# Parametric Dimensionality Reduction by Unsupervised Regression

Miguel Á. Carreira-Perpiñán
EECS, University of California, Merced
http://eecs.ucmerced.edu

Zhengdong Lu
CS, University of Texas, Austin
luz@cs.utexas.edu

## Abstract

*We introduce a parametric version (pDRUR) of the recently proposed Dimensionality Reduction by Unsupervised Regression algorithm. pDRUR alternately minimizes reconstruction error by fitting parametric functions given latent coordinates and data, and by updating latent coordinates given functions (with a Gauss-Newton method decoupled over coordinates). Both the fit and the update become much faster while attaining results of similar quality, and afford dealing with far larger datasets ($10^5$ points). We show in a number of benchmarks how the algorithm efficiently learns good latent coordinates and bidirectional mappings between the data and latent space, even with very noisy or low-quality initializations, often drastically improving the result of spectral and other methods.*

We consider the problem of dimensionality reduction, where given a high-dimensional dataset of $N$ points in $D$ dimensions $\mathbf{Y}_{D\times N} = (\mathbf{y}_1, \dots, \mathbf{y}_N)$, we want to estimate mappings $\mathbf{F}: \mathbf{y} \to \mathbf{x}$ (dimensionality reduction) and $\mathbf{f}: \mathbf{x} \to \mathbf{y}$ (reconstruction) between data points $\mathbf{y} \in \mathbb{R}^D$ and latent points $\mathbf{x} \in \mathbb{R}^L$ with $L < D$. One reason why this is a hard problem is that the latent points $\mathbf{X}_{L\times N} = (\mathbf{x}_1, \dots, \mathbf{x}_N)$ are unknown; the problem is unsupervised. If the $\mathbf{X}$ were known, estimating the mappings would be a far easier (parametric or nonparametric) regression problem: fit $\mathbf{F}$ to $(\mathbf{Y}, \mathbf{X})$ and $\mathbf{f}$ to $(\mathbf{X}, \mathbf{Y})$. There is a class of dimensionality reduction methods, spectral methods such as Isomap [17] or Laplacian eigenmaps (LE; [1]) that, precisely, estimate these latent coordinates $\mathbf{X}$ from the data. In recent years, spectral methods have been very popular due to their simplicity, lack of local optima, efficient computation using standard eigensolvers, and more importantly their success with some complex datasets. However, spectral methods do require a careful search over their parameters (number of nearest neighbors $k$, local scale, etc.), and even the best parameters may not produce satisfactory results. The literature contains many such examples: the latent coordinates often show defects (not corresponding to true structure in the data) where distant points in the manifold are mapped to tight clusters in latent space while other points are left outlying; low-density regions or holes in the manifold are amplified; streaks protrude far outside the boundary of the data; entire branches of the manifold may project on top of each other; and others. Reasons for this include a poor neighborhood graph, as well as the particular nature of each method. And if the latent points $\mathbf{X}$ fail to be an orderly projection of the data $\mathbf{Y}$, the mappings fitted to them (e.g. an out-of-sample extension; [2]) will be a poor representation of the data manifold and generalize badly to unseen data.

In this paper we focus on *unsupervised regression* methods for dimensionality reduction (reviewed in section 4). Here, one solves a regression problem where the outputs $\mathbf{Y}$ are given but not the inputs $\mathbf{X}$ (or vice versa), by minimizing an error function over both the mapping parameters and the unknown inputs, considered not as variables but as parameters as well. The minimization can be naturally done alternatingly, by fitting the mapping given the inputs are fixed (the usual, supervised regression) and by updating the inputs given the mapping is fixed. Specifically, we focus on a recent method, (nonparametric) *Dimensionality Reduction by Unsupervised Regression* (nDRUR; [3]), which when initialized from the coordinates produced by a spectral method has been shown to produce far improved latent representations and mappings. Unfortunately, as stated in [3], the high computational cost of this method (training is cubic on the number of points $N$, while testing is linear in $N$) currently limits its use to datasets of a few thousand points—a common characteristic of nonparametric methods. In this paper, we propose a parametric version of this method, so that the infinite-dimensional minimization becomes finite-dimensional. As we will show, this has far-reaching consequences. Not only does the step of fitting the mappings become much faster because it involves a smaller number of parameters; but, crucially, the complex, high-dimensional minimization over the latent coordinates simplifies enormously into decoupled, low-dimensional minimizations that are performed efficiently and robustly with a Gauss-Newton method. We find that we can achieve results of as good a quality as those of nDRUR but at a fraction of the time and space cost: a 100D dataset with $N = 20\,000$

takes a few minutes in a modern workstation.

After briefly describing the nonparametric DRUR method in section 1, we present our new method and its optimization in section 2, report extensive experimental results in section 3 and review related work in section 4.

## 1. Nonparametric DRUR

The method of nonparametric Dimensionality Reduction by Unsupervised Regression (nDRUR; [3]) variationally minimizes the following objective function over mappings $\mathbf{f}$, $\mathbf{F}$ and latent coordinates $\mathbf{X}_{L\times N}$:

$$\min_{\mathbf{X},\mathbf{f},\mathbf{F}} E(\mathbf{X},\mathbf{f},\mathbf{F}) = E_{\mathbf{f}}(\mathbf{X},\mathbf{f}) + E_{\mathbf{F}}(\mathbf{X},\mathbf{F}) \quad (1)$$

$$E_{\mathbf{f}}(\mathbf{X},\mathbf{f}) = \textstyle\sum_{n=1}^{N} \|\mathbf{y}_n - \mathbf{f}(\mathbf{x}_n)\|^2 + \lambda_{\mathbf{f}} R_{\mathbf{f}}(\mathbf{f}) \quad (2)$$

$$E_{\mathbf{F}}(\mathbf{X},\mathbf{F}) = \textstyle\sum_{n=1}^{N} \|\mathbf{x}_n - \mathbf{F}(\mathbf{y}_n)\|^2 + \lambda_{\mathbf{F}} R_{\mathbf{F}}(\mathbf{F}) \quad (3)$$

with appropriate quadratic regularizers $R_{\mathbf{f}}(\mathbf{f})$ and $R_{\mathbf{F}}(\mathbf{F})$. The variational minimization over $(\mathbf{f},\mathbf{F})$ for fixed $\mathbf{X}$ yields a unique solution consisting of a radial basis function expansion centred at each of the data points ($\mathbf{x}_1,\dots,\mathbf{x}_N$ or $\mathbf{y}_1,\dots,\mathbf{y}_N$). Computationally, it involves solving two regularized linear systems of $N\times N$ and is thus $\mathcal{O}(N^3)$ in time and $\mathcal{O}(N^2)$ in memory. Both mappings are nonparametric, and mapping a test point is $\mathcal{O}(NL)$ for $\mathbf{f}$ and $\mathcal{O}(ND)$ for $\mathbf{F}$, besides the $\mathcal{O}(N(D+L))$ memory cost of storing $\mathbf{X}$ and $\mathbf{Y}$. The minimization over $\mathbf{X}$ is substantially more complex: it is highly nonlinear and takes place in a large space of $NL$ parameters, since each $\mathbf{x}_n$ appears within $\mathbf{f}$ as a centre and so all $N$ terms in $E_{\mathbf{f}}$ are coupled. This same problem affects other nonparametric unsupervised regression methods such as GPLVM.

## 2. Parametric DRUR

Although it may be possible to derive a faster but approximate nDRUR method by using active-set techniques of the type used in spectral methods and Gaussian processes, here we propose the more direct approach of redefining the problem parametrically. We define *parametric Dimensionality Reduction by Unsupervised Regression* (pDRUR) by the same objective function (1) over $(\mathbf{X},\mathbf{f},\mathbf{F})$ but now $\mathbf{f}$ and $\mathbf{F}$ take a particular parametric form; for simplicity of notation, we will still write "$\min_{\mathbf{f}}$" to mean a minimization over the parameters of $\mathbf{f}$ rather than a variational minimization. The regularizers $R_{\mathbf{f}}(\mathbf{f})$ and $R_{\mathbf{F}}(\mathbf{F})$ become now penalties on the parameters (e.g. weight decay in neural nets). Possible choices for $\mathbf{f}$, $\mathbf{F}$ include multilayer perceptrons (MLPs) and radial basis function networks (RBFs). Given enough hidden units or basis functions, respectively, both have the universal mapping approximation property. We give details in the adaptation step. We use alternating minimization over $\mathbf{X}$ (projection step) and $(\mathbf{f},\mathbf{F})$ (adaptation step), and initialize $\mathbf{X}$ to the output of a spectral method; see figure 1. Importantly, the overall training cost is linear in $N$.

**input** $\mathbf{Y}_{D\times N} = (\mathbf{y}_1,\dots,\mathbf{y}_N)$
Obtain $\mathbf{X}_{L\times N} = (\mathbf{x}_1,\dots,\mathbf{x}_N)$ from a spectral method
Fit parametric mappings $\mathbf{f}$ to $(\mathbf{X},\mathbf{Y})$ and $\mathbf{F}$ to $(\mathbf{Y},\mathbf{X})$
**repeat**
  *Project*: **for** $n = 1,\dots,N$
    $\mathbf{x}_n$ = approximate minimizer of (7) with Gauss-Newton
  **end**
  *Adapt*: approximately fit parametric mappings $\mathbf{f}$, $\mathbf{F}$
**until** convergence
**return** $\mathbf{f}$, $\mathbf{F}$, $\mathbf{X}$

Figure 1. Parametric DRUR algorithm.

The parametric choice has the disadvantages over the nonparametric RBFs that the adaptation step over $(\mathbf{f},\mathbf{F})$ may now have local optima (depending on the parametric model), and that model selection must be solved (number of hidden units or basis functions, regularization parameters)—though this is no different from model selection in a standard regression setting. But it has two important advantages: a dramatically simpler optimization over $\mathbf{X}$, and faster $\mathbf{f}$, $\mathbf{F}$ both in training and testing.

### 2.1. Projection step: optimization over $\mathbf{X}$

For fixed, parametric $\mathbf{f}$ and $\mathbf{F}$, we have the following minimization over $\mathbf{X}$:

$$\min_{\mathbf{X}\in\mathbb{R}^{N\times L}} \textstyle\sum_{n=1}^{N} \|\mathbf{y}_n - \mathbf{f}(\mathbf{x}_n)\|^2 + \lambda_{\mathbf{f}} R_{\mathbf{f}}(\mathbf{f}) \quad (4)$$

$$+ \textstyle\sum_{n=1}^{N} \|\mathbf{x}_n - \mathbf{F}(\mathbf{y}_n)\|^2 + \lambda_{\mathbf{F}} R_{\mathbf{F}}(\mathbf{F}) \quad (5)$$

$$= \textstyle\sum_{n=1}^{N} E_n(\mathbf{x}_n) + \lambda_{\mathbf{f}} R_{\mathbf{f}}(\mathbf{f}) + \lambda_{\mathbf{F}} R_{\mathbf{F}}(\mathbf{F}) \quad (6)$$

*which separates over each* $\mathbf{x}_n \in \mathbb{R}^L$ because $\mathbf{f}$ and $\mathbf{F}$ do not depend on $\mathbf{X}$ (unlike in nDRUR, where $\mathbf{X}$ are the basis function centres in $\mathbf{f}$). Thus, instead of one large nonlinear minimization over $NL$ parameters, we have $N$ independent nonlinear minimizations each on $L$ parameters (the regularization terms do not depend on $\mathbf{X}$). Consider then the following problem (where we omit the subindex $n$ overall in this section):

$$\min_{\mathbf{x}\in\mathbb{R}^L} E(\mathbf{x}) = \|\mathbf{y} - \mathbf{f}(\mathbf{x})\|^2 + \|\mathbf{x} - \mathbf{F}(\mathbf{y})\|^2. \quad (7)$$

Here, $\mathbf{x}\in\mathbb{R}^L$ is the only free variable, and $\mathbf{y}\in\mathbb{R}^D$ and the functions $\mathbf{f}$ and $\mathbf{F}$ are fixed. We will assume that $\mathbf{f}$ is differentiable wrt $\mathbf{x}$, with a $D\times L$ Jacobian matrix and a $D\times L\times L$ third-order Hessian tensor:

$$\mathbf{J}(\mathbf{x}) = \left(\frac{\partial f_d}{\partial x_l}\right)_{dl}, \quad d = 1,\dots,D,\ l = 1,\dots,L$$

$$\mathbf{H}(\mathbf{x}) = \left(\frac{\partial^2 f_d}{\partial x_l \partial x_m}\right)_{dlm}, \quad d = 1,\dots,D,\ l,m = 1,\dots,L.$$

In the following, we will omit the dependence of $\mathbf{J}$ and $\mathbf{H}$ on $\mathbf{x}$ for clarity. Expressions for $\mathbf{J}$ for specific models appear in section 2.2. We now compute the gradient and Hessian of the objective $E$ wrt $\mathbf{x}$:

$$\nabla E(\mathbf{x}) = 2(-\mathbf{J}^T(\mathbf{y} - \mathbf{f}(\mathbf{x})) + \mathbf{x} - \mathbf{F}(\mathbf{y})) \quad (8)$$

$$\nabla^2 E(\mathbf{x}) = 2(\mathbf{I} + \mathbf{J}^T\mathbf{J} - \mathbf{H}^T(\mathbf{y} - \mathbf{f}(\mathbf{x}))) \quad (9)$$

where the notation for the product with $\mathbf{H}^T$ means summing across index $d$:

$$\mathbf{H}^T(\mathbf{y} - \mathbf{f}(\mathbf{x})) = \textstyle\sum_{d=1}^{D} (y_d - f_d(\mathbf{x}))\nabla^2 f_d(\mathbf{x}). \quad (10)$$

The form of the gradient and Hessian suggests using a Gauss-Newton method to minimize over $\mathbf{x}$. We can get a positive-definite approximation to the Hessian of $E$ (eq. (9)) by discarding the second-order term on $\mathbf{H}$:

$$\nabla^2 E(\mathbf{x}) \approx 2(\mathbf{I} + \mathbf{J}^T\mathbf{J}) \quad (11)$$

so the new iterate along the Gauss-Newton search direction is $\tilde{\mathbf{x}} = \mathbf{x} + \alpha\mathbf{p}$ with

$$\mathbf{p} = (\mathbf{I} + \mathbf{J}^T\mathbf{J})^{-1}(\mathbf{J}^T(\mathbf{y} - \mathbf{f}(\mathbf{x})) - \mathbf{x} + \mathbf{F}(\mathbf{y})) \quad (12)$$

and the step length $\alpha$ may simply be obtained by a backtracking search starting with $\alpha_0 = 1$ (corresponding to the full Gauss-Newton step, which will always be taken near convergence). The computational cost per iteration of this method for a single $\mathbf{x}_n$ is $\mathcal{O}(L^2 D)$ times the cost of computing one entry of the Jacobian (the cost term $L^3$ from solving the linear system can be ignored because $L < D$). This is just $L$ times the cost of computing the gradient. If the $L \times L$ linear system was large, it would be possible to solve it approximately with linear conjugate gradients, but that rarely would be the case: $L$ is the dimension of the latent space.

We can use a Gauss-Newton method because our objective function $E(\mathbf{x})$ is the sum of two least-squares problems of dimensions $D$ and $L$, respectively. The formulation of the pDRUR objective function has here a strong benefit. In a typical Gauss-Newton method, the approximation to the Hessian has the form $\mathbf{J}^T\mathbf{J}$ and thus can be singular (or ill-conditioned) whenever $\mathbf{J}$ is singular (or ill-conditioned), requiring careful addition of a term $\lambda\mathbf{I}$ with $\lambda > 0$ to it (the Levenberg-Marquardt method). However, in our case we get for free a strictly positive-definite Hessian approximation because of the term $\|\mathbf{x} - \mathbf{F}(\mathbf{y})\|^2$ in the objective. This has a regularizing effect, adding $\mathbf{I}$ to the Hessian, and being well-conditioned anywhere. Thanks to this fact, using theorem 10.1 in [11] we can prove global convergence if the line search satisfies certain conditions (e.g. the Wolfe conditions); that is, the method will converge to a local minimizer no matter the initial point. The convergence rate is in general linear, but faster than that of gradient descent because of the additional Hessian information. The approximation to the Hessian will be good if, near the minimizer, the orthogonal projection on $\mathbf{f}$ of the $\mathbf{y}$–error $(\mathbf{y} - \mathbf{f}(\mathbf{x}))$ is small, or if $\mathbf{f}$ has small curvature at $\mathbf{x}$.

Experimentally within pDRUR we found that nearly all points $\mathbf{x}_n$ converged to a relative error of $10^{-4}$ in 4 or fewer iterations and most in 1–2. The full step $\alpha = 1$ is accepted in the line search almost always: 99% of the times far from the minimizer (in the first pDRUR iteration) and 99.9% as iterations progress, where nearly all $\mathbf{x}_n$ converge in a single full step.

We also experimented with two other minimization methods. Gradient descent ($\tilde{\mathbf{x}} = \mathbf{x} - \eta\nabla E(\mathbf{x})$ with a step $\eta > 0$) required far more (tens) iterations and was overall slower by an order of magnitude. The fixed-point iteration $\tilde{\mathbf{x}} = \mathbf{g}(\mathbf{x})$ with $\mathbf{g}(\mathbf{x}) = \mathbf{F}(\mathbf{y}) + \mathbf{J}(\mathbf{x})^T(\mathbf{y} - \mathbf{f}(\mathbf{x}))$ (obtained by equating the gradient to zero) is intuitively appealing, as it shifts $\mathbf{x}$ from $\mathbf{F}(\mathbf{y})$ by a vector equal to the data space error $\mathbf{y} - \mathbf{f}(\mathbf{x})$ projected on the tangent space of $\mathbf{f}$ at the current $\mathbf{x}$. However, it did not converge in general. Since $\mathbf{g}(\mathbf{x}) = \mathbf{x} - \frac{1}{2}\nabla E(\mathbf{x})$, this iteration is actually gradient descent with a constant step $\eta = \frac{1}{2}$.

## 2.2. Adaptation step: parametric choices for f and F

The adaptation step involves two independent minimizations of (2) and (3), i.e., two standard regressions. If $\mathbf{f}$ and $\mathbf{F}$ are linear, pDRUR results in PCA (like nDRUR; [3]). We consider two parametric models and give the fit equations as well as the Jacobian wrt $\mathbf{x}$ (assume inputs $\mathbf{X}$, outputs $\mathbf{Y}$).

**Radial basis function network (RBF)** We consider $M$ Gaussian basis functions of width $\sigma$ (just as in nDRUR but with $M < N$) and a bias term:

$$\mathbf{f}(\mathbf{x}) = \mathbf{W}\Phi(\mathbf{x}) + \mathbf{w} = \textstyle\sum_{m=1}^{M} \mathbf{w}_m\phi_m(\mathbf{x}) + \mathbf{w} \quad (13)$$

where $\phi_m(\mathbf{x}) = \exp\left(-\frac{1}{2}\left\|(\mathbf{x} - \boldsymbol{\mu}_m)/\sigma\right\|^2\right)$. For simplicity and as is common with RBFs, we obtain the basis function centres by $k$–means clustering on the input data, and $\sigma$ by a grid search (training the RBF on a subset of the data and testing it on the rest, and picking the best $\sigma$). This suboptimal strategy does not guarantee a monotonic error decrease but having an occasional oscillation in the error is a small price to pay in exchange for the simplicity and speed of the fit. The adaptation step equations for the weights are analogous to those of nDRUR, and unlike for the MLP, the solution is unique, given by:

$$\min_{\mathbf{W}} \left\|\mathbf{Y} - \mathbf{W}\mathbf{G}_{\mathbf{xy}} - \mathbf{w}\mathbf{1}^T\right\|_F^2 + \lambda \operatorname{tr}\left(\mathbf{W}\mathbf{G}_{\mathbf{xx}}\mathbf{W}^T\right) \Rightarrow$$

$$\mathbf{W}(\mathbf{G}_{\mathbf{xy}}\mathbf{G}_{\mathbf{xy}}^T + \lambda\mathbf{G}_{\mathbf{xx}}) = (\mathbf{Y} - \mathbf{w}\mathbf{1}^T)\mathbf{G}_{\mathbf{xy}}^T \quad (14)$$

$$\mathbf{w} = \tfrac{1}{N}(\mathbf{Y} - \mathbf{W}\mathbf{G}_{\mathbf{xy}})\mathbf{1} \quad (15)$$

where we use a regularization term (on $\mathbf{W}$ only, not $\mathbf{w}$) with user parameter $\lambda \geq 0$, $\mathbf{1}$ is a column vector of $N$ ones, and

with the matrices $\mathbf{G_{xy}}$ of $M \times N$ with elements $\phi_m(\mathbf{x}_n)$; $\mathbf{G_{xx}}$ of $M \times M$ with elements $\phi_m(\boldsymbol{\mu}_m)$; $\mathbf{Y}$ of $D \times N$ (data points) and $\mathbf{X}$ of $L \times N$ (projections); $\mathbf{W}$ of $D \times M$ (weights) and $\mathbf{w}$ of $D \times 1$ (biases). The equation for $\mathbf{w}$ shows it captures the average error not accounted for by $\mathbf{W}$. Substituting it in the equation for $\mathbf{W}$, the explicit solution for $\mathbf{W}$ is given by the $M \times M$ positive definite linear system (positive semidefinite in non-generic cases):

$$\mathbf{W}\left(\mathbf{G_{xy}}\mathbf{G_{xy}}^T + \lambda \mathbf{G_{xx}} - \tfrac{1}{N}(\mathbf{G_{xy}}\mathbf{1})(\mathbf{G_{xy}}\mathbf{1})^T\right) = \mathbf{Y}\left(\mathbf{I} - \tfrac{1}{N}\mathbf{1}\mathbf{1}^T\right)\mathbf{G_{xy}}^T. \quad (16)$$

For $\mathbf{F}$ we can compute the centres and widths once and for all since $\mathbf{Y}$ is constant. For $\mathbf{f}$, $\mathbf{X}$ is not constant and besides will typically change a lot from its initial value (as seen in the experiments), so we need to update the centres and widths. The centres are recomputed with $k$–means at every iteration (or every few iterations), initialized at the previous iteration's value; for the very first iteration, we take the best $k$–means result from 20 random initial point assignments. We cross-validate the widths at each iteration for $\mathbf{f}$ and at the first iteration for $\mathbf{F}$ by training on a portion of the training data, testing on the rest, and picking the best width.

The computational cost is $\mathcal{O}(NM(M+D))$ in training time and $\mathcal{O}(MD)$ in memory, mainly driven by setting up the linear system for $\mathbf{W}$ (solving it is a negligible $\mathcal{O}(M^3)$ since $M \ll N$ in practice).

The gradient of $\mathbf{f}$ wrt $\mathbf{x}$ (Jacobian $\mathbf{J}$) is:

$$\mathbf{J}(\mathbf{x}) = \tfrac{1}{\sigma^2}\mathbf{W}\,\mathrm{diag}\,(\Phi')\,(\mathbf{M} - \mathbf{x}\mathbf{1}_M^T)^T \quad (17)$$

where $\Phi'$ applies $\varphi'$ to each $-\frac{1}{2}\|(\mathbf{x} - \boldsymbol{\mu}_m)/\sigma\|^2$, and $\mathbf{M} = (\boldsymbol{\mu}_1 \ldots \boldsymbol{\mu}_M)$. For Gaussian basis functions, $\varphi_m(t) = e^t$.

**Multilayer perceptron (MLP)** We consider a MLP with a single layer of $M$ hidden units with sigmoidal activation function and biases, trained with backpropagation:

$$\mathbf{f}(\mathbf{x}) = \mathbf{W}_2\sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2 \quad (18)$$

where $\sigma(x) = 1/(1+\exp(x))$ applies elementwise. The gradient wrt the weights of the squared reconstruction error $E = \sum_{n=1}^N \|\mathbf{y}_n - \mathbf{f}(\mathbf{x}_n)\|^2$ is:

$$\begin{aligned}
\nabla_{\mathbf{W}_2} E &= -2\textstyle\sum_{n=1}^N (\mathbf{y}_n - \mathbf{f}(\mathbf{x}_n))\boldsymbol{\sigma}_n^T \\
\nabla_{\mathbf{b}_2} E &= -2\textstyle\sum_{n=1}^N (\mathbf{y}_n - \mathbf{f}(\mathbf{x}_n)) \\
\nabla_{\mathbf{W}_1} E &= -2\textstyle\sum_{n=1}^N \mathrm{diag}\,(\boldsymbol{\sigma}_n')\,\mathbf{W}_2^T(\mathbf{y}_n - \mathbf{f}(\mathbf{x}_n))\mathbf{x}_n^T \\
\nabla_{\mathbf{b}_1} E &= -2\textstyle\sum_{n=1}^N \mathrm{diag}\,(\boldsymbol{\sigma}_n')\,\mathbf{W}_2^T(\mathbf{y}_n - \mathbf{f}(\mathbf{x}_n))
\end{aligned}$$

where $\boldsymbol{\sigma}_n$ and $\boldsymbol{\sigma}_n'$ apply $\sigma$ and $\sigma'$ elementwise to $\mathbf{W}_1\mathbf{x}_n + \mathbf{b}_1$, respectively. Momentum, weight decay and other techniques can be applied. Unlike with nDRUR, the adaptation step now has local optima. However, the minimization is far simpler than that of an autoencoder, which directly minimizes the reconstruction error

$$E(\mathbf{f}, \mathbf{F}) = \textstyle\sum_{n=1}^N \|\mathbf{y}_n - \mathbf{f}(\mathbf{F}(\mathbf{y}_n))\|^2 \quad (19)$$

over a network with 3 hidden layers. Instead, pDRUR solves two minimizations over two networks with a single hidden layer.

The gradient of $\mathbf{f}$ wrt $\mathbf{x}$ (Jacobian $\mathbf{J}$) is:

$$\mathbf{J}(\mathbf{x}) = \mathbf{W}_2\,\mathrm{diag}\,(\sigma'(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1))\,\mathbf{W}_1. \quad (20)$$

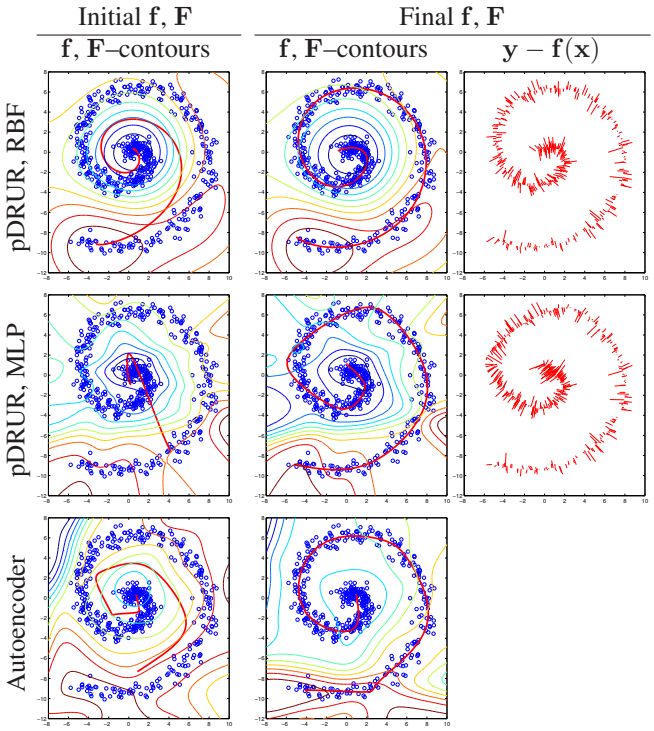


Figure 2. Noisy spiral with $N = 500$ points in 2D. *Row 1*: pDRUR with 10 RBFs and $\lambda = 0.1$, 10 iterations. *Row 2*: pDRUR with MLPs with a layer of 10 sigmoidal hidden units, 10 iterations. *Row 3*: autoencoder with 10–1–10 hidden layers. In all cases, the initial $\mathbf{X}$ were the true $\mathbf{X}$ with Gaussian noise of stdev 1 ($= \frac{1}{10}$ the range of $\mathbf{X}$). Note: most of our figures should be viewed in color.

## 3. Experiments

We have carried out a number of experiments with various initializations for $\mathbf{X}$ and compared with closely related methods: (1) autoencoders with 3 layers of $H$–$L$–$H$ hidden units trained for up to 1 000 epochs (using Matlab's Neural Network Toolbox), by initializing its mappings $\mathbf{f}$ and $\mathbf{F}$ to fit $(\mathbf{X}, \mathbf{Y})$ and $(\mathbf{Y}, \mathbf{X})$, respectively, just as DRUR does; and (2) two unsupervised regression methods that fit $\mathbf{f}$ and $\mathbf{X}$ but not $\mathbf{F}$: unsupervised kernel regression (UKR; [9]; software at `http://www.techfak.uni-bielefeld.de/~sklanke`), and Gaussian process latent variable model (GPLVM; [6]; software at `http://www.cs.man.ac.uk/~neill/gplvm`).

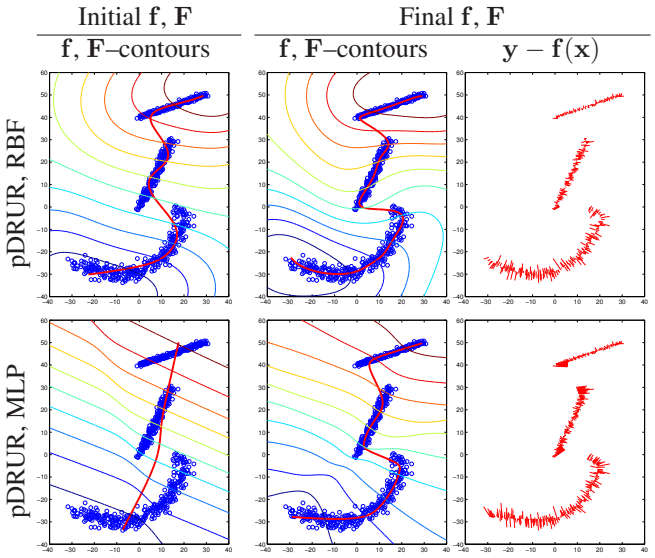


Figure 3. Noisy, disconnected 1D manifolds with $N = 700$ points in 2D. *Row 1*: pDRUR ($\mathbf{f}$: 15 RBFs; $\mathbf{F}$: 20 RBFs; $\lambda = 0.1$ for both), 20 iterations. *Row 2*: pDRUR with MLPs with a layer of 20 sigmoidal hidden units, 10 iterations. In all cases, the initial $\mathbf{X}$ were given by the first principal component of $\mathbf{Y}$.

Compared with nDRUR, we generally find that pDRUR can achieve a comparable quality (in reconstruction error and visually) with a much more parsimonious model and far faster. For example, in the spiral, clusters and small Swiss roll datasets we need only 10, 15 and 30 basis functions, while nDRUR would require 500, 700 and 1 000, respectively. For the larger datasets (e.g. the 100D Swiss roll with $N = 20\,000$ points), neither of nDRUR, UKR and GPLVM run in our workstation.

**Noisy spiral (fig. 2)** This dataset is very hard to solve using random or PCA initializations for $\mathbf{X}$, but easy if initializing from Isomap or LE. To find an intermediate level of difficulty, we use the true underlying $\mathbf{X}$ but perturbed with Gaussian noise, which creates controlled local disorder. Note how, because of the noise in $\mathbf{X}$ the initial $\mathbf{f}$ and $\mathbf{F}$ can be rather far from the manifold structure. Still, we found over multiple random replications that pDRUR (with either MLPs or RBFs) and the autoencoder reliably recover the manifold.

**Noisy disconnected curves (fig. 3)** This examines pDRUR's ability to work with clustered data where each cluster has manifold structure. The PCA initialization causes parts of the manifolds to fold over each other in $\mathbf{X}$, but pDRUR recovers a reasonable solution that preserves each manifold's structure. Note how the errors $\mathbf{y}-\mathbf{f}(\mathbf{x})$ over the points $\mathbf{x} \in \mathbf{X}$ are approximately orthogonal to the mapping $\mathbf{f}$. From eq. (8), equating the gradient to zero we obtain $-\mathbf{J}^T(\mathbf{y}-\mathbf{f}(\mathbf{x}))+\mathbf{x}-\mathbf{F}(\mathbf{y}) = \mathbf{0}$. If we learn a good mapping $\mathbf{F}$ then we will have $\mathbf{x} \approx \mathbf{F}(\mathbf{y})$ and so $\mathbf{J}^T(\mathbf{y} - \mathbf{f}(\mathbf{x})) = \mathbf{0}$ indeed. Also note how the contours of $\mathbf{F}$ (the loci of points $\mathbf{y}$ that map to the same $\mathbf{x}$) tend to meet $\mathbf{f}$ roughly orthogonally.

**Effect of regularization for RBFs (fig. 4)** For a 3D Swiss roll dataset with $N = 1\,000$ points, we varied systematically the regularization coefficients $\lambda_{\mathbf{f}}$ and $\lambda_{\mathbf{F}}$ ($10^{-5}$, $10^{-3}$, $10^{-1}$); the number of basis functions (30, 60, 90); and the noise level on $\mathbf{X}$ (Gaussian with stdev 5 or 10). We found little difference over the number of basis functions and the noise level (although, naturally, this depends on each dataset). Large $\lambda$ leads to smoother mappings and distorts the $\mathbf{X}$, as seen in the figure. In general, we find that using the smallest $\lambda$ that prevents ill-conditioning of the linear system for the weights works well. Note that UKR and GPLVM both failed with this noisy initial $\mathbf{X}$.

**Swiss roll (fig. 5, 6, 8)** Fig. 5 shows the result of UKR, GPLVM and pDRUR when initialized from a typical LE result for the Swiss roll; notice that, though global ordering exists, the initial $\mathbf{X}$ show strong local clustering and "holes" devoid of points, neither of which represent true structure in the manifold. UKR and GPLVM barely improve upon the initial $\mathbf{X}$, unlike pDRUR, which practically uniformizes the $\mathbf{X}$. UKR can also be trained with a homotopy algorithm but this failed to recover the Swiss roll.

Fig. 6 illustrates pDRUR's robustness to noise in the initial $\mathbf{X}$. Over multiple random replicates we have observed that pDRUR perfectly recovers the Swiss roll with stdev 20; as the noise increases, pDRUR's $\mathbf{X}$ increasingly show defects, but amazingly pDRUR is still able to recover the global ordering up to stdev 60. An autoencoder also works well with stdev 20, but starts losing global ordering at stdev 40 and (as shown) completely loses it at stdev 60. Finally, the figure shows a large-scale experiment where we lifted a Swiss roll with $N = 20\,000$ points into $D = 100$ dimensions by adding 97 noise dimensions. Almost perfect recovery is attained in the very first iteration.

Fig. 8 shows the Swiss roll with a hole. Isomap blows up holes (or low-density regions) because, as is well known, it is unable to preserve geodesic distances across holes. pDRUR goes a long way to recovering the true $\mathbf{X}$, while UKR and GPLVM barely improve it, or even worsen it (note several outlying $\mathbf{x}$ in GPLVM). One reason why pDRUR is able to improve the initial $\mathbf{X}$ is the existence of both mappings $\mathbf{f}$ and $\mathbf{F}$, which the objective function encourages to be the inverse of each other on the manifold. This prevents distant points $\mathbf{x}$ from being mapped to close $\mathbf{y}$'s, and distant points $\mathbf{y}$ from being mapped to close $\mathbf{x}$'s; having only $\mathbf{f}$ does not enforce the latter.

**Rotated MNIST digits (fig. 9)** We selected 220 digits '7' at random from the MNIST database ($28 \times 28$ greyscale images) and rotated each by 4–degree increments, to create a large dataset of $N = 19\,800$ images in $D = 784$ dimensions. Each of the 220 rotation sequences (see sample at top of figure) traces a closed loop in 784D space, but there

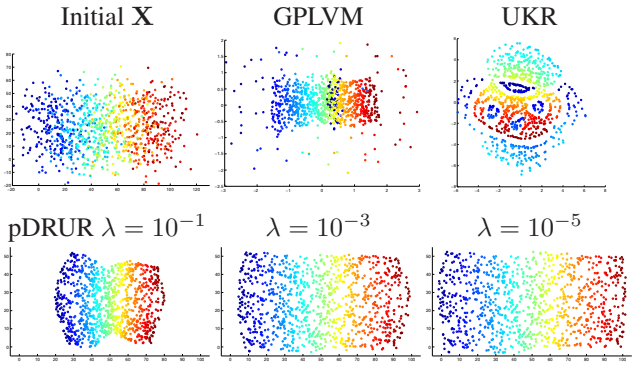


Figure 4. Swiss roll with $N = 1\,000$ points in 3D and Gaussian noise on $\mathbf{X}$ of stdev 10. *Row 1*: initial $\mathbf{X}$ and result by UKR and GPLVM. *Row 2*: result for pDRUR (30 RBFs, 100 iterations) and effect of regularization $\lambda$ (same for both $\mathbf{f}$ and $\mathbf{F}$).

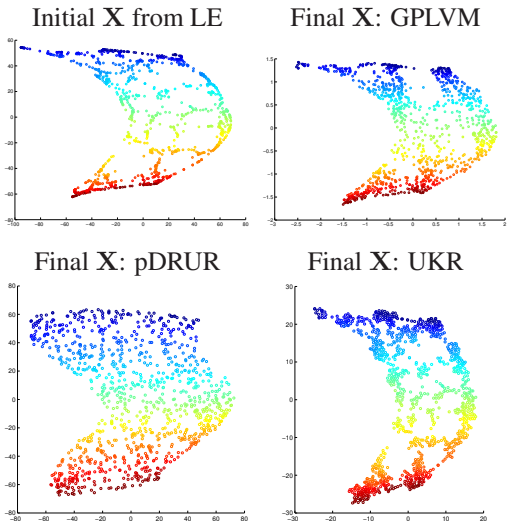


Figure 5. Swiss roll ($N = 1\,000$ points in 3D). Latent coordinates $\mathbf{X}$ found by different methods (pDRUR, GPLVM, UKR) from a Laplacian eigenmaps (LE) initialization.

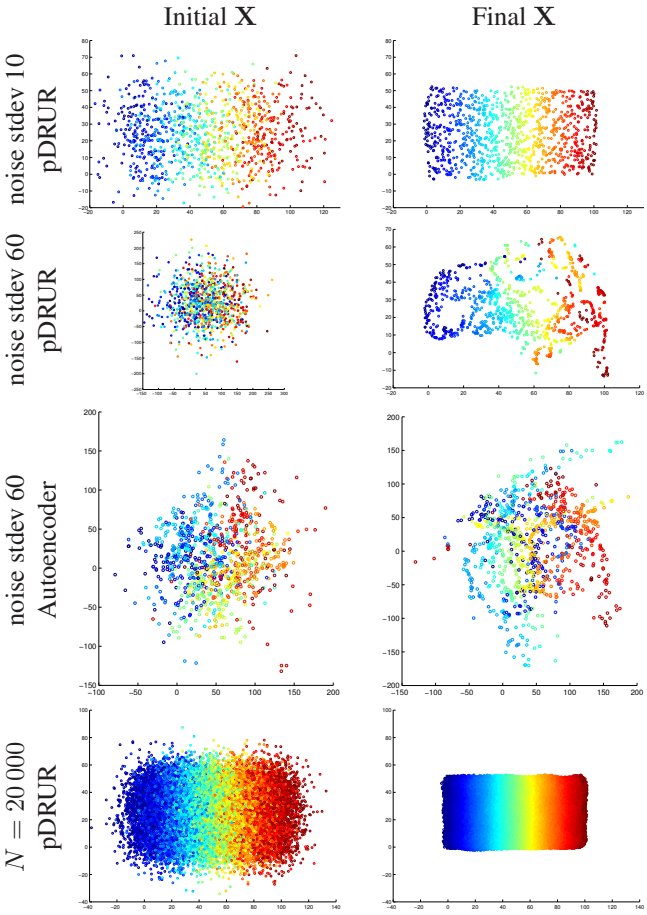


Figure 6. Swiss roll, initial and final results with pDRUR and autoencoders. *Row 1*: $N = 1\,000$ points in 3D, pDRUR (30 RBFs, $\lambda = 10^{-5}$, 100 iterations) initialized from true $\mathbf{X}$ with Gaussian noise with stdev 10 ($= \frac{1}{5}$ of the shorter side of the roll). *Row 2*: as row 1 but now $\mathbf{X}$ has stdev 60 and pDRUR has 70 RBFs. *Row 3*: as row 2 with a 70–2–70 autoencoder. *Row 4*: $N = 20\,000$ points in 100D (with 97 dimensions of Gaussian noise of stdev 0.5), initial $\mathbf{X}$ are true $\mathbf{X}$ plus Gaussian noise with stdev 10. In each row, the axes have the same scale.

is also large variation in the handwriting style of the digits. We show pDRUR's result when initialized from PCA. The initial $\mathbf{X}$ collapse the loops, but pDRUR is able to open them up and thus recover the rotation angle, as can be inferred from the color of each sequence in 2D, and from the projection with $\mathbf{F}$ of the test sequence in the right plot. The reconstruction with $\mathbf{f}$ clearly captures the rotation too (and using $> 2$ dimensions would recover less blurred images).

**Runtime comparison** Fig. 10 shows comparative run times (in seconds in a 2.66 GHz PC with 2 GB RAM) for an $N$-point $D$-dim Swiss roll (where $D - 3$ dimensions consist of zero-mean Gaussian noise) with noisy initialization as in fig. 4, for pDRUR (70 RBFs, 10 iterations), GPLVM (70 active points, 10 iterations), UKR (homotopy), nDRUR (10 iterations). Missing times in the graph correspond to running out of memory or taking too long. We use the authors' published Matlab code, and although Matlab runtime comparisons are unreliable, pDRUR's order-of-magnitudes' advantage is clear (for $N > 7\,000$ points, all other methods run out of memory, while pDRUR takes just 74 min. with $N = 100\,000$). Besides, pDRUR also gets an almost perfect reconstruction of the Swiss roll in all cases, while UKR and GPLVM do not (see fig. 4); and pDRUR's result barely changes after the first 2 iterations, while the other methods require many more to stabilize. The (asymptotic) slope of the runtime curves in the log-log plots shows that pDRUR is linear on $N$ and $D$ while the other methods are quadratic or superlinear on $N$.

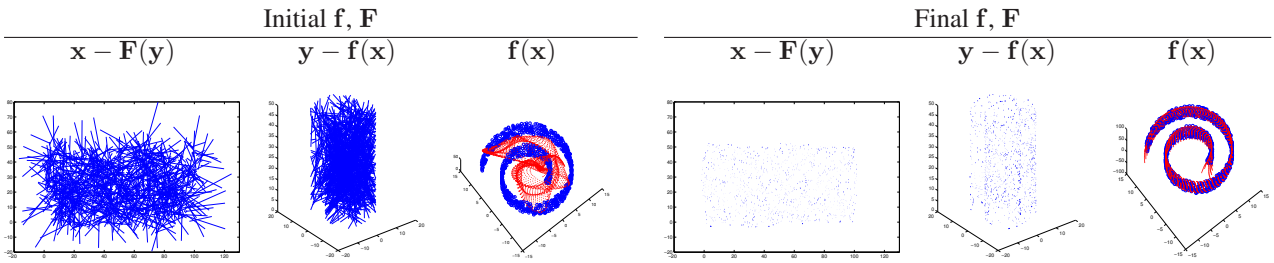


Figure 7. Initial and final pDRUR mappings for row 1 in fig. 6.

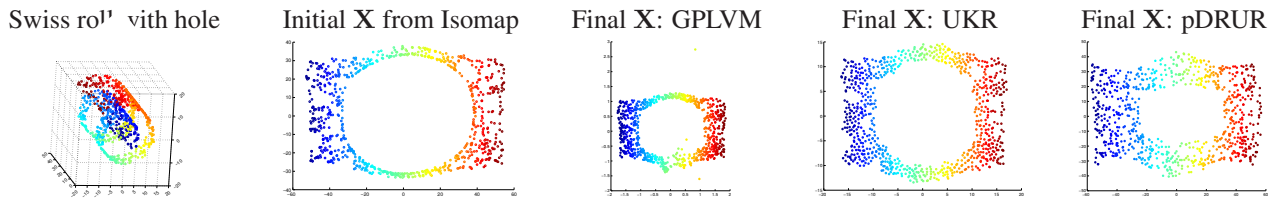


Figure 8. Swiss roll with a hole ($N = 791$ points in 3D): final $\mathbf{X}$ for methods initialized with Isomap's $\mathbf{X}$ (pDRUR: 30 RBFs).

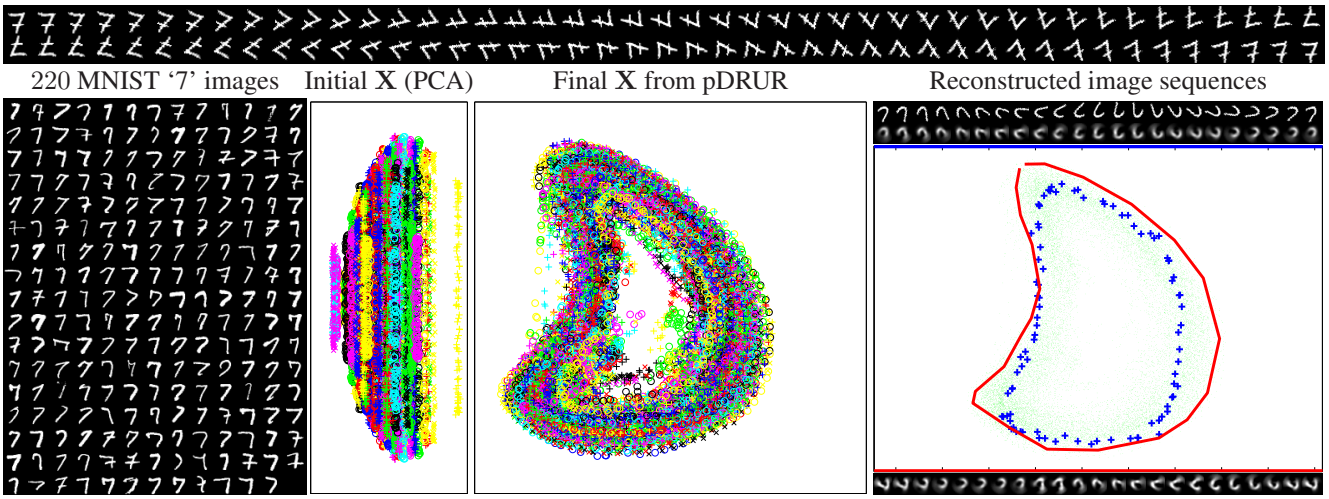


Figure 9. *Left*: 220 different MNIST $28 \times 28$ images, each rotated at 4–degree intervals totaling $N = 19\,800$ points in $D = 784$ dimensions (sample sequence above). pDRUR mapped this to 2D ($\mathbf{f}$: 5 RBFs, $\mathbf{F}$: 5 RBFs, $\lambda_{\mathbf{f}} = \lambda_{\mathbf{F}} = 0.01$, 200 iterations, $\mathbf{X}$ initialized by PCA). *Middle plots*: each rotation sequence is color- and marker-coded in latent space. *Right*: an out-of-sample path in latent space (red) and the images that $\mathbf{f}$ produces (below); and a test sequence of images, its 2D projection with $\mathbf{F}$ (blue $+$) and its reconstruction with $\mathbf{f} \circ \mathbf{F}$ (above).

## 4. Related work

Most work on unsupervised regression fits a function $\mathbf{f}$ and latent coordinates $\mathbf{X}$ to observed data. Its roots go back to factor analysis methods that estimate both the scores ($\mathbf{X}$) and the factor loadings ($\mathbf{f}$) by minimizing an error function alternatingly [19, 18], resulting in PCA; nonlinear factor analysis has also been applied in this way [8] (note the probabilistic model on $\mathbf{x}$ is thus lost). One of the definitions of principal curves [4] alternates between fitting a spline and minimizing the reconstruction error wrt $\mathbf{X}$. Other work has used different forms of $\mathbf{f}$: MARS [7], neural nets [16], and more recently in the machine learning literature RBFs [15], kernel regression [9] and Gaussian processes [6]. To project test points $\mathbf{y}$, either one fits $\mathbf{F}$ a posteriori to $(\mathbf{Y}, \mathbf{X})$, or minimizes the reconstruction error over $\mathbf{x}$ (which is expensive and prone to local optima). One problem with nonparametric forms for $\mathbf{f}$ is that the error can be driven to zero by separating infinitely apart the $\mathbf{X}$, and so these methods need to constrain the latter. The mapping $\mathbf{F}$ in nDRUR eliminates this problem.

Far less work exists on unsupervised regression to fit a function $\mathbf{F}$ and $\mathbf{X}$. This includes [12], which considers reducing the dimensionality of time series with applications to tracking; and [10], which defines $\mathbf{F}$ as kernel regression.

Finally, the joint estimation of $\mathbf{f}$, $\mathbf{F}$ and $\mathbf{X}$ seems to have

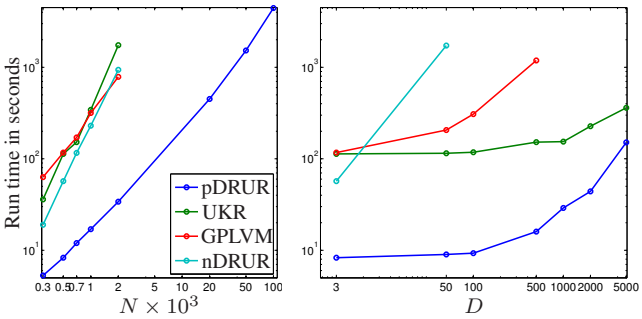

Figure 10. Log-log plot of the run time for different methods on a Swiss roll dataset with $N \leq 10^5$ points in $D = 3$ dimensions (left) and with $N = 500$ points in $D \leq 5\,000$ dimensions (right).

been proposed only recently. Besides the nonparametric DRUR formulation [3], Ranzato et al [14, 13] have proposed an objective function similar to (1), but designed to learn overcomplete ($L > D$), sparse codes rather than dimensionality reduction. They use penalty terms to encourage learning sparse codes, where only a few dimensions of $\mathbf{x}$ are non-zero for a given data vector $\mathbf{y}$; this is useful for learning parts-based representations (e.g. a handwritten digit as a combination of strokes). Their mapping $\mathbf{f}$ is linear and $\mathbf{F}$ is slightly nonlinear, and their parameters and the codes are optimized alternately using gradient descent.

Autoencoders can be derived from the pDRUR objective function by eliminating $\mathbf{x} = \mathbf{F}(\mathbf{y})$ and optimizing only over $\mathbf{f}$ and $\mathbf{F}$. Training autoencoders from random initializations is exceedingly slow (because the network is deep) and typically yields bad local minima. Recent work [5] has suggested that these problems decrease with a better initialization. In our experiments, we have found that pretraining $\mathbf{f}$ and $\mathbf{F}$ separately given a reasonably good $\mathbf{X}$ (e.g. from a spectral method) indeed works very well, although pDRUR seems significantly more robust to noise in $\mathbf{X}$.

## 5. Conclusion

We have proposed a parametric formulation of the DRUR method that results in a very efficient optimization, affording to apply the method to far larger datasets. Our parametric DRUR method appears to achieve solutions of a quality as high as those of nonparametric DRUR, significantly improving noisy or defective initializations (e.g. from a spectral method), sometimes dramatically so. Related methods we compared with do not show this degree of robustness. We conjecture that this is due to our effective optimization strategy and to the use of an enlarged search space over $\mathbf{X}$, which perhaps may facilitate escaping from local optima. The success of the algorithm strongly argues against out-of-sample extensions of spectral methods based on the latent coordinates $\mathbf{X}$ directly computed by the spectral method; a few pDRUR iterations (sometimes a single one) typically provide a much better $\mathbf{X}$ and consequently much better mappings. In summary, we believe that its scalability to larger datasets, its robustness, and its ability to provide both parametric projection and reconstruction mappings, make pDRUR an eminently applicable method for dimensionality reduction.

**Acknowledgments** This work was partially supported by NSF CAREER award IIS–0754089.

## References

[1] M. Belkin and P. Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15(6):1373–1396, June 2003.

[2] M. Á. Carreira-Perpiñán and Z. Lu. The Laplacian Eigenmaps Latent Variable Model. *AISTATS*, 2007.

[3] M. Á. Carreira-Perpiñán and Z. Lu. Dimensionality reduction by unsupervised regression. *CVPR*, 2008.

[4] T. J. Hastie and W. Stuetzle. Principal curves. *J. Amer. Stat. Assoc.*, 84(406):502–516, June 1989.

[5] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 2006.

[6] N. Lawrence. Probabilistic non-linear principal component analysis with Gaussian process latent variable models. *JMLR*, 2005.

[7] M. LeBlanc and R. Tibshirani. Adaptive principal surfaces. *J. Amer. Stat. Assoc.*, 89(425):53–64, Mar. 1994.

[8] R. P. McDonald. A general approach to nonlinear factor analysis. *Psychometrika*, 27(4):397–415, Dec. 1967.

[9] P. Meinicke, S. Klanke, R. Memisevic, and H. Ritter. Principal surfaces from unsupervised kernel regression. *IEEE PAMI*, 2005.

[10] R. Memisevic. Unsupervised kernel regression for nonlinear dimensionality reduction. Master's th., U. Bielefeld, 2003.

[11] J. Nocedal and S. J. Wright. *Numerical Optimization*, Springer, second edition, 2006.

[12] A. Rahimi, B. Recht, and T. Darrell. Learning to transform time series with a few examples. *IEEE PAMI*, 2007.

[13] M. Ranzato, Y.-L. Boureau, and Y. LeCun. Sparse feature learning for deep belief networks. *NIPS*, 2008.

[14] M. Ranzato, C. Poultney, S. Chopra, and Y. LeCun. Efficient learning of sparse representations with an energy-based model. *NIPS*, 2007.

[15] A. J. Smola, S. Mika, B. Schölkopf, and R. C. Williamson. Regularized principal manifolds. *JMLR*, 2001.

[16] S. Tan and M. L. Mavrovouniotis. Reducing data dimensionality through optimizing neural network inputs. *AIChE Journal*, 41(6):1471–1479, June 1995.

[17] J. B. Tenenbaum, V. de Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 2000.

[18] P. Whittle. On principal components and least square methods of factor analysis. *Skand. Aktur. Tidskr.*, 36, 1952.

[19] G. Young. Maximum likelihood estimation and factor analysis. *Psychometrika*, 6(1):49–53, Feb. 1940.