Nonlinear Low-Dimensional Regression Using Auxiliary Coordinates

Weiran Wang **Miguel Á. Carreira-Perpiñán**
EECS, School of Engineering, University of California, Merced

Abstract

When doing regression with inputs and outputs that are high-dimensional, it often makes sense to reduce the dimensionality of the inputs before mapping to the outputs. Much work in statistics and machine learning, such as reduced-rank regression, sliced inverse regression and their variants, has focused on linear dimensionality reduction, or on estimating the dimensionality reduction first and then the mapping. We propose a method where both the dimensionality reduction and the mapping can be nonlinear and are estimated jointly. Our key idea is to define an objective function where the low-dimensional coordinates are free parameters, in addition to the dimensionality reduction and the mapping. This has the effect of decoupling many groups of parameters from each other, affording a far more effective optimization than if using a deep network with nested mappings, and to use a good initialization from sliced inverse regression or spectral methods. Our experiments with image and robot applications show our approach to improve over direct regression and various existing approaches.

We consider the problem of *low-dimensional regression*, where we want to estimate a mapping between inputs $\mathbf{x} \in \mathbb{R}^{D_\mathbf{x}}$ and outputs $\mathbf{y} \in \mathbb{R}^{D_\mathbf{y}}$ that are both continuous and high-dimensional (such as images, or control commands for a robot), but going through a low-dimensional, or latent, space $\mathbf{z} \in \mathbb{R}^{D_\mathbf{z}}$: $\mathbf{y} = \mathbf{g}(\mathbf{F}(\mathbf{x}))$, where $\mathbf{z} = \mathbf{F}(\mathbf{x})$, $\mathbf{y} = \mathbf{g}(\mathbf{z})$ and $D_\mathbf{z} < D_\mathbf{x}, D_\mathbf{y}$. In some situations, this can be preferable to a *direct (full-dimensional) regression* $\mathbf{y} = \mathbf{G}(\mathbf{x})$, for example if, in addition to the regression, we are interested in obtaining a low-dimensional representation of $\mathbf{x}$ for its own sake (e.g. visualization or feature extraction). Even when the true mapping $\mathbf{G}$ is not low-dimensional, using a direct regression requires many parameters ($D_\mathbf{x} D_\mathbf{y}$ in linear regression) and their estimation may be unreliable with small sample sizes. Using a low-dimensional composite mapping $\mathbf{g} \circ \mathbf{F}$ with fewer parameters can be seen as a form of regularization and lead to better generalization with test data. Finally, a common practical approach is to reduce the dimension of $\mathbf{x}$ independently of $\mathbf{y}$, say with principal component analysis (PCA), and then solve the regression. However, the latent coordinates $\mathbf{z}$ obtained in this way do not necessarily preserve the information that is needed to predict $\mathbf{y}$. This is the same reason why one would use linear discriminant analysis rather than PCA to preserve class information. We want low-dimensional coordinates $\mathbf{z}$ that eliminate information in the input $\mathbf{x}$ that is not useful to predict the output $\mathbf{y}$, in particular to reduce noise. In this sense, the problem can be seen as *supervised dimensionality reduction*.

Consider then the problem of least-squares regression (although our arguments should apply to other loss functions). The simplest approach to estimate the dimensionality reduction mapping $\mathbf{F}$ and the regression mapping $\mathbf{g}$ is to minimize the objective function

$$E_1(\mathbf{F}, \mathbf{g}) = \sum_{n=1}^{N} \|\mathbf{y}_n - \mathbf{g}(\mathbf{F}(\mathbf{x}_n))\|^2 + \lambda_\mathbf{g} R(\mathbf{g}) + \lambda_\mathbf{F} R(\mathbf{F}) \quad (1)$$

given a training set $\mathbf{X}_{D_\mathbf{x} \times N}$, $\mathbf{Y}_{D_\mathbf{y} \times N}$, where $\lambda_\mathbf{F}, \lambda_\mathbf{g} \geq 0$ and R is a regularizer, e.g. penalizing large weights in a neural net or radial basis function (RBF) network. Indeed, the earliest method for low-dimensional regression, reduced-rank regression (RRR) (Anderson, 1951; Reinsel and Velu, 1998), takes both mappings as linear: $\mathbf{F} = \mathbf{B}\mathbf{x}$ and $\mathbf{g} = \mathbf{A}\mathbf{z}$ (assuming centered data). In RRR the global optimum of (1) is given by an eigenproblem for $\mathbf{A}$ and $\mathbf{B}$, although in some problems alternating optimization over $\mathbf{A}$ and $\mathbf{B}$ may be a better option computationally. Unfortunately, when both $\mathbf{F}$ and $\mathbf{g}$ are nonlinear, the optimization becomes very difficult. In order to have the property of universal approximation—practically speaking, for the mappings to be flexible enough—neural net architectures

Appearing in Proceedings of the 15^{th} International Conference on Artificial Intelligence and Statistics (AISTATS) 2012, La Palma, Canary Islands. Volume XX of JMLR: W&CP XX.

require at least one hidden layer of nonlinear units (e.g. sigmoidal for multilayer perceptrons, Gaussian for RBFs) (Bishop, 2006). Conceptually, computing the derivatives of (1) wrt (the parameters of) $\mathbf{F}$ and $\mathbf{g}$ is a simple application of the chain rule (backpropagation). However, these derivatives are very poorly scaled, with the gradient magnitudes being far smaller for the parameters at the innermost levels, which causes the Hessian of E_1 to be ill-conditioned; this worsens with the number of layers (Rögnvaldsson, 1994). First-order methods take tiny steps, slowly zigzagging down a curved valley, while full-fledged second-order methods have limited application because of the large size of the Hessian, which is not sparse. While a direct regression $\mathbf{G}$ would require only one layer of hidden units, in the low-dimensional composite mapping $\mathbf{g} \circ \mathbf{F}$ both $\mathbf{g}$ and $\mathbf{F}$ require one layer of hidden units each, leading to a deeper net. The same difficulty affects the use of autoencoders in unsupervised dimensionality reduction. In practice, this leads to very long training times and a parameter estimate which could well be far from a real minimum. An additional problem is that (1) has local optima, however there are various ways of constructing good initializations, such as using one of the linear methods (e.g. RRR or SIR), or a spectral method (e.g. Isomap; Tenenbaum et al., 2000). Recently, the technique of pretraining (Hinton and Salakhutdinov, 2006) has shown some promise.

It is then not surprising that most work on low-dimensional regression has avoided the nonlinear formulation (1). Besides RRR and related methods (such as partial least squares and canonical correlation analysis; Reinsel and Velu, 1998), an important line of work has sought to solve for the mapping $\mathbf{F}$ separately from $\mathbf{g}$ but using information from $\mathbf{y}$. Once $\mathbf{F}$ has been obtained, one can estimate $\mathbf{g}$ on the pairs $(\mathbf{y}_n, \mathbf{F}(\mathbf{x}_n))$ using any regression method. For example, sliced inverse regression (SIR) (Li, 1991) quantizes the data $\mathbf{Y}$ into "slices" or clusters, which in turn induce a quantization of the $\mathbf{x}$–space. Each $\mathbf{x}$–slice (all points $\mathbf{x}_n$ that map to the same $\mathbf{y}$–slice) is then replaced with its mean, and computing PCA on these means gives a linear $\mathbf{F}$. A kernelized version, KSIR (Wu, 2008), gives instead a nonlinear $\mathbf{F}$, the analogue of kernel PCA; see also Kim and Pavlovic (2008). Instead of the slicing condition, one can use a conditional independence criterion and achieve a linear $\mathbf{F}$ as in kernel dimensionality reduction (Fukumizu et al., 2004). The advantage of these approaches is that the solution for $\mathbf{F}$ can usually be obtained by solving a nonsparse eigenproblem or linear system in one shot, and $\mathbf{g}$ can be chosen and estimated independently afterwards. However, while in some special cases (see Li, 1991) slicing $\mathbf{y}$ is indeed enough to extract all the information needed to obtain an optimal $\mathbf{F}$, in general it does not seem possible to decouple $\mathbf{F}$ from $\mathbf{g}$ entirely—certainly not in the nonlinear case. Besides, achieving in practice a good slicing in high-dimensions with small sample sizes can be unreliable. Kernelization has the added disadvantage of having to solve eigenproblems with order N, which scales poorly and requires cumbersome approximations unless one uses small datasets.

We believe that nonlinear methods that jointly optimize over $\mathbf{g}$ and $\mathbf{F}$ hold the greatest potential. In this paper, we revisit the objective (1) but seek to approximate it by another objective having an easier optimization that is scalable (linear in N) and enables its practical use. This is an adaptation of the unsupervised method of Carreira-Perpiñán and Lu (2010).

1 Low-Dimensional Regression Using Auxiliary Coordinates

Much of the ill-conditioning in the objective function (1) is due to the deep nesting in the function $\mathbf{g} \circ \mathbf{F}$. If we let the low-dimensional coordinates $\mathbf{Z}_{D_\mathbf{z} \times N} = (\mathbf{z}_1, \dots, \mathbf{z}_N)$ be independent, auxiliary parameters to be optimized over, we unfold the squared error into two terms that decouple given $\mathbf{Z}$ and break the nesting:

$$E_2(\mathbf{F}, \mathbf{g}, \mathbf{Z}) = \sum_{n=1}^N \|\mathbf{y}_n - \mathbf{g}(\mathbf{z}_n)\|^2 + \lambda_\mathbf{g} R(\mathbf{g}) \quad (2)$$
$$+ \sum_{n=1}^N \|\mathbf{z}_n - \mathbf{F}(\mathbf{x}_n)\|^2 + \lambda_\mathbf{F} R(\mathbf{F}).$$

Now, every squared error involves only a shallow mapping; the parameters in $(\mathbf{F}, \mathbf{g}, \mathbf{Z})$ are equally scaled, which improves the conditioning of the problem; and the derivatives required are simpler (no chain rule). At first sight it may seem overkill to have one free parameter vector $\mathbf{z}_n$ per data point $(\mathbf{x}_n, \mathbf{y}_n)$, but as seen below we can use good initial values for them, and their optimization given the mappings is fast and very simple. The overall training cost is linear in N. We apply alternating optimization over $\mathbf{Z}$ and $(\mathbf{F}, \mathbf{g})$.

Optimization over F and g This decouples into two independent optimizations, one for $\mathbf{F}$ and one for $\mathbf{g}$. Each is equivalent to a direct regression but operating on a lower input or output dimension $D_\mathbf{z}$, and, crucially, using shallower functions than $\mathbf{g} \circ \mathbf{F}$, so we limit ill-conditioning, as desired. An added benefit, not negligible in practice, is that the derivatives required are simpler to compute (reducing the possibility of manual error or the overheads of automatic differentiation), and faster to compute. Although we could use a sigmoidal neural net, we use instead radial basis function (RBF) networks and some heuristics that make the step over $\mathbf{F}$ and $\mathbf{g}$ even faster. We describe $\mathbf{g}$; the case of $\mathbf{F}$ is analogous. Then $\mathbf{g}(\mathbf{z}) = \mathbf{W}\mathbf{\Phi}(\mathbf{z})$ with $M \ll N$ Gaussian RBFs $\phi_m(\mathbf{z}) = \exp(-\frac{1}{2}\|(\mathbf{z} - \boldsymbol{\mu}_m)/\sigma\|^2)$, including a bias term, and $R(\mathbf{g}) = \|\mathbf{W}\|^2$ is a quadratic regularizer on the weights. As usual with RBFs

(Bishop, 2006), we determine the centers $\boldsymbol{\mu}_m$ by k-means on $\mathbf{Z}$ (initialized at the previous iteration's centers), and the weights $\mathbf{W}$ have a unique solution given by a linear system. Other parameters (σ, λ, M) are cross-validated. The total cost is $\mathcal{O}(M(D_{\mathbf{y}} + D_{\mathbf{z}}))$ in memory and $\mathcal{O}(NM(M + D_{\mathbf{y}}))$ in training time, mainly driven by setting up the linear system for $\mathbf{W}$ (involving the Gram matrix $\phi_m(\mathbf{z}_n)$); solving it exactly is a negligible $\mathcal{O}(M^3)$ since $M \ll N$ in practice, and using an iterative solver initialized at the $\mathbf{W}$ value from the previous step can reduce the cost to $\mathcal{O}(M^2)$. (Note that for $\mathbf{F}$ we only need run k-means and factorize the linear system once and for all in the first iteration, since its input $\mathbf{X}$ does not change.)

The distribution of the coordinates $\mathbf{Z}$ changes dramatically in the first few iterations (see experiments), while the error decreases quickly, but after that $\mathbf{Z}$ changes little. This allows a heuristic speedup: after a few iterations (50 in our experiments) we stop running k-means for $\mathbf{g}$, and fix its centers permanently. From this moment on, the objective function essentially corresponds to having linear mappings applied to fixed vectors $\boldsymbol{\Phi}(\mathbf{z}_n)$ and $\boldsymbol{\Phi}(\mathbf{x}_n)$, and (see later) we conjecture it has a unique solution for $(\mathbf{g}, \mathbf{F}, \mathbf{Z})$. The worst-case risk of this heuristic is simply a slightly suboptimal estimate of the mappings; note the k-means step (customarily used in RBFs) is itself suboptimal.

Optimization over Z This represents an added task over the direct regression, but fortunately an easy one. For fixed $\mathbf{g}$ and $\mathbf{F}$, the optimization of (2) decouples over each $\mathbf{z}_n \in \mathbb{R}^{D_{\mathbf{z}}}$. Thus, instead of one large nonlinear minimization over $ND_{\mathbf{z}}$ parameters, we have N independent nonlinear minimizations each on $D_{\mathbf{z}}$ parameters, of the form (we omit the subindex n here):

$$\min_{\mathbf{z}\in\mathbb{R}^{D_{\mathbf{z}}}} E(\mathbf{z}) = \|\mathbf{y} - \mathbf{g}(\mathbf{z})\|^2 + \|\mathbf{z} - \mathbf{F}(\mathbf{x})\|^2. \quad (3)$$

Here, $\mathbf{z} \in \mathbb{R}^{D_{\mathbf{z}}}$ is the only free variable, and $\mathbf{x} \in \mathbb{R}^{D_{\mathbf{x}}}$, $\mathbf{y} \in \mathbb{R}^{D_{\mathbf{y}}}$ and the functions $\mathbf{g}$ and $\mathbf{F}$ are fixed. The squared-error form of $E(\mathbf{z})$ immediately suggests a Gauss-Newton approach, where we compute a positive definite approximation to its Hessian using only first derivatives (equivalently, we linearize $\mathbf{g}$):

$$\nabla E(\mathbf{z}) = 2(-\mathbf{J}^T(\mathbf{z})(\mathbf{y} - \mathbf{g}(\mathbf{z})) + \mathbf{z} - \mathbf{F}(\mathbf{x})) \quad (4)$$

$$\nabla^2 E(\mathbf{z}) \approx 2(\mathbf{I} + \mathbf{J}(\mathbf{z})^T\mathbf{J}(\mathbf{z})) \quad (5)$$

where $\mathbf{J}(\mathbf{z}) = \frac{1}{\sigma^2}\sum_{m=1}^{M} \mathbf{w}_m\phi_m(\mathbf{z})(\boldsymbol{\mu}_m - \mathbf{z})^T$ is the Jacobian of $\mathbf{g}$. Since our Hessian approximation is strictly positive definite (because of the $\mathbf{I}$ term) and the Jacobian is bounded, theorem 10.1 in Nocedal and Wright (2006) guarantees convergence under usual line-search conditions (e.g. Wolfe) to a local minimizer no matter the initial point. The convergence order is linear but much faster than (conjugate) gradient methods. The second-order term we discard in the true Hessian has the form $\sum_{d=1}^{D_{\mathbf{y}}}(y_d - g_d(\mathbf{z}))\nabla^2 g_d(\mathbf{z})$, so it is small when either our model is good (small residual $\mathbf{y} - \mathbf{g}(\mathbf{z})$) or it has small curvature ($\nabla^2 g_d(\mathbf{z})$). Either case will result in a good convergence rate. So, the new iterate along the Gauss-Newton search direction is $\tilde{\mathbf{z}} = \mathbf{z} + \alpha\mathbf{p}$ with $\mathbf{p} = (\mathbf{I} + \mathbf{J}^T\mathbf{J})^{-1}(\mathbf{J}^T(\mathbf{y} - \mathbf{g}(\mathbf{z})) - \mathbf{z} + \mathbf{F}(\mathbf{x}))$. In practice we find that the full Gauss-Newton step $\alpha = 1$ is nearly always accepted, except in early iterations, so the line search is rarely run; and 1–2 Gauss-Newton steps are usually enough to decrease the relative error to around 10^{-4}. The cost per step for a single $\mathbf{z}_n$ is $\mathcal{O}(D_{\mathbf{z}}^2 D_{\mathbf{y}})$ times the cost of computing one entry of the Jacobian (the cost term $D_{\mathbf{z}}^3$ from solving the linear system can be ignored because $D_{\mathbf{z}} < D_{\mathbf{y}}$). This is just $D_{\mathbf{z}}$ times the cost of computing the gradient. Thus, thanks to the Hessian information, we achieve a far faster convergence than simple gradient methods at barely any extra cost, again a nice consequence of introducing low-dimensional coordinates in the objective. The cost over all $\mathbf{Z}$ is then $\mathcal{O}(ND_{\mathbf{z}}^2 D_{\mathbf{y}})$; in practice this costs less than the step over $\mathbf{g}$ and $\mathbf{F}$. The N optimizations could be run in parallel.

Initialization One advantage of having the auxiliary low-dimensional coordinates $\mathbf{Z}$ as free parameters is that we can use many good ways to initialize them (rather than using random values), ranging from purely unsupervised dimensionality reduction (PCA, Isomap and other spectral methods), possibly run on $\mathbf{X}$ or on $(\mathbf{X}, \mathbf{Y})$, to sliced inverse regression, reduced-rank regression and related methods.

Other Choices of Mappings In some applications it may be convenient to make $\mathbf{g}$, $\mathbf{F}$ or both linear. Linear $\mathbf{F}$ can be particularly useful with very high-dimensional inputs. Indeed, much earlier work has focused on estimating (in one shot) a linear $\mathbf{F}$ and then estimating a nonlinear $\mathbf{g}$. Another interesting possibility that is especially efficient in our method is to take $\mathbf{g}$ as linear and $\mathbf{F}$ as an RBF network, thus achieving a nonlinear dimensionality reduction and feature extraction. Since the basis functions for $\mathbf{F}$ are trained once and for all in the first iteration, the model $\mathbf{F}$ becomes a linear mapping operating on inputs $\boldsymbol{\Phi}(\mathbf{x}_n)$, so the steps over $(\mathbf{F}, \mathbf{g})$ and $\mathbf{Z}$ are all quadratic and have a unique solution given by solving a linear system. In fact, we conjecture the objective function in this case has a unique global optimum (up to symmetries). It is also possible to let $\mathbf{g}$ and $\mathbf{F}$ be nonparametric. The step over $(\mathbf{F}, \mathbf{g})$ has a unique solution given by a basis function expansion where each data point $\mathbf{x}_n$ or $\mathbf{z}_n$ is a center (so no k-means is needed). However, the step over $\mathbf{Z}$ becomes far more complex (Carreira-Perpiñán and Lu, 2008).

Bias with Respect to (1) Our formulation does introduce one problem: the optima of (2) are not necessarily optima of (1) and vice versa. Thus, our com-

posite mapping $\mathbf{g} \circ \mathbf{F}$ is not optimal in the usual sense. However, it is easy to see from the gradients that if $\mathbf{z}_n = \mathbf{F}(\mathbf{x}_n)$ for all n at an optimum of (2) then $\mathbf{g} \circ \mathbf{F}$ is indeed optimal in the usual sense of (1). In practice we observe $\mathbf{z}_n$ is always very close to $\mathbf{F}(\mathbf{x}_n)$, suggesting the bias is not important. Our experiments show our algorithm achieves better results than a range of low-dimensional regression methods, and better or comparable than direct regression. We think the computational ease of our algorithm far compensates for the slight suboptimality with respect to (1).

We see three ways of decreasing the bias if desired. One could optimize (1) from our $\mathbf{F}$ and $\mathbf{g}$ (discarding $\mathbf{Z}$). However, as discussed in the introduction, the point is that optimizing (1) is very slow. This option treats our algorithm as a fast way to get a very good initialization. It is similar in spirit to pretraining (Hinton and Salakhutdinov, 2006), but the latter is even slower and obtains $\mathbf{F}$, $\mathbf{g}$ that are less refined because they are trained as unsupervised Boltzmann machines rather than to minimize the prediction error. A second option is to run a fast postprocessing step where one fits $\mathbf{F}$ to $(\mathbf{X}, \mathbf{Z})$, discards $\mathbf{Z}$, fits $\mathbf{g}$ to $(\mathbf{F}(\mathbf{X}), \mathbf{Y})$, and returns the resulting $\mathbf{F}$ and $\mathbf{g}$. It is easy to see that this always decreases the error (1). By comparing with training the deep net in experiments, we show this option practically eliminates the (anyway small) bias, producing near-optimal $\mathbf{F}$, $\mathbf{g}$ much faster. A third option we leave for future work is to drive the $\mathbf{F}$–term to zero in a graduated way during the optimization, thus ensuring we obtain an optimum of (1) while never optimizing the deep $\mathbf{g} \circ \mathbf{F}$ (effectively, applying a quadratic-penalty method to minimizing the $\mathbf{g}$–error s.t. the constraints $\mathbf{F}(\mathbf{X}) = \mathbf{Z}$).

Cross-validation and Number of Parameters Given a test point $\mathbf{x}$, to project it to latent space and to predict its output we simply compute $\mathbf{z} = \mathbf{F}(\mathbf{x})$ and $\mathbf{y} = \mathbf{g}(\mathbf{z})$, resp. This also follows from a missing-data point of view (Carreira-Perpiñán and Lu, 2011): if we augment the training set $\mathbf{X}$ with $\mathbf{x}$ and minimize (2) over the unknowns $\mathbf{z}$ and $\mathbf{y}$, keeping $\mathbf{F}$ and $\mathbf{g}$ fixed, we equivalently minimize $E(\mathbf{y}, \mathbf{z}) = \|\mathbf{y} - \mathbf{g}(\mathbf{z})\|^2 + \|\mathbf{z} - \mathbf{F}(\mathbf{x})\|^2$, which yields the result above. Thus, our predictive function is $\mathbf{g} \circ \mathbf{F}$, the same as in (1), which we use for cross-validating the user parameters: regularization parameter λ, width σ, number of basis functions M (for $\mathbf{F}$, $\mathbf{g}$), and the dimensionality of $\mathbf{z}$. Note our approach introduces no additional user parameters over those already existing in the formulation (1).

The number of parameters in our low-dimensional regressor $\mathbf{g} \circ \mathbf{F}$ (weights and centers only, we ignore the auxiliary parameters $\mathbf{Z}$, which are not used for testing) is $\mathcal{O}(M_{\mathbf{F}}(D_{\mathbf{x}} + D_{\mathbf{z}}) + M_{\mathbf{g}}(D_{\mathbf{y}} + D_{\mathbf{z}}))$. For a direct RBF network from $\mathbf{x}$ to $\mathbf{y}$ with M basis functions we have $\mathcal{O}(M(D_{\mathbf{x}} + D_{\mathbf{y}}))$ parameters. If using linear mappings, the low-dimensional case has $\mathcal{O}(D_{\mathbf{z}}(D_{\mathbf{x}} + D_{\mathbf{y}}))$ and the direct one $\mathcal{O}(D_{\mathbf{x}} D_{\mathbf{y}})$. The runtime for a test point is proportional to the number of parameters in all cases. The low-dimensional regressor will have fewer parameters when $D_{\mathbf{z}}$ is sufficiently smaller than $D_{\mathbf{x}}$ and $D_{\mathbf{y}}$, and depending on how many basis functions are used.

2 Experimental Evaluation

Our figure of merit is the prediction error with respect to the ground truth. We determine the user parameters for the direct and low-dimensional regression by cross-validation. We initialize $\mathbf{Z}$ from PCA or Isomap trained on the joint $(\mathbf{X}, \mathbf{Y})$ (which tends to work better than just $\mathbf{X}$) and use the alternating optimization of section 1 with early stopping on a validation set, which in our datasets happened at around 100 iterations. Compared to the direct regression, we achieve a lower or equal error while using fewer or the same number of parameters. All other methods do worse than ours and often worse than the direct regression. We compare our method with two direct regressors $\mathbf{y} = \mathbf{G}(\mathbf{x})$: an RBF network and a Gaussian process (GP, trained with the software of Rasmussen and Williams, 2005); and with two low-dimensional regressors $\mathbf{y} = \mathbf{g}(\mathbf{F}(\mathbf{x}))$ for which either software is available (KSIR; Wu, 2008) or results in comparable datasets are published (Kim and Pavlovic, 2008). We also tried estimating $\mathbf{F}$ from unsupervised dimensionality reduction (PCA, Isomap) but the results (not shown) were far worse. Finally, note that our problems have $D_{\mathbf{y}} > D_{\mathbf{z}}$, otherwise the dimensionality reduction problem $\mathbf{z} = \mathbf{F}(\mathbf{x})$ is no easier than the prediction problem $\mathbf{y} = \mathbf{G}(\mathbf{x})$.

Rotated MNIST Digits '7' We selected 40 digits '7' at random from the MNIST database (28×28 grayscale images, so $D_{\mathbf{x}} = 784$), added some noise, and rotated each by 6–degree increments from 0 to 360 degrees, to create a dataset of $N = 2\,400$ images in $D = 784$ dimensions. The output $\mathbf{y}$ of a digit with rotation θ was created by rigidly rotating by θ (a linear map) a fixed upright digit–7 point set in 2D (14 points so $D_{\mathbf{y}} = 28$), yielding a skeletonized version of the image at the corresponding rotation (fig. 1). Thus, the composite mapping goes through a bottleneck of dimension $D_{\mathbf{z}} = 2$ given by the Cartesian coordinates of the angle. We do validation on another 600 images (fig. 2) and test on 3000 images. The $\mathbf{Z}$ points found by Isomap (using $k = 20$ neighbors and the known dimensionality $D_{\mathbf{z}} = 2$) when run on the $(\mathbf{X}, \mathbf{Y})$ dataset capture the loop nature of the data, but images with the same rotation have widely separated $\mathbf{z}$, because there is also a large variation in the handwriting style of the digits (fig. 3 left); the same effect appears in fig. 9 of Carreira-Perpiñán and Lu (2010). The results

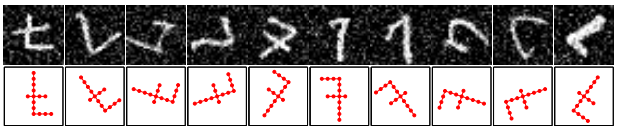


Figure 1: Training subset of noisy rotated MNIST '7' (input image $\mathbf{x}$ and output skeleton $\mathbf{y}$).

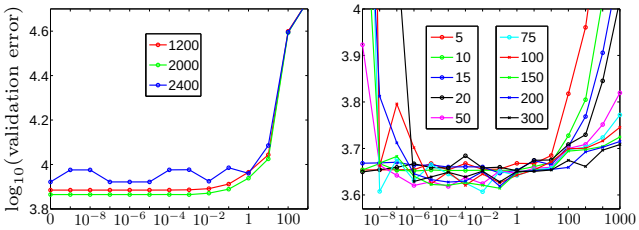


Figure 2: Cross-validation curves for our algorithm on rotated MNIST '7'. *Left*: $\mathbf{F}$ RBF, $\mathbf{g}$ linear. Each curve corresponds to a different number of centers for F ($\lambda_{\mathbf{F}} = 10^{-2}$), and the validation is over $\lambda_{\mathbf{g}}$. *Right*: both $\mathbf{F}$, $\mathbf{g}$ RBF. We fix $M_{\mathbf{F}} = 1\,200$ centers and $\lambda_{\mathbf{F}} = 10^{-2}$. The validation is over the $M_{\mathbf{g}}$ and $\lambda_{\mathbf{g}}$.

of KSIR and kernel PCA are no better. Our algorithm (results after 16 iterations) turns the initial $\mathbf{Z}$ from Isomap into a nearly 1D loop by collapsing images with the same rotation irrespective of their individual style (since these all map to the same skeleton). Using RBF networks for $\mathbf{F}$ and $\mathbf{g}$, our method achieves the lowest test error, far better than one-shot low-dimensional regressors (e.g. KSIR), and better than direct regression with GP or RBF networks, while using fewer parameters (table 1, fig. 3 right).

Scratched MNIST Digits We consider all MNIST digit types now (commonly assumed to lie in a low-dimensional manifold because of variations in slant, thickness, style, etc.) and add scratches to the original images of random orientation and thickness. We want to predict the corresponding scratch-free image. We picked 2 000 images (200 for each digit 0–9) for training, 2 000 for validation and 2 000 for test. Thus we map from $D_{\mathbf{x}} = 784$ to $D_{\mathbf{y}} = 784$. Notice that the scratches greatly alter the Euclidean distances in the 784D input space, which results in poor performance of the nearest neighbor regressor on this data set. We find that, with a latent dimension of only $D_{\mathbf{z}} = 60$, we do as well or slightly better than direct regression error with RBF. Further confirmation that our 60D features are good is how much they improve the nearest neighbor regressor (using distances in latent space).

Our algorithm, with both $\mathbf{F}$ and $\mathbf{g}$ RBF networks, uses the same number of parameters as the direct regression. We set the parameters as follows. We first cross-validate the direct RBF regression to find the number of centers and regularization. We then train our algorithm with the same centers and regularization for $\mathbf{F}$,

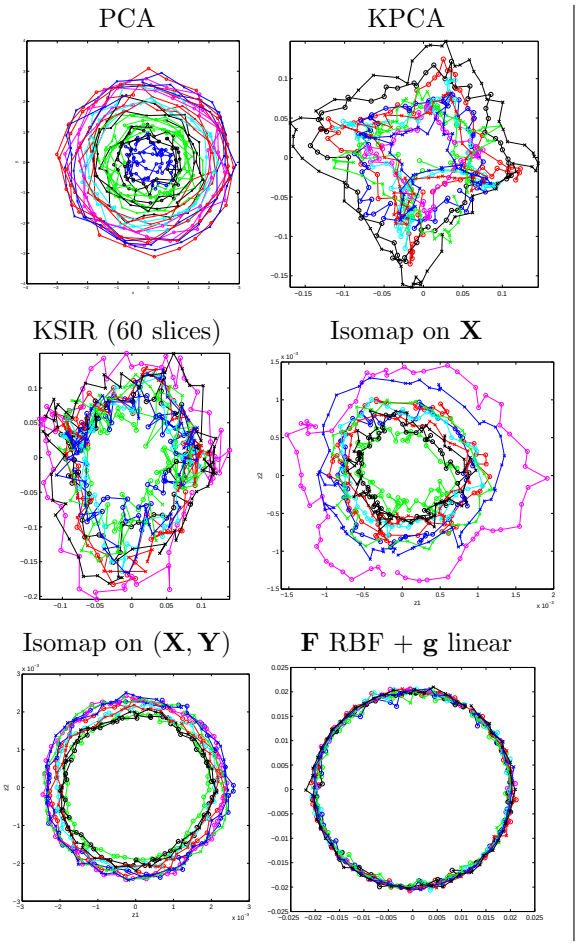
Method	SSE
direct linear	51 710
direct RBF (1200 centers)	32 495
direct RBF (2000 centers)	29 525
Gaussian process	29 208
KPCA (3) + RBF (2000, 1)	49 782
KSIR (60, 26) + RBF(20, 10^{-5})	39 421
Ours: $\mathbf{F}$ RBF $(2000, 10^{-2})$ + $\mathbf{g}$ linear (10^{-3})	29 612
Ours: $\mathbf{F}$ RBF $(1200, 10^{-2})$+$\mathbf{g}$ RBF $(75, 10^{-2})$	27 346

Table 1: Sum of squared errors on rotated MNIST '7' test set, with optimal parameters coded as RBF (M, λ), KPCA (Gaussian kernel width), KSIR (number of slices, Gaussian kernel width). Number of iterations for our method: 16 (linear $\mathbf{g}$), 7 (RBF $\mathbf{g}$).

and the same regularization for $\mathbf{g}$. We run 5 times for each parameter setting for different random k-means initializations to find the centers of $\mathbf{g}$. Finally, we pick the $\mathbf{F}$ and $\mathbf{g}$ with the smallest regression error for $\mathbf{g} \circ \mathbf{F}$ as before. We find that errors on both direct RBF and our low-dimensional regression algorithm are very similar. We also ran a nearest neighbor regressor based on the extracted 60D features. For each test point $\mathbf{x}$, we obtain a low-dimensional feature $\mathbf{F}(\mathbf{x})$, and find nearest neighbors among the training set low-dimensional projections $\mathbf{F}(\mathbf{x})$ or $\mathbf{z}$. The regression errors are shown in fig. 4. The relative improvement for the nearest neighbor regressor is 39.4%, which can also be seen in the sample images shown on the right. Note that Kim and Pavlovic (2008), using similarly scratched USPS digits, barely improved over nearest neighbor regressor, and did not compare with a direct RBF regression.

Serpentine Robot Forward Kinematics In practice, robots are redundant by design, i.e., they have more degrees of freedom than strictly required to perform a task, because this facilitates planning. For example, a redundant robot arm has many angle configurations to grasp a given object, to improve the chances that some of them remain feasible in the presence of obstacles in workspace. This naturally gives rise to low-dimensional regression settings. Serpentine robots are of particular interest due to their high maneuverability, but they also have complex kinematics. Although the manufacturer provides an analytical forward kinematics mapping that predicts the workspace position of the end-effector given joint angles, it deviates from the actual one because of the effect of e.g. loads, and must be calibrated (or learned).

We consider here a planar simulated serpentine robot (fig. 6 left) consisting of a chain of 10 articulated links of equal length, with the first link anchored at the origin. Its "head" or end-effector is thus constrained to move in a plane, but it holds a "camera" (represented by the 3D coordinates of its 8 corners) through a fixed-



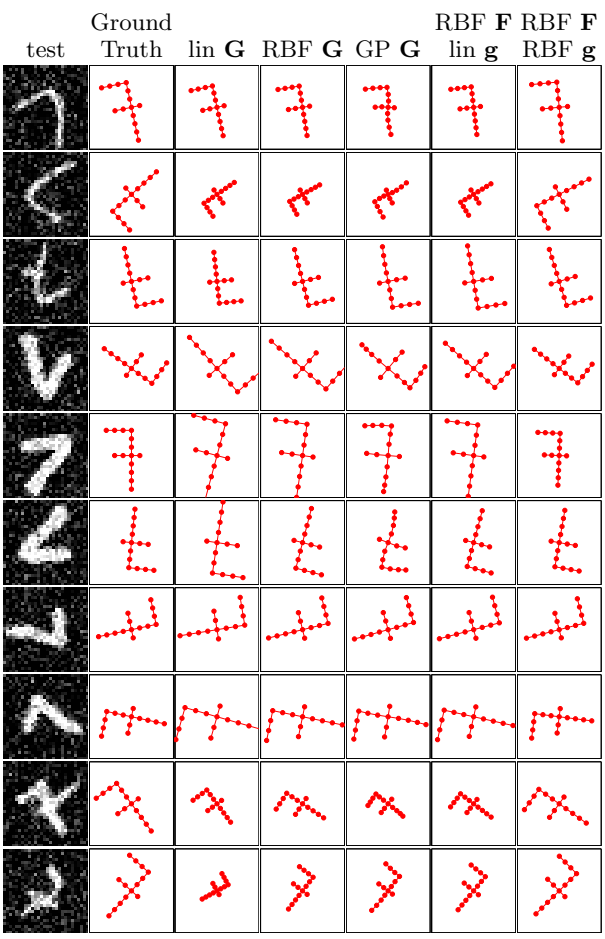


Figure 3: *Left*: embeddings $\mathbf{Z}$ from different methods for MNIST '7'. Each curve (only 10 plotted to avoid clutter) corresponds to one original image and its continuously rotated versions. *Right*: sample test predictions by different methods. GT: ground truth, $\mathbf{G}$: direct regression (linear, RBF net, Gaussian process), $(\mathbf{F}, \mathbf{g})$: our method.

length link that it can orient in 3D space. Thus, we have a mapping from $D_{\mathbf{x}} = 12$ dimensions (the joint angles) to $D_{\mathbf{z}} = 4$ dimensions (the 2D position and 2D orientation of the end-effector) to $D_{\mathbf{y}} = 24$ dimensions (the camera configuration). We assume we have recorded $N = 2\,000$ pairs of angles and camera configurations while the robot moved around. We used Isomap ($k = 20$) as initial $\mathbf{Z}$. Cross-validation in our method clearly showed the correct latent dimensionality (fig. 5), not seen in Isomap's residual variance, and achieved a quite lower test error than direct regression with RBF or GP mappings (table 2). Fig. 6 (right) shows the latent parameterization found correlates one-to-one with the ground-truth one; the colors of column i code our method's coordinate z_i for $i = 1$ to 4.

Comparison with optimizing nested model $\mathbf{g} \circ \mathbf{F}$ We repeated the robot experiment by directly minimizing the nested model (1) without auxiliary coordinates. All conditions were identical: same data (training, validation, test), model size and initial $\mathbf{Z}$ (so same initial mappings fitted to $(\mathbf{X}, \mathbf{Z})$ and $(\mathbf{Z}, \mathbf{Y})$); early stopping (training stops when the validation error (1)

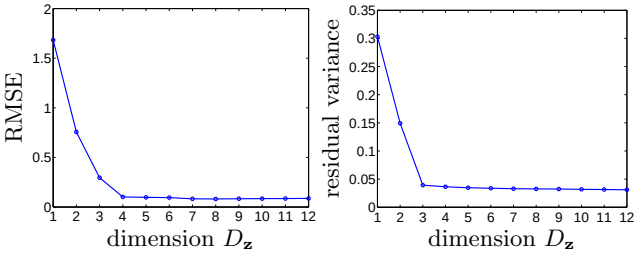


Figure 5: Cross-validation of intrinsic dimensionality for serpentine robot forward kinematics. *Left*: validation over $D_{\mathbf{z}}$ for our algorithm. *Right*: residual variance of Isomap on $(\mathbf{X}, \mathbf{Y})$ ($k = 20$ nearest neighbors).

increases); running on a single core of a PC. We used alternating optimization of (1): the step over $\mathbf{g}$ for fixed $\mathbf{F}$ is identical to ours, a regression on $(\mathbf{F}(\mathbf{X}), \mathbf{Y})$; the step over $\mathbf{F}$ uses conjugate gradients and a good line search (C. Rasmussen's `minimize`); L-BFGS did about as well. Other variations worked far worse (optimizing jointly all parameters, or using gradient descent with or without momentum). The large number of parameters in $\mathbf{W}_{\mathbf{F}}$ ($M = 2000$ basis functions so 8 000

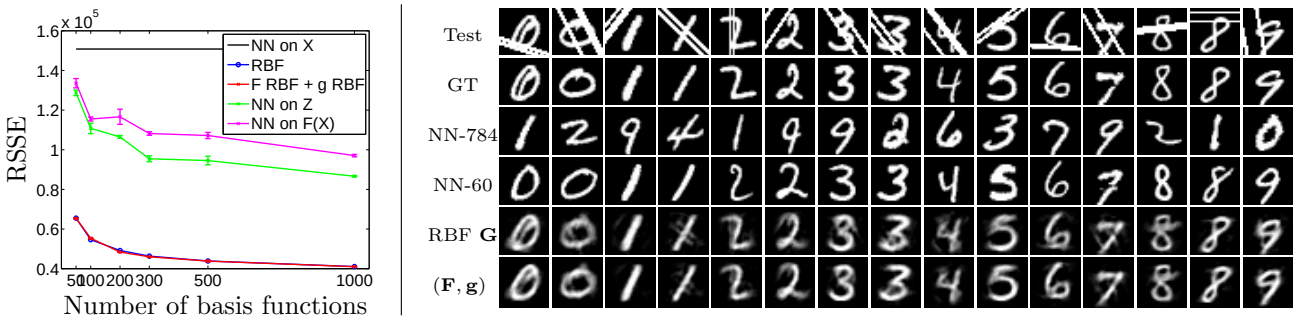


Figure 4: Scratched MNIST digits results. *Left*: prediction error for our method (lower curve, barely distinguishable from direct regression) and for a nearest neighbor regressor on input $\mathbf{x}$ directly (black line) and on our latent features $\mathbf{F}(\mathbf{x})$ or $\mathbf{z}$ (magenta, green curves). Errorbars over 5 runs of our method using different k-means initialization. *Right*: sample test reconstructions by different methods. GT: ground truth $\mathbf{y}$, NN-784 and NN-60: nearest neighbor regressor in original and reduced space, RBF $\mathbf{G}$: direct RBF regression, $(\mathbf{F}, \mathbf{g})$: our method.

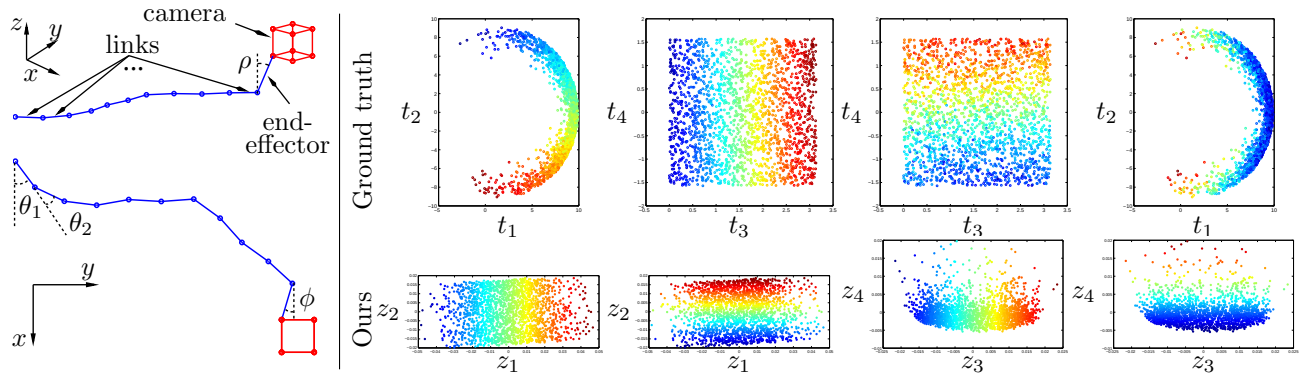


Figure 6: *Left*: illustration of a serpentine robot configuration from two views. *Right*: correspondence between the ideal 4D latent representation that generated the data set (first row) and dimension reduction achieved by our algorithm (second row). When we move in one dimension (z_1, z_2, z_3, and z_4 for the four columns respectively) in our latent space (color varying from blue to red), we observe a corresponding motion in the ideal latent space.

Method	RMSE
direct regression, linear (reg. 0)	2.2827
direct regression, RBF (2000, 10^{-6})	0.3356
direct regression, Gaussian process	0.7082
KPCA (2.5) + RBF ($400, 10^{-10}$)	3.7455
KSIR (400, 100) + RBF ($1000, 10^{-8}$)	3.5533
Ours: $\mathbf{F}$ RBF ($2000, 10^{-6}$)+$\mathbf{g}$ RBF ($100, 10^{-9}$)	0.1006

Table 2: Root mean squared error (per corner of the camera) of the output $\mathbf{y}$ on a test set of serpentine robot forward kinematics. Methods coded as in table 1.

weights) makes it hard to use second-order methods.

Fig. 7 shows the error (1) on training and test. Training the nested model proceeds slowly as expected. It takes 1885 iterations and ≈ 3 hours runtime to reach early stopping. Training with our algorithm quickly decreases the error, with early stopping occurring after 34 iterations and 200 seconds runtime. The bias there is very small compared with the nested convergence error. Already stopping here gives a near-optimal model in $\frac{1}{50}$th of the runtime; at that point, the nested model's error is far bigger, closer to the initial one. We then switch to the nested training in order to remove the bias. This practically occurs in the very first iteration, which simply fits $\mathbf{g}$ to $(\mathbf{F}(\mathbf{X}), \mathbf{Y})$ (eliminating $\mathbf{Z}$). Further training proceeds at its usual slow rate and has little bias left to eliminate. If we let our algorithm converge on the training set instead, the bias is bigger, but its decrease in the first iteration after the switch is also bigger and the final error is comparable.

3 Discussion

We have motivated our approach as a way to unfold the squared error with a deep mapping in (1) into two squared errors with shallower mappings, equalizing the scaling of the parameters and thus reducing ill-conditioning. Alternating optimization is simple and fast because the mappings decouple given the low-dimensional coordinates, and the $\mathbf{Z}$–step decouples over all coordinates given the mappings. Alternating optimization is much slower if applied to $(\mathbf{F}, \mathbf{g})$ in (1) because the parameters are then coupled (Nocedal and Wright, 2006) and $\mathbf{F}$ is nonlinearly nested inside $\mathbf{g}$. As evidenced by our experiments, our objective function

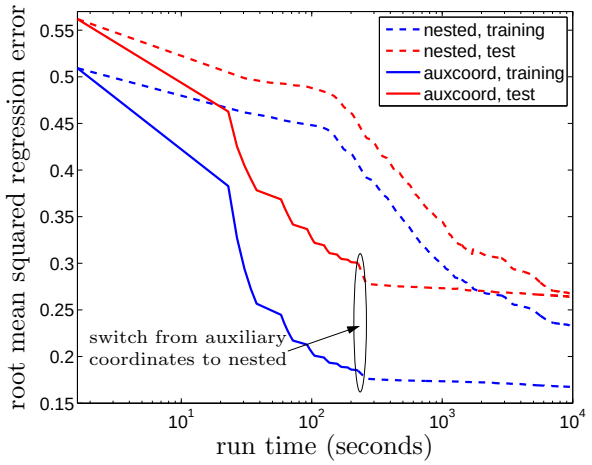


Figure 7: Learning curves for the nested objective (1) as a function of runtime, for the nested training (dashed) vs auxiliary coordinates (solid), on the training (blue) and test (red) datasets. Validation error curves similar to test not shown to avoid clutter.

achieves a very good model with early stopping; the small bias there is almost eliminated with an extra iteration over $\mathbf{g}$ alone (removing $\mathbf{Z}$). This achieves an error comparable to that of the nested model but over an order of magnitude faster. As with other biased but fast learning algorithms, such as contrastive divergence (Carreira-Perpiñán and Hinton, 2005), the ability to get a pretty good model pretty fast (and the option to eliminate the bias at extra cost) is practically useful.

In a sense, the one-shot methods described in the introduction can be seen as doing a single iteration of our approach, since they first get $\mathbf{Z}$ or $\mathbf{F}$ (usually from an eigenproblem) and then they fit $\mathbf{g}$ to $(\mathbf{Z}, \mathbf{Y})$. For example, in sliced inverse regression (SIR) and related methods one first fits $\mathbf{F}$ based on certain information that involves $\mathbf{y}$. Then, $\mathbf{F}$ is kept fixed and its outputs $\mathbf{z} = \mathbf{F}(\mathbf{x})$ are used to fit $\mathbf{g}$. (In our method, $\mathbf{F}$ and $\mathbf{g}$ can be updated in parallel.) Unless a perfect decoupling between $\mathbf{F}$ and $\mathbf{g}$ and thus perfect coordinates $\mathbf{Z}$ could be achieved, which seems only possible in very special cases, it would make sense to improve the result by iterating the process to fit $\mathbf{F}$–then–$\mathbf{g}$, as in our approach (although SIR lacks a joint objective function over $\mathbf{F}$ and $\mathbf{g}$). Otherwise, the estimation of $\mathbf{g}$ is stuck with suboptimal inputs $\mathbf{Z}$. We can then classify low-dimensional regression methods along an axis that measures how much the outputs $\mathbf{y}$ influence the dimensionality reduction $\mathbf{F}$: unsupervised dimensionality reduction for $\mathbf{F}$ (e.g. PCA or Isomap) on the inputs $\mathbf{X}$ alone, ignoring $\mathbf{Y}$ entirely; unsupervised dimensionality reduction for $\mathbf{F}$ on the joint $(\mathbf{X}, \mathbf{Y})$; SIR and related methods that learn $\mathbf{F}$ using some information from $\mathbf{Y}$, fix it and learn $\mathbf{g}$ given $\mathbf{F}$; and methods that jointly optimize an objective function where $\mathbf{F}$ and $\mathbf{g}$ are coupled, such as (1) and (2).

Our approach is close in spirit to unsupervised regression methods for dimensionality reduction such as Smola et al. (2001); Meinicke et al. (2005) and in particular Carreira-Perpiñán and Lu (2008, 2010, 2011) (see also Ranzato et al., 2007). Here one has inputs $\mathbf{X}$ but no outputs $\mathbf{Y}$, and one lets the unobserved projections of $\mathbf{X}$ in latent space be separate variables to be optimized over. However, without the information provided by $\mathbf{Y}$, the latent space found by them will not generally be optimal for regression.

A problem related to low-dimensional regression is manifold alignment (e.g. Ham et al., 2005). The latter is usually formulated as finding a single latent space that is shared by different high-dimensional spaces (without specifying any as inputs or outputs), in a semi-supervised setting (given only a few corresponding pairs across spaces), and using spectral methods (e.g. graph Laplacians). In our fully supervised setting this seems less useful, although if we had additional unlabeled inputs (or outputs) it may make sense to use manifold alignment to initialize $\mathbf{Z}$ in our method.

4 Conclusion

Allowing both the dimensionality reduction and the regression mappings to be nonlinear is a potentially very powerful approach to low-dimensional regression, but it creates a deep mapping with nested layers and long-range couplings between parameters that makes training very difficult. We have proposed a new objective function by introducing low-dimensional coordinates to decouple both mappings, thus working on a larger search space that is better conditioned. This still achieves very good nonlinear solutions but is now easier to optimize, and allows exact or second-order steps even with high dimensions and large datasets. Training time is linear in the sample size, with efficient steps that cause large changes to the initialization in a few iterations. Our results far outperform one-shot methods such as (kernel) sliced inverse regression in the quality of the model achieved, and direct learning of the nested mapping in training speed. In our view, truly nonlinear methods for dimensionality reduction have been unfairly neglected in the literature and we hope our work shows they can be used in practice.

Here we have taken $\mathbf{F}$, $\mathbf{g}$ to be shallow mappings (with just one layer of hidden units), because this is sufficient to obtain universal approximators, and at present it is not clear that deeper mappings are superior. However, our approach should apply to deep nets by making the intermediate unit activations at each layer be auxiliary coordinates and optimizing over them as well.

Acknowledgments
MACP thanks Stefano Carpin for valuable discussions. Work funded by NSF CAREER award IIS–0754089.

References

T. W. Anderson. Estimating linear restrictions on regression coefficients for multivariate normal distributions. *Annals of Mathematical Statistics*, 22(3): 327–351, Sept. 1951.

C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer Series in Information Science and Statistics. Springer-Verlag, Berlin, 2006.

M. Á. Carreira-Perpiñán and G. E. Hinton. On contrastive divergence learning. In R. G. Cowell and Z. Ghahramani, editors, *Proc. of the 10th Int. Workshop on Artificial Intelligence and Statistics (AISTATS 2005)*, pages 33–40, Barbados, Jan. 6–8 2005.

M. Á. Carreira-Perpiñán and Z. Lu. Dimensionality reduction by unsupervised regression. In *Proc. of the 2008 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'08)*, Anchorage, AK, June 23–28 2008.

M. Á. Carreira-Perpiñán and Z. Lu. Parametric dimensionality reduction by unsupervised regression. In *Proc. of the 2010 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'10)*, pages 1895–1902, San Francisco, CA, June 13–18 2010.

M. Á. Carreira-Perpiñán and Z. Lu. Manifold learning and missing data recovery through unsupervised regression. In *Proc. of the 12th IEEE Int. Conf. Data Mining (ICDM 2011)*, pages 1014–1019, Vancouver, BC, Dec. 11–14 2011.

K. Fukumizu, F. R. Bach, and M. I. Jordan. Dimensionality reduction for supervised learning with reproducing kernel Hilbert spaces. *Journal of Machine Learning Research*, 5:73–99, Jan. 2004.

J. Ham, D. Lee, and L. Saul. Semisupervised alignment of manifolds. In R. G. Cowell and Z. Ghahramani, editors, *Proc. of the 10th Int. Workshop on Artificial Intelligence and Statistics (AISTATS 2005)*, pages 120–127, Barbados, Jan. 6–8 2005.

G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, July 28 2006.

M. Kim and V. Pavlovic. Dimensionality reduction using covariance operator inverse regression. In *Proc. of the 2008 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'08)*, Anchorage, AK, June 23–28 2008.

K.-C. Li. Sliced inverse regression for dimension reduction. *J. Amer. Stat. Assoc.*, 86(414):316–327 (with comments, pp. 328–342), June 1991.

P. Meinicke, S. Klanke, R. Memisevic, and H. Ritter. Principal surfaces from unsupervised kernel regression. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 27(9):1379–1391, Sept. 2005.

J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer-Verlag, New York, second edition, 2006.

M. Ranzato, C. Poultney, S. Chopra, and Y. LeCun. Efficient learning of sparse representations with an energy-based model. In B. Schölkopf, J. Platt, and T. Hofmann, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 19, pages 1137–1144. MIT Press, Cambridge, MA, 2007.

C. E. Rasmussen and C. Williams. *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning Series. MIT Press, Cambridge, MA, 2005.

G. C. Reinsel and R. P. Velu. *Multivariate Reduced-Rank Regression. Theory and Applications*. Number 136 in Lecture Notes in Statistics. Springer-Verlag, 1998.

T. Rögnvaldsson. On Langevin updating in multilayer perceptrons. *Neural Computation*, 6(5):916–926, Sept. 1994.

A. J. Smola, S. Mika, B. Schölkopf, and R. C. Williamson. Regularized principal manifolds. *Journal of Machine Learning Research*, 1:179–209, June 2001.

J. B. Tenenbaum, V. de Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, Dec. 22 2000.

H.-M. Wu. Kernel sliced inverse regression with applications to classification. *Journal of Computational and Graphical Statistics*, 17(3):590–610, Sept. 2008.

A Appendix: Supplementary Material

A.1 Proof that the postprocessing step always reduces the bias

We show that the postprocessing step described in section 1 (under "Bias with Respect to (1)") always reduces the bias. Define the nested objective function

$$E_1(\mathbf{F},\mathbf{g}) = \sum_{n=1}^{N} \|\mathbf{y}_n - \mathbf{g}(\mathbf{F}(\mathbf{x}_n))\|^2 + \lambda_{\mathbf{g}} R(\mathbf{g}) + \lambda_{\mathbf{F}} R(\mathbf{F}) \tag{1'}$$

and the auxiliary-coordinates objective function (where $\mathbf{Z} = (\mathbf{z}_1, \dots, \mathbf{z}_n)$)

$$\begin{aligned} E_2(\mathbf{F},\mathbf{g},\mathbf{Z}) = {} & \textstyle\sum_{n=1}^{N} \|\mathbf{y}_n - \mathbf{g}(\mathbf{z}_n)\|^2 + \lambda_{\mathbf{g}} R(\mathbf{g}) \\ & + \textstyle\sum_{n=1}^{N} \|\mathbf{z}_n - \mathbf{F}(\mathbf{x}_n)\|^2 + \lambda_{\mathbf{F}} R(\mathbf{F}). \end{aligned} \tag{2'}$$

Given a point $(\mathbf{Z}_2, \mathbf{F}_2, \mathbf{g}_2)$ in the space of E_2, we define the *postprocessing step* as obtaining the point $(\mathbf{F}_1, \mathbf{g}_1)$ in the space of E_1, where $\mathbf{F}_1 = \mathbf{F}_2$ and $\mathbf{g}_1$ is fit to minimize the regression error

$$\min_{\mathbf{g}} \sum_{n=1}^{N} \|\mathbf{y}_n - \mathbf{g}(\mathbf{z}'_n)\|^2 + \lambda_{\mathbf{g}} R(\mathbf{g}) \tag{6}$$

where $\mathbf{g}$ is initialized from $\mathbf{g}_2$ and $\mathbf{z}'_n = \mathbf{F}_2(\mathbf{x}_n)$, $n = 1, \dots, N$. That is, the postprocessing step discards $\mathbf{Z}$ and corrects $\mathbf{g}$ by fitting it to $(\mathbf{F}_2(\mathbf{X}), \mathbf{Y})$.

Theorem A.1. *The postprocessing step decreases the objective E_1 or leaves it unchanged.*

Proof. We have:

$$\begin{aligned} & E_1(\mathbf{F}_1, \mathbf{g}_1) = \\ & = \sum_{n=1}^{N} \|\mathbf{y}_n - \mathbf{g}_1(\mathbf{F}_1(\mathbf{x}_n))\|^2 + \lambda_{\mathbf{g}} R(\mathbf{g}_1) + \lambda_{\mathbf{F}} R(\mathbf{F}_1) \\ & \le \sum_{n=1}^{N} \|\mathbf{y}_n - \mathbf{g}_2(\mathbf{F}_2(\mathbf{x}_n))\|^2 + \lambda_{\mathbf{g}} R(\mathbf{g}_2) + \lambda_{\mathbf{F}} R(\mathbf{F}_2) \\ & = E_1(\mathbf{F}_2, \mathbf{g}_2) \end{aligned}$$

since $\mathbf{F}_1 = \mathbf{F}_2$ and $\mathbf{g}_1$ minimizes (6) starting from $\mathbf{g}_2$. □

In practice, the point $(\mathbf{Z}_2, \mathbf{F}_2, \mathbf{g}_2)$ typically comes from minimizing E_2, possibly with early stopping on E_1 (i.e., exiting the minimizer when the error E_1 on a validation set of points $(\mathbf{x}, \mathbf{y})$ increases). Because $(\mathbf{F}_2, \mathbf{g}_2)$ was reached by minimizing E_2 and not E_1, it is not optimal w.r.t. E_1. Thus the postprocessing step always results in a point $(\mathbf{F}_1, \mathbf{g}_1)$ that improves over $(\mathbf{F}_2, \mathbf{g}_2)$. This is true even if we reduce but not completely minimize (6).