# Out-of-Sample Extrapolation of Learned Manifolds

Tat-Jun Chin and David Suter, *Senior Member*, *IEEE*

**Abstract**—We investigate the problem of extrapolating the embedding of a manifold learned from finite samples to novel out-of-sample data. We concentrate on the manifold learning method called *Maximum Variance Unfolding* (MVU), for which the extrapolation problem is still largely unsolved. Taking the perspective of MVU learning being equivalent to Kernel Principal Component Analysis (KPCA), our problem reduces to extending a kernel matrix generated from an unknown kernel function to novel points. Leveraging on previous developments, we propose a novel solution, which involves approximating the kernel eigenfunction by using Gaussian basis functions. We also show how the width of the Gaussian can be tuned to achieve extrapolation. Experimental results, which demonstrate the effectiveness of the proposed approach, are also included.

**Index Terms**—Manifold learning, out-of-sample extrapolation, Maximum Variance Unfolding.

## 1 Introduction

Given a finite set of input data densely sampled from a smooth manifold in the input space, manifold learning methods seek the embedding of the data in a lower dimensional space in a manner that preserves the structure of the underlying manifold. Manifold learning has been attracting enormous attention as a promising alternative dimensionality reduction method, particularly in the field of computer vision where it has been realized that linear methods such as Principal Component Analysis (PCA) are unsuitable for analyzing complex nonlinear distributions of images. For a recent survey of manifold learning methods, refer to [1] and [2].

In this paper, we address the issue of extrapolating learned manifolds to out-of-sample data; that is, given a previously unseen point, what is the corresponding embedding of the point on the learned manifold? Realistically, we cannot only consider novel points, which are confined to be located within the empirically learned manifold. We must also consider extrapolating *slightly* off-manifold points, which arise due to perturbations caused by noise in the data acquisition or input preprocessing stages. Here, we concentrate on the manifold learning method called *Maximum Variance Unfolding* (MVU) [3], also known as *Semi-Definite Embedding* (SDE), for which the extrapolation problem is still largely unsolved [3]. MVU belongs to a class of methods called *spectral embedding methods*, where the embedding coordinates are derived from the top or bottom eigenvectors of specially constructed matrices [2].

### 1.1 Previous Work

Bengio et al. [4], [5] showed that there is a direct relation between spectral embedding methods and Kernel PCA (KPCA) [6] and how both are special cases of learning the principal eigenfunctions of an operator defined from a kernel and the unknown data generating density. For the spectral embedding methods of Locally Linear Embedding (LLE), Laplacian Eigenmaps, and Isomap (all described in [2]), Bengio et al. [4], [5] defined *data-dependent kernels* that can be applied outside the training set for extrapolation. The data-dependent kernels can implicitly be defined by the specific spectral embedding algorithms. This generalizes methods such as Landmark Isomap [7]. Landmark Isomap performs Isomap on a subset of the training points (landmarks) and predicts the embedding of the other points by using a data-dependent kernel, thus achieving extrapolation *for Isomap*.[1] However, for MVU, it is unclear how a data-dependent kernel can be constructed, since semidefinite programming is conducted to optimize the kernel matrix to learn the embedding. The algebraic form of the data-dependent kernel is unknown.

A recent work [9] examined the more general problem of extending a kernel matrix generated from an unknown kernel function to novel points. Among other contributions, a framework was proposed for the extrapolation of LLE to unseen data. Integral to their solution is a matrix approximation theorem, which determines the solution to the problem (see Lemma 1 in [9]):

$$\arg\min_{\mathbf{Q}\succeq 0} \|\mathbf{K}-\mathbf{K}_{\mathbf{Q}}\|_p. \tag{1}$$

The symbol $\mathbf{Q}\succeq 0$ means that $\mathbf{Q}$ is positive semidefinite, and $\|\cdot\|_p$ is the matrix $p$-norm (their solution is independent of $p$). Here, $\mathbf{K}$ is the kernel matrix for which a kernel

[1] Despite similar naming, Landmark SDE/MVU [8] does not provide extrapolation for MVU. Rather, it aims at reducing the training time of MVU only.

- T.-J. Chin is with the Institute for Infocomm Research, Agency for Science, Technology, and Research, 21 Heng Mui Keng Terrace, Singapore 119613. E-mail: tjchin@i2r.a-star.edu.sg.
- D. Suter is with the Department of Electrical and Computer Systems Engineering, Monash University, Clayton, Victoria 3168, Australia. E-mail: d.suter@eng.monash.edu.au.

*Manuscript received 7 Nov. 2006; revised 23 Apr. 2007; accepted 25 Sept. 2007; published online 12 Oct. 2007.*
*Recommended for acceptance by Z. Ghahramani.*
*For information on obtaining reprints of this article, please send e-mail to: tpami@computer.org, and reference IEEECS Log Number TPAMI-0790-1106.*
*Digital Object Identifier no. 10.1109/TPAMI.2007.70813.*

function is sought, whereas $\mathbf{K_Q}$ is the kernel matrix from a kernel function of the following form:

$$k_{\mathbf{Q}}(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x})^T \mathbf{Q}\, \phi(\mathbf{z}), \tag{2}$$

where $\phi : \mathbb{R}^m \longrightarrow \mathbb{R}^n$ is an intermediate feature map from the input space $\mathbb{R}^m$ to the feature space $\mathbb{R}^n$. Note that the eventual mapping of $\mathbb{R}^m$ is to an RKHS $\mathcal{F}$ induced by the kernel function $k_{\mathbf{Q}}(\cdot, \cdot)$. For the case of LLE, it is shown in [9] how the kernel matrix $\mathbf{K}$ and intermediate mapping $\phi(\mathbf{x})$ can be defined by the LLE algorithm, after which the optimal matrix $\mathbf{Q}$ can be computed so that $k_{\mathbf{Q}}(\cdot, \cdot)$ can freely be applied on novel data. In the case of MVU, since $\mathbf{K}$ is obtained via a semidefinite optimization, it is unclear what form $\phi$ must take before a $\mathbf{Q}$ can be determined.

A class of general solution would be to learn a mapping either directly, for example, [10] uses a neural network, or by attempting to invert a learned mapping from embedding to the input space, for example, [11], which claims to invert a Generalized Radial Basis Function (GRBF) mapping. The first approach is problematic in that it is a large learning exercise and, if conducted using MLPs (as in [10]), can consume huge computational resources. Even worse, it is difficult to decide the number of hidden units. This is especially true for high-dimensional inputs. Note that cross validation is infeasible, since, given one network setting, the training effort is prohibitively expensive (see Section 3). The second approach is problematic, because it really requires the solution of a set of nonlinear equations (see [11]). Although Gong et al. [10] introduce a "linear approximation" solution, they provide no justification or guarantee that this is a good approximation. Indeed, our results in Section 3 seem to show that it is not, in general, a good approximation.

There also exists a simple nonparametric approach that was proposed in [12] for the extrapolation of LLE. This could potentially be applied in any manifold learning methods, which involve preserving local geometry as an objective. For a novel point, the method in [12] first computes its linear reconstruction weights from its neighbors. The extrapolation coordinates are then obtained by constructing a new point in the embedding space using the embedded coordinates of the neighbors according to the same weight coefficients. However, as pointed out in [12], a nonparametric mapping can discontinuously change as query points move between different neighborhoods. Furthermore, for novel points that are off manifold, the locally linear condition is not satisfied anymore; hence, it is uncertain how a nonparametric mapping would behave in such conditions. Based on a completely different idea, our method uses a set of basis functions, which are centered at the training data to approximate the underlying eigenfunction. Although this is also nonparametric, by controlling the width of the basis functions, we can achieve a degree of smoothing to prevent discontinuous mappings. Furthermore, our method is intentionally designed to handle off-manifold points by optimizing the basis function parameters.

### 1.2 Contributions

We propose a method that allows the extrapolation of manifolds learned via MVU to novel out-of-sample points. To the best of our knowledge, this has not been attempted before. We also provide experimental results, which show the effectiveness of our solution.

## 2 Extrapolation of Learned Manifolds

We set the background by briefly describing MVU [3], [2] and the Nyström formula [13], [14] for out-of-sample extrapolation [4], [5]. First, we define some commonly used symbols. The set of training vectors is given by $\mathcal{X} = \{\mathbf{x}_1, \cdots, \mathbf{x}_n\} \in \mathbb{R}^m$, and manifold learning computes the corresponding embedding $\mathcal{E} = \{\mathbf{e}_1, \cdots, \mathbf{e}_n\} \in \mathbb{R}^r$ such that the manifold structure sampled by $\mathcal{X}$ is preserved. Typically, $r \ll m$, and the value of $r$ reflects the intrinsic dimensionality of the underlying manifold. For spectral embedding methods, $\mathcal{E}$ is obtained from the top or bottom eigenvectors of specially constructed matrices represented by $\mathbf{K}$ in this paper.

### 2.1 Maximum Variance Unfolding

Given $\mathcal{X}$, the underlying principle of MVU is to "unfold" the data set by pulling the vectors as far apart as possible while observing local distance and angle constraints so that the eventual transformation from input vectors $\{\mathbf{x}_1, \cdots, \mathbf{x}_n\}$ to output vectors $\{\boldsymbol{\psi}_1, \cdots, \boldsymbol{\psi}_n\}$ locally looks like a rotation plus translation [2]. If MVU is interpreted from the KPCA perspective, then the $\{\psi_1, \cdots, \psi_n\}$ inhabit an RKHS induced by a kernel function $k(\cdot, \cdot)$. The Gram matrix $\tilde{\mathbf{K}}$ of $\{\psi_1, \cdots, \psi_n\}$ is the kernel matrix associated with $k(\cdot, \cdot)$, where $\tilde{\mathbf{K}}_{ij} = \psi_i \cdot \psi_j = k(x_i, x_j)$. Let $\mathbf{K}$ be the *normalized* (or centered) version of $\tilde{\mathbf{K}}$. This means that each entry of $\mathbf{K}$ contains the dot-product between the centered output vectors in a kernel-induced RKHS, that is, $\mathbf{K}_{ij} = \bar{\psi}_i \cdot \bar{\psi}_j$, where $\bar{\psi}_i = \psi_i - \frac{1}{n}\sum_{k=1}^n \psi_k$. The eigenvectors of $\mathbf{K}$ also provide the desired embedding $\mathcal{E} = \{\mathbf{e}_1, \cdots, \mathbf{e}_n\}$ of $\mathcal{X} = \{\mathbf{x}_1, \cdots, \mathbf{x}_n\}$. MVU uses numerical semidefinite programming (optimization over positive semidefinite matrices) to directly find the best $\mathbf{K}$. Thus, we do not have the actual algebraic form of the underlying kernel function $k(\mathbf{x}_i, \mathbf{x}_j) = \tilde{\mathbf{K}}_{ij}$.

### 2.2 The Nyström Formula for Out-of-Sample Extrapolation

Spectral embedding methods will eventually arrive at a matrix $\mathbf{K}$ (not necessarily positive semidefinite), for which the eigenvectors provide the desired embedding. For LLE, Isomap, and Laplacian Eigenmap, $\mathbf{K}$ is constructed according to their specific algorithmic procedures that take the training data into account. Bengio et al. [5] showed that for each of these methods, a *data-dependent* kernel function $k_n(\cdot, \cdot)$ can algebraically be formulated, that is, $\mathbf{K}_{ij} = k_n(\mathbf{x}_i, \mathbf{x}_j)$. Note that $k_n(\mathbf{x}_i, \mathbf{x}_j)$ depends not only on $\mathbf{x}_i$ and $\mathbf{x}_j$, but also on the whole training set $\mathcal{X}$. It is emphasized that for spectral embedding methods, $\mathbf{K}$ is the normalized version of an underlying kernel matrix $\tilde{\mathbf{K}}$. Equivalently, $k_n(\cdot, \cdot)$ is the normalized version of the underlying kernel function $k(\cdot, \cdot)$, which gives rise to $\tilde{\mathbf{K}}$, that is, $\tilde{\mathbf{K}}_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$. Let $\nu_{ip}$ be the $i$th coordinate of the $p$th eigenvector of $\mathbf{K}$ with eigenvalue $l_p$. Given that the form of the data-dependent kernel function $k_n(\cdot, \cdot)$ is known, the Nyström formula [5], [14] is evaluated as

$$f_{p,n}(\mathbf{x}) = \frac{1}{l_p}\sum_{i=1}^{n} \boldsymbol{\nu}_{ip} k_n(\mathbf{x}, \mathbf{x}_i), \tag{3}$$

where $f_{p,n}$ is the Nyström estimator for the $p\text{th}$ eigenvector of $\mathbf{K}$ with $n$ samples. Note the similarity with the KPCA projection formula [6]. Bengio et al. [5] also showed that $f_{p,n}$ estimates the $p\text{th}$ eigenfunction of a linear operator defined using the true underlying data distribution. This formula was used to estimate extensions of eigenvectors in Gaussian Process regression [15] and to speed up kernel machines [14]. In the manifold learning context, $f_{p,n}(\mathbf{x})$ predicts the (scaled) embedding of a new point $\mathbf{x}$; that is, the extrapolation of $\mathbf{x}$ for the $p\text{th}$ embedding coordinate is simply the extension of the $p\text{th}$ eigenvector for new data $\mathbf{x}$ of the normalized kernel matrix $\mathbf{K}$. Results in [4] and [5] demonstrated the accuracy for LLE and Isomap extrapolation.

### 2.3 Out-of-Sample Extrapolation for Maximum Variance Unfolding

For MVU, however, $\mathbf{K}$ is obtained through semidefinite programming. Using the Nyström formula for out-of-sample extrapolation is problematic, since it is unknown how we should define a $k_n(\cdot,\cdot)$ that is suitable to MVU. To this end, we approximate the kernel eigenfunction $f_{p,n}(\mathbf{x})$ with basis functions $r(\cdot,\cdot)$, as done in [15] to generalize Gaussian Process covariance matrices learned via Expectation-Maximization to novel inputs. Via the Nyström formula, the *approximate* $p\text{th}$ scaled eigenfunction of $\mathbf{K}$ by using the basis functions is

$$f_{p,n}(\mathbf{x}) = \sum_{i=1}^{n} b_{pi} r(\mathbf{x}, \mathbf{x}_i), \tag{4}$$

with weights $\mathbf{b}_p = [b_{p1}, \cdots, b_{pn}]^T = (\mathbf{R} + \lambda\mathbf{I})^{-1}\boldsymbol{\nu}_p$, where $\boldsymbol{\nu}_p$ is the $p\text{th}$ eigenvector of $\mathbf{K}$. Matrix $\mathbf{R}$ indicates the Gram matrix for $r(\cdot,\cdot)$ on all the training vectors, and the regularization term $\lambda\mathbf{I}$ is included to stabilize its inverse. The corresponding *approximate* data-dependent kernel function [15] to substitute for the unknown underlying $k_n(\cdot,\cdot)$ is

$$\begin{aligned}\tilde{k}_n(\mathbf{x}, \mathbf{z}) &= \sum_{p=1}^{r} l_p f_{p,n}(\mathbf{x}) f_{p,n}(\mathbf{z}) \\ &= \mathbf{r}(\mathbf{x})^T (\mathbf{R} + \lambda\mathbf{I})^{-1} \mathbf{K} (\mathbf{R} + \lambda\mathbf{I})^{-1} \mathbf{r}(\mathbf{z}),\end{aligned} \tag{5}$$

with $\mathbf{r}(\mathbf{x}) = [r(\mathbf{x}_1, \mathbf{x}), \cdots, r(\mathbf{x}_n, \mathbf{x})]^T$. One can easily see that if $\lambda$ is sufficiently small, the resulting kernel matrix from evaluating $\tilde{k}_n(\cdot,\cdot)$ on the training set is equal to $\mathbf{K}$ (under the assumption that the $(r+1)\text{th}$ and above eigenvalues of $\mathbf{K}$ are numerically insignificant).

Recall that for MVU [8], [3], the $p\text{th}$ embedding coordinate of the $i\text{th}$ training point $\mathbf{x}_i$ is obtained as $\sqrt{l_p}\nu_{ip}$. Therefore, (4) must be rescaled by a factor of $\sqrt{l_p}$ as well for the result to function correctly as the MVU extrapolation of the $p\text{th}$ embedding coordinate for a new point $\mathbf{x}$. In addition, from the viewpoint of KPCA, we can project $\mathbf{x}$ onto the $p\text{th}$ kernel principal component of $\boldsymbol{\mathcal{X}}$ with kernel matrix $\mathbf{K}$ as

$$w_{p,n}(\mathbf{x}) = \frac{1}{\sqrt{l_p}} \sum_{i=1}^{n} \boldsymbol{\nu}_{ip} \tilde{k}_n(\mathbf{x}, \mathbf{x}_i) = \mathbf{P}_p \mathbf{r}(\mathbf{x}), \tag{6}$$

where vector $\mathbf{P}_p := l_p^{-\frac{1}{2}} \boldsymbol{\nu}_p^T \mathbf{R} (\mathbf{R} + \lambda\mathbf{I})^{-1} \mathbf{K} (\mathbf{R} + \lambda\mathbf{I})^{-1}$. Note that $w_{p,n}(\mathbf{x})$ is equal to $\sqrt{l_p} f_{p,n}(\mathbf{x})$, which is precisely what we require for MVU extrapolation. This arises due to the equivalence of the Nyström formula to the KPCA projection formula, as observed previously in [4], [5], and [14].

To ensure that new points are centered with regard to the mean of the training data when being subjected to the approximate data-dependent kernel $\tilde{k}_n(\cdot,\cdot)$, we normalize $r(\cdot,\cdot)$ to its data-dependent form before plugging it into $\tilde{k}_n(\cdot,\cdot)$:

$$\begin{aligned}r_n(\mathbf{x}, \mathbf{z}) = {} & r(\mathbf{x}, \mathbf{z}) - E_{\mathbf{x}'}[r(\mathbf{x}', \mathbf{z})] - E_{\mathbf{z}'}[r(\mathbf{x}, \mathbf{z}')] \\ & + E_{\mathbf{x}'}[E_{\mathbf{z}'}[r(\mathbf{x}', \mathbf{z}')]],\end{aligned} \tag{7}$$

where $E_{\mathbf{x}}[f(\mathbf{x})]$ represents the expected value of $f(\mathbf{x})$ taken over the underlying distribution density of the training data. By introducing this data dependency in $\tilde{k}_n(\cdot,\cdot)$ of (6), we perform a data-centering step that is analogous to the one used in KPCA before projecting a new test point. The expectation $E_{\mathbf{x}}[f(\mathbf{x})]$ is approximated by averaging over the empirical distribution:

$$E_{\mathbf{x}}[f(\mathbf{x})] = \frac{1}{n} \sum_{i=1}^{n} f(\mathbf{x}_i). \tag{8}$$

Equations (4), (5), and (6) are accordingly modified by substituting $r(\cdot,\cdot)$ with $r_n(\cdot,\cdot)$ and computing $\mathbf{R}_{ij} = r_n(\mathbf{x}_i, \mathbf{x}_j)$. What is left is determining the form of $r(\cdot,\cdot)$ and its parameters, as well as the appropriate value of $\lambda$. We propose using the Gaussian basis function

$$r(\mathbf{x}, \mathbf{z}) = \exp(-\|\mathbf{x} - \mathbf{z}\|^2/\sigma), \tag{9}$$

for defining $\tilde{k}_n(\cdot,\cdot)$. This choice is motivated by the good results that we have obtained for the manifold extrapolation in Section 3. To select the optimal values of $\sigma$ and $\lambda$ for a particular learned manifold with centered kernel matrix $\mathbf{K}$, we tune $\tilde{k}_n(\cdot,\cdot)$ by using a set of artificially generated on-manifold and off-manifold points with their corresponding embedding.

#### 2.3.1 *Generating On-Manifold and Off-Manifold Tuning Samples*

To generate novel on-manifold samples, we apply the method in [11]. GRBF networks are fitted on $\{\mathbf{e}_1, \cdots, \mathbf{e}_n\}$ to create a mapping $f_{GRBF}$ from the embedding space $\mathbb{R}^r$ to the input space $\mathbb{R}^m$. We interpolate within $\{\mathbf{e}_1, \cdots, \mathbf{e}_n\}$ by finding midpoints between $t$ pairs of points (either randomly chosen or using nearest neighbor pairs only) to obtain $t$ new coordinates $\{\mathbf{e}_1^{on}, \cdots, \mathbf{e}_t^{on}\}$. These are then mapped to the input space to obtain $\{\mathbf{x}_1^{on}, \cdots, \mathbf{x}_t^{on}\}$, that is, $\mathbf{x}_i^{on} = f_{GRBF}(\mathbf{e}_i^{on})$. Provided that the GRBF networks are properly fitted with the right parameters (for example, the number of centroids, type of basic function, and associated parameters), $f_{GRBF}$ is capable of producing very accurate on-manifold samples. This is illustrated in Section 3. Refer to [11] for details on fitting GRBF networks on the embedding.

For off-manifold samples, we locally apply standard PCA in the training set. The intrinsic dimensionality $r$ of the underlying manifold has been recovered from MVU. For each $\mathbf{x}_i$, a local patch of $k$-nearest neighbors ($k$-NN's; we set $k = 20$) is computed, and with these, a PCA is conducted, with $\mathbf{x}_i$ being the origin. The off-manifold sample $\mathbf{x}_i^{off}$ is obtained by biasing $\mathbf{x}_i$ *slightly* away from the manifold along the direction of the $(r+1)$th principal component. During tuning, we require that the embedding of $\mathbf{x}_i^{off}$ be extrapolated close to, if not exactly as, $\mathbf{e}_i$. With this, we obtain the off-manifold points $\{\mathbf{x}_1^{off}, \cdots, \mathbf{x}_n^{of!f}\}$ with their corresponding embedding $\{\mathbf{e}_1^{off}, \cdots, \mathbf{e}_n^{off}\} = \{\mathbf{e}_1, \cdots, \mathbf{e}_n\}$. The focus here is on learning a robust extrapolation function, which can deal with noisy data within *close proximity* of the learned manifold. Hence, the direction of bias for $\mathbf{x}_i^{off}$ does not need to exactly be orthogonal to the manifold surface at $\mathbf{x}_i$.

### 2.3.2 Tuning a Data-Dependent Kernel Function

Let $\boldsymbol{\mathcal{X}}^{tune} = \{\mathbf{x}_1^{on}, \cdots, \mathbf{x}_t^{on}\} \bigcup \{\mathbf{x}_1^{off}, \cdots, \mathbf{x}_n^{off}\}$ contain the generated on-manifold and off-manifold samples with their ground truth embedding $\boldsymbol{\mathcal{E}}^{tune} = \{\mathbf{e}_1^{on}, \cdots, \mathbf{e}_t^{on}\} \bigcup \{\mathbf{e}_1^{off}, \cdots, \mathbf{e}_n^{off}\}$. To obtain the optimal parameters of our approximate data-dependent kernel $\tilde{k}_n(\cdot,\cdot)$, we solve the problem as follows:

$$\{\sigma^*, \lambda^*\} = \arg\min_{\sigma,\lambda} \frac{1}{t+n} \sum_{i=1}^{t+n} \|\mathbf{P}_{1:r}\mathbf{r_n}(\mathbf{s}_i) - \epsilon_i\|^2, \tag{10}$$

where $\mathbf{s}_i \in \boldsymbol{\mathcal{X}}^{tune}$, $\epsilon_i \in \boldsymbol{\mathcal{E}}^{tune}$, $\mathbf{P}_{1:r} := \left[\mathbf{P}_1{}^T \cdots \mathbf{P}_r{}^T\right]^T$, and $\mathbf{r_n}(\mathbf{x}) := [r_n(\mathbf{x}_1, \mathbf{x}), \cdots, r_n(\mathbf{x}_n, \mathbf{x})]^T$. See (6). Note that $\mathbf{P}_{1:r}$ is constructed using the data-dependent form of $r(\cdot,\cdot)$. Multiplying $\mathbf{P}_{1:r}$ with $\mathbf{r_n}(\mathbf{s}_i)$ simply returns the $r$-dimensional embedding of $\mathbf{s}_i$ on the manifold that was learned from $\boldsymbol{\mathcal{X}}$ by using MVU. To solve (10) in our experiments, we fix $\lambda$ to a small value (0.01) and optimize over $\sigma$. We resorted to hand-tuning (10) to achieve this goal. This approach is practically tractable, since there is only one parameter.

### 2.3.3 Computational Complexity

We focus on the extrapolation step $\mathbf{P}_{1:r}\mathbf{r_n}(\mathbf{s}_i)$ as this is sufficient to give us a picture on the effort required to solve (10); that is, at each tuning iteration, the overall effort is just the accumulation of $t+n$ times of $\mathbf{P}_{1:r}\mathbf{r_n}(\mathbf{s}_i)$ and also of computing, squaring, and summing up the residuals. Since $\lambda$ is fixed and $\mathbf{P}_{1:r}$ is independent of $\mathbf{s}_i$, at each tuning iteration (that is, for a fixed $\sigma$), matrix $\mathbf{P}_{1:r}$ can be computed once, cached, and reused for all $\mathbf{s}_i$'s. Hence, the effort required to produce $\mathbf{P}_{1:r}$ can be ignored. Similarly, the third and fourth terms on the right-hand side of (7) are independent of $\mathbf{s}_i$, that is, variable $\mathbf{z}$ in (7). Thus, at each tuning iteration, the corresponding terms in $\mathbf{r_n}(\mathbf{s}_i)$ can be stored and reused for all $\mathbf{s}_i$'s. Since we are interested in the complexity of (10) with regard to the number of training data $n$ only and evaluating function $r(\cdot,\cdot)$ scales according only to the length of the input vector, we denote by $c_r$ the computational effort required to evaluate the function $r(\cdot,\cdot)$ once. Given $n$ training vectors, evaluating $\mathbf{r_n}(\mathbf{s}_i)$ once will incur $n \cdot c_r + 4n$ computations,[2] which is accumulated from the following procedures:

1. $n$ evaluations of $r(\mathbf{x}_j, \mathbf{s}_i)$, where $\mathbf{x}_j \in X$, and $1 \le j \le n$. Each $r(\mathbf{x}_j, \mathbf{s}_i)$ are then cached.
2. $(n-1)$ additions and one division to produce $\frac{1}{n}\sum_{j=1}^{n} r(\mathbf{x}_j, \mathbf{s}_i)$ as the second term in (7) with reference to (8). This value is also cached.
3. For each of the $n$ entries in $\mathbf{r_n}(\mathbf{s}_i) = [r_n(\mathbf{x}_1, \mathbf{s}_i), \cdots, r_n(\mathbf{x}_n, \mathbf{s}_i)]^T$, only two subtractions and one addition are needed, since all terms of (7) (including the third and fourth, as mentioned before) were cached from previous computations.

Multiplying $\mathbf{P}_{1:r}$ with $\mathbf{r_n}(\mathbf{s}_i)$ will cost $r(2n-1)$ computations. This gives an overall computational effort of $n(c_r + 2r + 4) - r$, thus evaluating that $\mathbf{P}_{1:r}\mathbf{r_n}(\mathbf{s}_i)$ scales with $\mathcal{O}(n)$.

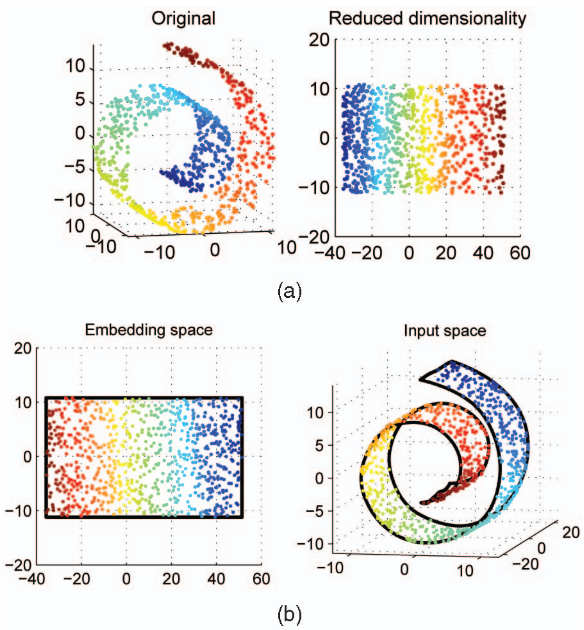


Fig. 1. (a) An 800-point Swiss-roll data set with its MVU embedding. (b) A bounding box, which encapsulates the embedding coordinate range, and novel randomly interpolated points (in various colors) are accurately mapped via GRBF onto the Swiss-roll manifold in the input space. Note that (b) does not involve the training points or their embedding coordinates at all.

## 3 Experimental Results

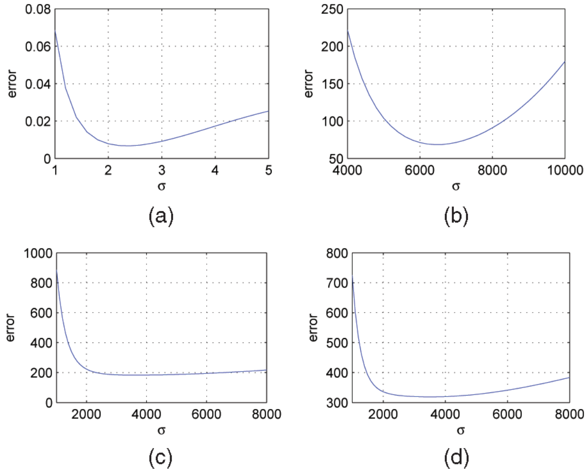
### 3.1 Synthetic Data

We first consider the synthetic Swiss-roll data set. This refers to a set of vectors uniformly sampled from a 2D manifold in the shape of a Swiss roll in 3D space. Data sets with different numbers of samples, that is, $[500, 600, \cdots, 1{,}700]$, were generated[3] and fed to the MVU algorithm by using 6-neighborhood [3]. MVU returned a 2D embedding for each data set (see Fig. 1a).

We applied four methods to produce a function for manifold extrapolation: the proposed method, inverse GRBF [11], MLP [10], and the $k$-NN reconstruction method proposed for LLE in [12]. For each Swiss-roll embedding, the following steps were performed to train the first three methods (the $k$-NN method does not require training):

[2] We regard the basic arithmetic operations of addition, subtraction, multiplication, and division to consume the same amount of computational effort.

[3] Using the code available at http://www.cs.utoronto.ca/~roweis/lle/code/swissroll.m.



(a) (b)

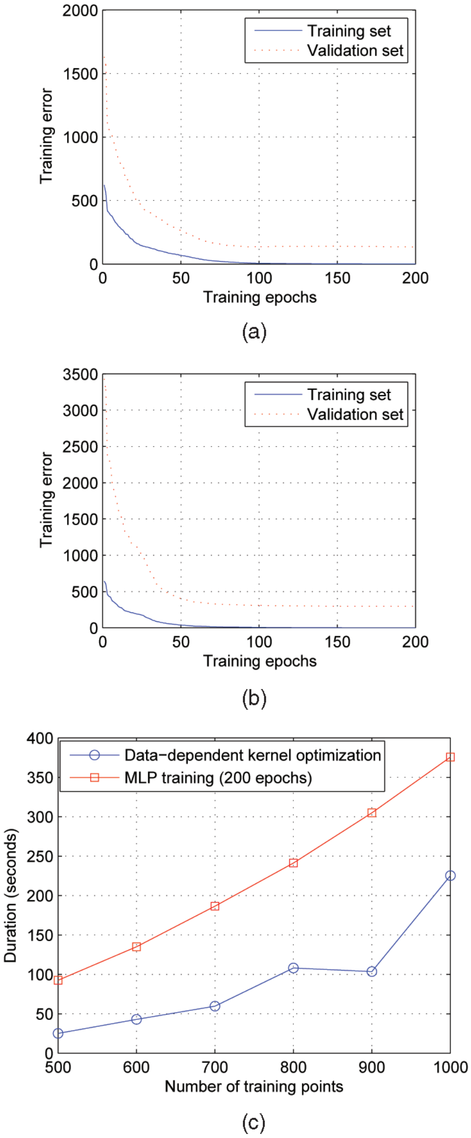
(c) (d)

Fig. 2. Tuning data-dependent kernels. In all cases, $\lambda$ is fixed to 0.01. For (a), the curve for the 800-sample data set is shown. Note that for all data sets, the error plotted is given by (10). (a) Swiss-roll data set. (b) Teapots data set. (c) Face data set. (d) Soft-toy data set.

1. Fit GRBF networks on the embedding and randomly generate $n$ on-manifold tuning samples within the embedding, where $n$ is the number of training points.
2. Generate off-manifold tuning samples by biasing each training point 1 unit distance away from the manifold along the normal (both directions) of the local hyperplane.
3. Tune a data-dependent kernel function by using the tuning samples, that is, solve (10).
4. Find the inverse of the GRBF networks obtained in step 1 according to [11].
5. Train an MLP network according to [10] by using the training samples as inputs and the embedding as targets. The tuning samples obtained in steps 1 and 2 are used as a validation set. Training is performed until the validation error is equal to the minimum error in step 3 or a maximum number of epochs (200) is achieved. We set the number of neurons in the hidden layer to $n$ to make MLP more comparable to the proposed method, where there are $n$ Gaussian basis functions centered at the training data.

Fig. 1b illustrates the GRBF mapping, which appears to perform well, since embedding interpolations are mapped to correct locations in the input space. Fig. 2a shows the tuning curve of the data-dependent kernel for the 800-point data set. The curve appears to be convex, thus guaranteeing that consistent solutions can be achieved.

Figs. 3a and 3b illustrate the training progress of the MLP networks. It is evident that arbitrarily, small validation errors cannot be achieved within an appreciably small number of epochs, since after about 100 epochs, the validation errors simply remain *roughly* constant. Furthermore, these constant validation errors are much larger than the error at the solution point of (10) (both methods minimize the same objective function). However, it can be seen that errors from the *training set* monotonically decrease until they are almost zero, indicating that the MLP training was correctly performed. Fig. 3c illustrates a comparison of training times between optimizing (10) and updating weights of MLP networks. The long training time of MLPs renders performing cross validation to select the optimal number of hidden-layer neurons unattractive compared to our method, which requires only one instance of solving (10).



(a)

(b)

(c)

Fig. 3. (a) and (b) MLP training profile for two Swiss-roll data sets. Similar curves were obtained for the other data sets of different sizes. (c) To produce an objective comparison of the training times, Matlab's `fminunc` function was used to solve (10). The MLP network weights were updated using the Scaled Conjugate Gradient (SCG) algorithm, as done in [10]. We used Netlab's [16] SCG implementation. In our experiments, manual tuning of (10) is generally much faster than `fminunc`. Terminating the MLP optimization earlier (say, at the 100th epoch when the validation error persists roughly at the same level) will halve the training time, but this would require the crafting of a more sophisticated stopping criterion for SCG. (a) 500 training points. (b) 1000 training points. (c) Comparison of training times.

For each Swiss-roll data set, in the same process of generating the $n$ training points $\mathcal{X} \in \mathbb{R}^3$, 500 testing points $\mathcal{Y} \in \mathbb{R}^3$ were created. Note that the testing points should not be confused with the tuning samples generated from $\mathcal{X}$ to tune a data-dependent kernel. The tuning samples were created using the methods described in Section 2.3.1, whereas the testing points were directly obtained from the Swiss-roll generation process. The ground truth embedding coordinates of $\mathcal{X}$ and $\mathcal{Y}$, respectively, $\mathcal{T} \in \mathbb{R}^2$ and $\mathcal{U} \in \mathbb{R}^2$ were also obtained.[4] From $\mathcal{X}$, MVU returns an embedding $\mathcal{E} \in \mathbb{R}^2$. Although $\mathcal{E}$ resembles $\mathcal{T}$, they are not exactly the same, since manifold learning, in general, is not perfect. The differences between $\mathcal{E}$ and $\mathcal{T}$ contribute to the *intrinsic perturbation* of the embedding $\mathcal{E}$, which can be quantified using the MSE as

$$R_{\mathcal{E}} = \frac{1}{n}\sum_{i=1}^{n} \|\mathbf{e}_i - \mathbf{t}_i\|_2^2, \qquad (11)$$

where $\mathbf{e}_i \in \mathcal{E}$ and $\mathbf{t_i} \in \mathcal{T}$ are corresponding points on the manifold.[5] Equation (11) provides a lower bound of error for any extrapolation method, since it captures how well MVU really captures the true embedding space. Note that the manifold extrapolation methods were based on learning a mapping from the corresponding points between $\mathcal{X}$ and $\mathcal{E}$, not $\mathcal{X}$ and $\mathcal{T}$.

Let the testing set $\mathcal{Y}$ be *extrapolated* (using the four compared methods separately) to produce a set of embedding coordinates denoted by $\mathcal{F}$. To compare the efficacy of different extrapolation methods, we applied the MSE as an error metric to quantify the extrapolation error of a particular method:

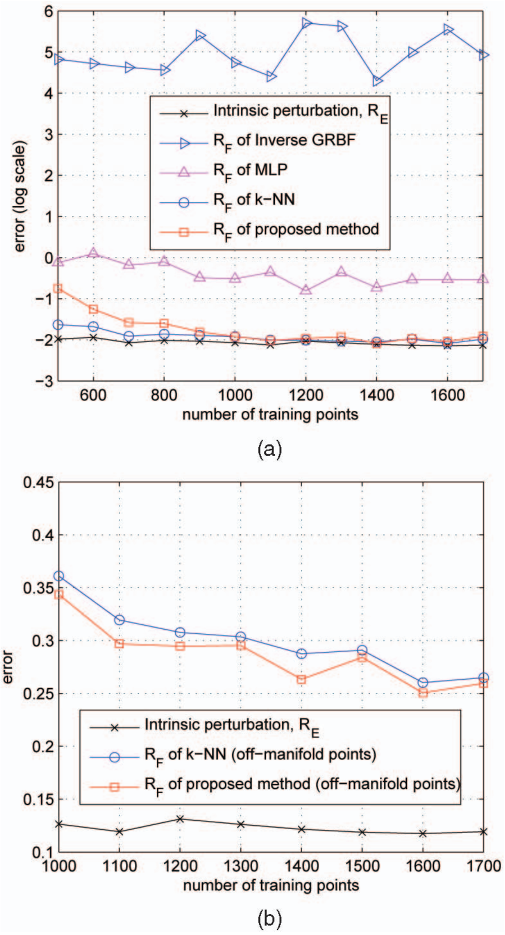
$$R_{\mathcal{F}} = \frac{1}{m}\sum_{i=1}^{m} \|\mathbf{f}_i - \mathbf{u}_i\|_2^2, \qquad (12)$$

where $\mathbf{f}_i \in \mathcal{F}$ and $\mathbf{u}_i \in \mathcal{U}$ are corresponding points on the manifold, and $m = 500$ is the number of testing points. For the proposed method, the testing set is simply used as an input to (6) with the optimized Gaussian widths, whereas for GRBF [11] and MLP [10], the testing set is evaluated using, respectively, the inverse GRBF networks and the trained MLP networks. For the $k$-NN method, the following steps, as described in [12, Section 6.1], are carried out for each vector $\mathbf{y}_i \in \mathcal{Y}$:

1. Identify the $k$-NN's of $\mathbf{y}_i$ among the *training set* $\mathcal{X}$.
2. Compute the linear weights $w_j$ that best reconstruct $\mathbf{y}$ from the $k$-NN's, subject to the sum-to-one constraint $\sum_{j=1}^{k} w_j = 1$.
3. Output $\mathbf{f}_i = \sum_{j=1}^{k} w_j \mathbf{e}_j$, where $\mathbf{e}_j$ are the embedding coordinates of the $k$-NN's produced from manifold learning.

In our evaluation, $k$ is set to 6 for the $k$-NN method (that is, the same number of neighbors used during manifold learning).

Fig. 4a illustrates the results, in terms of the extrapolation error of $\mathcal{Y}$ given by (12), of the four methods on the Swiss-roll data sets. It is evident that inverse GRBF and MLP perform very poorly, relative to the intrinsic perturbation, whereas $k$-NN and the proposed method produced much better results. The results also show that the proposed method does not perform as well as $k$-NN for smaller data sets (500 to 800 points), but eventually, very similar levels of extrapolation accuracy are achieved for both methods on subsequently larger data sets. This suggests that the performance of the proposed method improves with higher sampling resolutions of the underlying manifold. Indeed, Bengio et al. [5] prove that in the limit, the Nyström estimator (upon which the proposed method is based) converges to the true underlying eigenfunction of the data distribution. On the other hand, the $k$-NN method, which is based on the notion of locally linear neighborhood structures, seems to be less affected by sampling resolution.



(a)

(b)

Fig. 4. Comparing the efficacy of different extrapolation methods on the Swiss-roll data sets. Note that the error is in log scale. (a) With test points $\mathcal{Y}$ (*on manifold*). (b) With test points $\mathcal{Y}^{off}$ (*off manifold*).

To investigate the competency in extrapolating *slightly* off-manifold points, the testing set $\mathcal{Y}$ of each Swiss-roll data set is biased along the surface normal of the manifold to produce a new testing set $\mathcal{Y}^{off}$. The amount of bias is the average nearest neighbor distance of the training data. By using $\mathcal{U}$ as the ground truth of $\mathcal{Y}^{off}$, the extrapolation error given by (12) is recomputed. The results in Fig. 4b depict that for sufficiently large training sets, the proposed method consistently outperforms

[4] The actual process is first defines a set of 2D coordinates confined within a rectangle, respectively, $\mathcal{T}$ and $\mathcal{U}$, which are then "rolled" to create a Swiss-roll manifold $\mathcal{X}$ and $\mathcal{Y}$.

[5] Note that aside from imperfect manifold learning, $\mathcal{E}$ and $\mathcal{T}$ can differ due to scale, rotational, and translational differences, since the objective of MVU is only to maximize the output variance while preserving local geometry. Equation (11) is computed *after* accounting for scaling, rotation, and translation by first mapping $\mathcal{E}$ to $\mathcal{T}$ with an affine transformation estimated using least squares. Equation (12) is also computed *after* mapping $\mathcal{F}$ by using the mapping estimated from $\mathcal{E}$ and $\mathcal{T}$ (not from $\mathcal{F}$ and $\mathcal{U}$!).

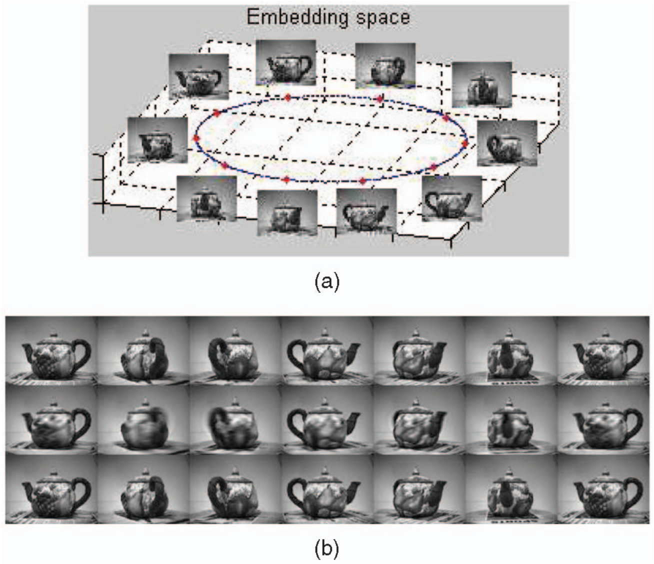


Fig. 5. (a) Blue dots represent the embedding, whereas red dots correspond to exemplar images. (b) The top and bottom rows show successive images of the manifold, whereas the middle row shows the GRBF interpolation between them. (a) MVU embedding of the Teapots dataset. (b) Samples of the on-manifold tuning set.

$k$-NN. This is not surprising, since the proposed method was optimized to handle slightly off-manifold points, for which the local neighborhood structure in the input space is not preserved in the embedding space, thus (slightly) violating the locally linear condition on which the $k$-NN method operates.

### 3.2 Image Data

We also considered three data sets of images: 1) the teapots data set [3], which contains 400 images (101 × 76) of a teapot in full rotation, 2) a set of 831 face images (50 × 60), where the subject incrementally varied his headpose without translational movement[6] (the "face" data set), and 3) a set of 2,500 images (64 × 64) densely sampled from a view hemisphere of a soft toy [17] (the "soft-toy" data set). These were processed as follows for training and testing:

1. Since the teapots and face data sets are much smaller, instead of partitioning to create disjointed training and testing sets, all images were used for training. This is to ensure that we have a set of dense samples of the underlying manifold for training. The testing images were obtained by *corrupting* all the training images with *random noise*.
2. For the soft-toy data set, each image is pretagged with longitudinal and latitudinal values, respectively, in the range $[0, 1, \cdots, 99]$ and $[0, 1, \cdots, 24]$. We chose images with longitudes and latitudes, respectively, of $[0, 2, \cdots, 98]$ and $[0, 2, \cdots, 24]$ as training images (a total of 650), whereas the rest was designated as testing images (a total of 1,850), which were also *blurred* by convolving them with a Gaussian filter.

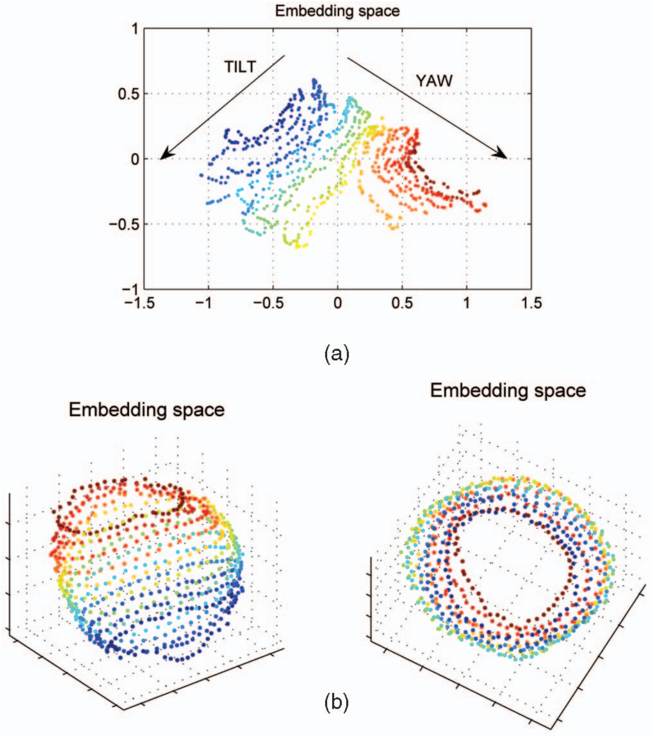


Fig. 6. For the face data set, the color progression depicts continuous frames of the head movement. For the soft-toy data set, each color represents a specific latitude on the view hemisphere. (a) MVU embedding of the face data set. (b) MVU embedding of the soft-toy data set (in two views).

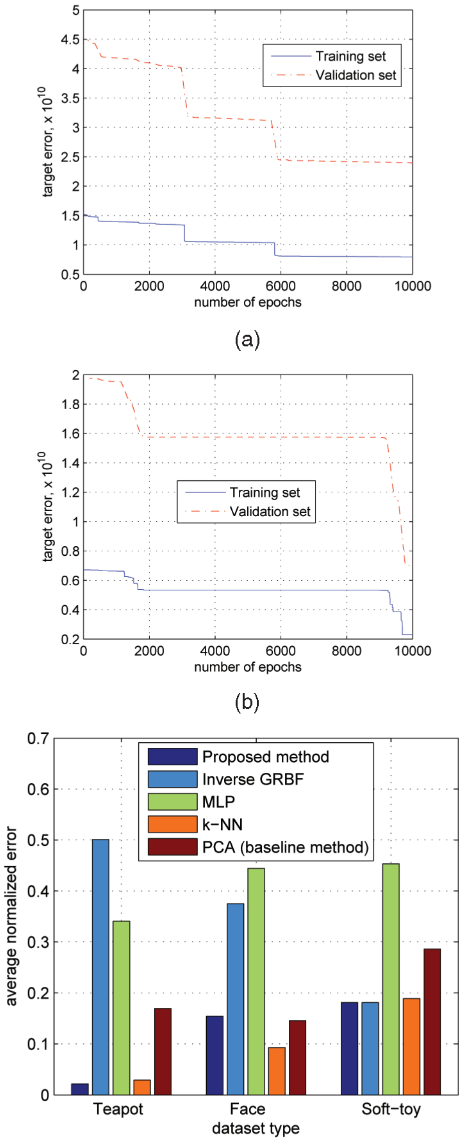
For the training sets created, MVU with 6-neighborhood returned the following results:

1. For the teapots data set, a 2D embedding, with the points forming a connected circle (the intrinsic manifold dimensionality is 1; see Fig. 5a).
2. For the face data set, a 2D embedding, with each dimension corresponding roughly to the tilt and yawn movement of the head (see Fig. 6a).
3. For the soft-toy data set, a 3D embedding, with the points forming a cylindrical structure (the intrinsic manifold dimensionality is 2; see Fig. 6b).

For each embedding, we tuned a data-dependent kernel function for manifold extrapolation and compared it against inverse GRBF [11], MLP [10], and the $k$-NN method [12] (with $k = 6$). Linear PCA, with the eigenspace dimensionality being equal to the MVU embedding dimensionality, was also implemented as a baseline method. Figs. 2b, 2c, and 2d show the tuning curves for all three data sets. For inverse GRBF, we simply computed the inverse [11] of the GRBF network fitted to generate the on-manifold tuning samples for the proposed method. Fig. 5b shows examples of the on-manifold tuning set generated for the teapots data set. They appear to smoothly interpolate; hence, it can be deduced that the embedding-to-input-space mapping performs well, even for very high dimensional inputs.

For MLP networks, the number of hidden layer neurons was set to $n$, where $n$ is the number of training vectors, again to make MLP more comparable to our method, which

[6] Available at http://www.seas.upenn.edu/~jhham/papers/face.mat.gz.



(c)

Fig. 7. (a) and (b) MLP training profiles for the face and soft-toy data sets for 10,000 epochs (the profile for the teapots data set looks similar and is omitted to conserve space). The network weights were updated using Netlab's [16] SCG implementation. (c) Summary of the manifold extrapolation performance of five methods on three data sets. The errors are normalized for better comparison. (a) Face data set, over $\sim 21$ hours. (b) Soft-toy data set, over $\sim 17$ hours. (c) Manifold extrapolation performance.

places $n$ Gaussian basis functions on the training data. The tuning set created to tune data-dependent kernels for the three data sets was used as a validation set during MLP training. Figs. 7a and 7b depict the training profiles for the face and soft-toy data sets for 10,000 epochs, which consumed about 21 and 17 hours to produce (the profile for the teapots data set looks similar). The validation errors are far higher than the minimum tuning error of the data-dependent kernels (note the scale of the errors); that is, even after considerable training effort, our MLP networks are only partially trained. Indeed, it is evident that cross validating to choose the number of optimal hidden neurons is practically infeasible. The required computational effort stands in stark contrast against the proposed method, where solving (10) is in the order of tens of minutes.

For testing, the error metrics that can be used are dependent on the availability of ground truths of the three data sets. The following metrics were defined and applied in our tests:

1. For teapots, each training and testing image is labeled with a value $[1, 2, \cdots, 400]$ that corresponds to a specific viewpoint. A testing image is extrapolated, its nearest neighbor in the embedding space (that is, one of the embedded training images) is sought, and the difference of the two label values is used as an error measurement. Note that the periodicity of the manifold has to be considered; for example, labels 3 and 398 have a distance of 5.
2. For faces, the euclidean distance between the extrapolated coordinate of a testing image to the coordinate of the embedding of the corresponding training image is used as an error measure. Although this is not necessarily a good performance measure (as the following results indicate), we are hampered by the unavailability of ground truth headpose coordinates. Nonetheless, visually comparing the results is still useful and interesting.
3. For soft toy, the nearest neighbor in the embedding space (that is, one of the embedded training images) of an extrapolated testing image is first obtained. The sum-of-squared-differences of the longitudinal and latitudinal values, with consideration to periodicity, is used as an error metric. This corresponds to the geodesic distance along the manifold surface. Note that 0 error is not possible due to the disjointed training and testing set.

Fig. 8 illustrates several results of the nearest neighbor matching in the embedding space. In general, the proposed method and $k$-NN produced the best results, with visually closer matches than the other methods. PCA occasionally gave dramatically erroneous results, which indicates that locally complex structures exist on the three image manifolds that cannot be modeled well with only a few linear principal components. It also seems that extrapolation via inverse GRBF does not perform well, even though the embedding-to-input-space mapping achieved good interpolations. Nonetheless, the soft-toy data set presents an exception to this result among the data sets considered, which is probably because it better satisfies the "linear approximation" condition [11] of the inverse GRBF method. Note that inverse GRBF produced sufficiently accurate manifold extrapolation for pose estimation by using the silhouettes in [11]. For the MLP networks, since they are only partially trained, the matching results are expectedly unsatisfactory.

Fig. 7c summarizes the performance by using the error metrics previously described of all the methods implemented. The errors were first normalized by respectively dividing against the upper bound of achievable error of each data set. The proposed technique produced the lowest average results for the teapots and soft-toy data sets, with $k$-NN following closely. For the face data set, $k$-NN gave the

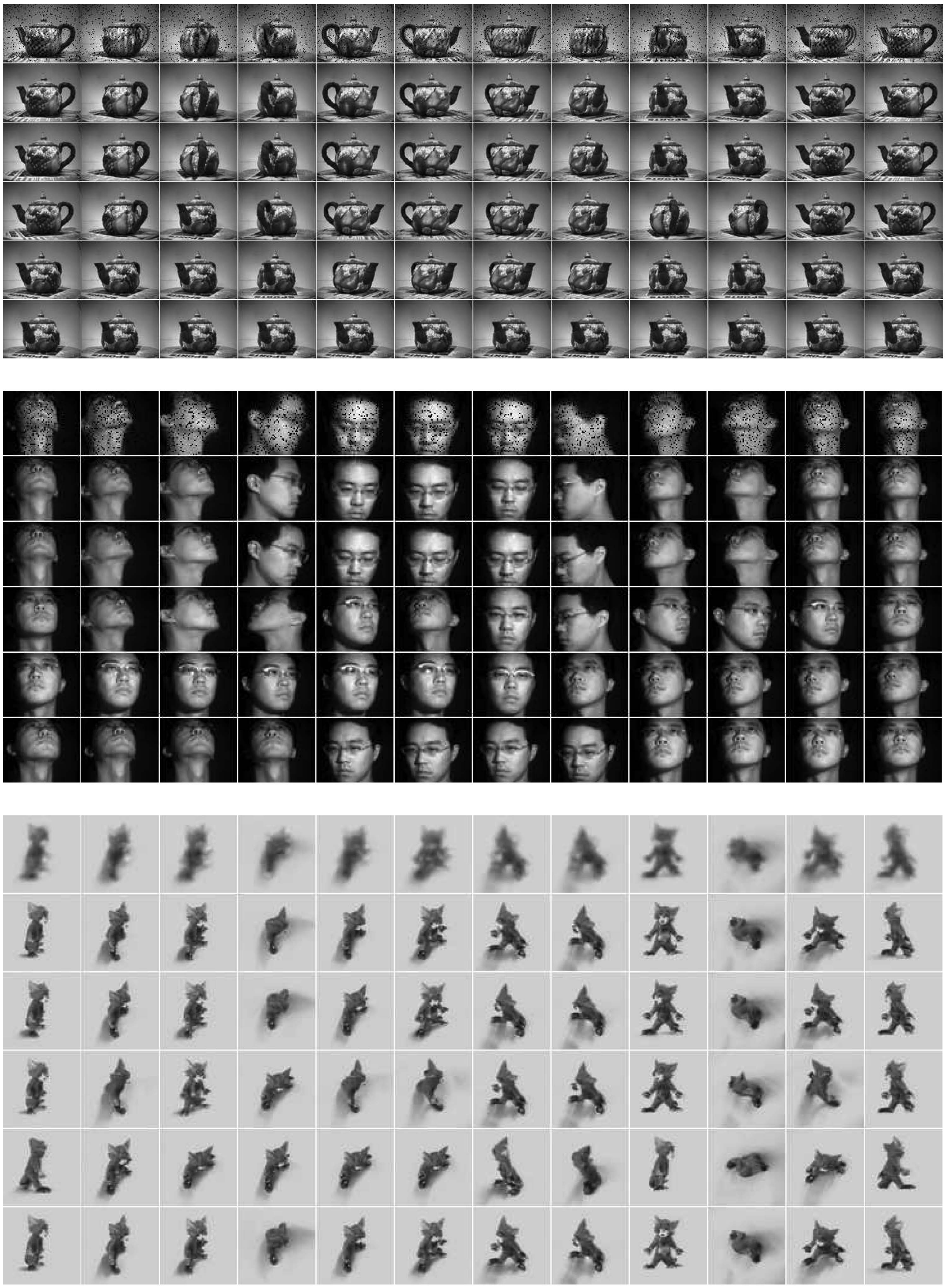


Fig. 8. Cleaning corrupted images. For each block, row 1 shows the testing images, whereas rows 2, 3, 4, 5, and 6 show the closest matching image respectively returned by the proposed method, $k$-NN, PCA, MLP, and inverse GRBF.

best results in terms of the error metric employed, whereas PCA is slightly better than the proposed method. Nonetheless, for the face data set, as shown in Fig. 8, image matching by the proposed method and $k$-NN is much better than PCA's; that is, note the clearly wrong PCA matchings in columns 4, 5, 6, 9, 10, and 11 of faces. We also emphasize that one should not view the application of these experiments as the pose estimation of objects and seek to compare against specialized pose estimation methods, for example, [17]. Our objective is manifold extrapolation, and it just so happened that the manifolds that we have considered are populated from images of objects in different viewpoints.

## 4 Conclusion

In this paper, we have proposed a novel method for extrapolating manifolds learned via MVU to novel out-of-sample data. Central to our approach is the Nyström formula for approximating eigenfunctions from finite samples, which has been observed in the literature to be equivalent to the KPCA projection formula. The essence of our solution lies in tuning data-dependent kernel functions derived from Gaussian basis functions by using artificially generated on-manifold and off-manifold samples. In conjunction with the Nyström formula, the tuned kernel function can then be freely applied to unseen before data. We have demonstrated the effectiveness of our proposed method through extensive experiments on synthetic and real data.

## Acknowledgments

The authors would like to thank the reviewers for their helpful comments.

## References

[1] C. Burges, "Geometric Methods for Feature Extraction and Dimensional Reduction," *Data Mining and Knowledge Discovery Handbook: A Complete Guide for Practitioners and Researchers,* L. Rokach and O. Maimon, eds. Kluwer Academic Publishers, 2005.

[2] L. Saul, K. Weinberger, F. Sha, J. Ham, and D. Lee, "Spectral Methods for Dimensionality Reduction," *Semisupervised Learning,* O. Chapelle, B. Schölkopf, and A. Zien, eds. MIT Press, 2006.

[3] K. Weinberger and L. Saul, "Unsupervised Learning of Image Manifolds by Semidefinite Programming," *Int'l J. Computer Vision,* vol. 70, no. 1, pp. 77-90, 2006.

[4] Y. Bengio, J.-F. Paiement, P. Vincent, O. Delalleau, N. Le Roux, and M. Ouimet, "Out-of-Sample Extensions for LLE, Isomap, MDS, Eigenmaps, and Spectral Clustering," *Proc. Advances in Neural Information Processing Systems,* 2003.

[5] Y. Bengio, O. Delalleau, N. Le Roux, J.-F. Paiement, P. Vincent, and M. Ouimet, "Learning Eigenfunctions Links Spectral Embedding and Kernel PCA," *Neural Computation,* vol. 16, no. 10, pp. 2197-2219, 2004.

[6] B. Schölkopf, A. Smola, and K.-R. Müller, "Nonlinear Component Analysis as a Kernel Eigenvalue Problem," *Neural Computation,* vol. 10, pp. 1299-1319, 1998.

[7] V. de Silva and J. Tenenbaum, "Global versus Local Methods in Nonlinear Dimensionality Reduction," *Proc. Advances in Neural Information Processing Systems,* pp. 705-712, 2003.

[8] K. Weinberger, B. Packer, and L. Saul, "Nonlinear Dimensionality Reduction by Semidefinite Programming and Kernel Matrix Factorization," *Proc. 10th Int'l Workshop Artificial Intelligence and Statistics (AISTATS '05),* 2005.

[9] S. Vishwanathan, K. Borgwardt, O. Guttman, and A. Smola, "Kernel Extrapolation," *Neurocomputing,* no. 69, pp. 721-729, 2006.

[10] H. Gong, C. Pan, Q. Yang, H. Lu, and S. Ma, "Neural Network Modeling of Spectral Embedding," *Proc. 17th British Machine Vision Conf. (BMVC '06),* 2006.

[11] A. Elgammal and C.-S. Lee, "Inferring 3D Body Pose from Silhouettes Using Activity Manifold Learning," *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR '04),* vol. 2, pp. 681-688, 2004.

[12] L.K. Saul and S.T. Roweis, "Think Globally, Fit Locally: Unsupervised Learning of Low-Dimensional Manifolds," *J. Machine Learning Research,* vol. 4, pp. 119-155, 2003.

[13] C. Baker, *The Numerical Treatment of Integral Equations.* Clarendon Press, 1977.

[14] C. Williams and M. Seeger, "Using the Nyström Method to Speed Up Kernel Machines," *Proc. Advances in Neural Information Processing Systems,* pp. 682-688, 2001.

[15] A. Schwaighofer, V. Tresp, and K. Yu, "Learning Gaussian Process Kernels via Hierarchical Bayes," *Proc. Advances in Neural Information Processing Systems,* 2005.

[16] I.T. Nabney, *NETLAB: Algorithms for Pattern Recognition.* Springer, 2004.

[17] G. Peters, "Efficient Pose Estimation Using View-Based Object Representations," *Machine Vision and Applications,* vol. 16, no. 1, pp. 59-63, 2004.

**Tat-Jun Chin** received the BEng degree in mechatronics engineering from the Universiti Teknologi Malaysia (UTM) in 2003 and the PhD degree from the Department of Electrical and Computer Systems Engineering (ECSE), Monash University, Victoria, Australia, in 2007. He is currently a research fellow in the Institute for Infocomm Research $(\mathrm{I}^2\mathrm{R})$, Agency for Science, Technology and Research $(\mathrm{A}^*\mathrm{STAR})$, Singapore. His research interests include computer vision and machine learning.

**David Suter** received the BSc degree in applied mathematics and physics from the Flinders University of South Australia in 1977 and the PhD degree in computer vision from La Trobe University in 1991. He is a professor in the Department of Electrical and Computer Systems Engineering (ECSE), Monash University. He was a general cochair for several conferences. He currently serves on the editorial boards of the *International Journal of Computer Vision* and the *International Journal of Image and Graphics*. His main research interest is motion estimation from images and visual reconstruction. He is a senior member of the IEEE.