. (2002). *Learning with kernels*. MIT Press.

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

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).