High-Dimensional Function Approximation Using Local Linear Embedding

Peter Andras
School of Computing and Mathematics
Keele University
Keele, Staffordshire, UK
p.andras@keele.ac.uk

***Abstract*—Neural network approximation of high-dimensional nonlinear functions is difficult due to the sparsity of the data in the high-dimensional data space and the need for good coverage of the data space by the 'receptive fields' of the neurons. However, high-dimensional data often resides around a much lower dimensional supporting manifold. Given that a low dimensional approximation of the target function is likely to be more precise than a high-dimensional approximation, if we can find a mapping of the data points onto a lower-dimensional space corresponding to the supporting manifold, we expect to be able to build neural network approximations of the target function with improved precision and generalization ability. Here we use the local linear embedding (LLE) method to find the low-dimensional manifold and show that the neural networks trained on the transformed data achieve much better function approximation performance than neural networks trained on the original data.**

Keywords—function approximation; high dimension; local linear embedding; neural network

I. INTRODUCTION

Approximation of high-dimensional functions with neural networks is difficult [1-5]. On one side the data covers the data space very sparsely due to the high dimensionality of this space. On the other side the good coverage of the data space with neuronal 'receptive fields' (i.e. the support set of the neuron's activation function where the value of this function is not close to constant) is also difficult. Either the receptive fields have to be very large, which reduces the discriminative power of the neurons, or there have to be very many neurons with smaller size 'receptive fields'.

Often the data points in the high-dimensional space reside around a much lower-dimensional manifold [1,5,6]. In principle, this manifold is not known and has to be found experimentally through analyzing the data. The reason for the existence of the low-dimensional data supporting manifold in principle is that the data is generated by a process that implies a number of possibly nonlinear correlations and associations between the components of the data vectors effectively reducing the dimensionality of the manifold around which the data resides.

There are a number of methods for calculating the lower-dimensional manifold and for projecting the original data points onto a corresponding low dimensional space [1,6-8]. Principal component analysis [1,8] is commonly used to determine the dimensionality of the data, and for example, self-organizing maps [7] or local linear embedding [6] can be used to find a mapping of the data into a low dimensional space that matches the manifold around which the original data points reside.

Considering that the increase of the data dimensionality reduces the approximation performance of neural networks of comparable structural complexity [1-5], and that the data often resides on a low-dimensional manifold, we aim to approximate the function defined on the high-dimensional data using a low-dimensional projection of the data points in order to improve the approximation performance, e.g. mean squared error on the test data set. To do this first we need to build the low-dimensional projection of the data space and then train a neural network over this data space and using the function values calculated for the original data points. We need to build the low dimensional projection of the data such that this extends to the unseen test data as well allowing the calculation of the projections for both the training and test data.

In this paper we use the local linear embedding [6] method to calculate the low-dimensional projection of the data points. We use the same size single hidden layer neural networks with Gaussian activation functions to learn the target functions both in the original data space and in the low-dimensional projection space. The results show that in-line with the expectations the approximation performance of the neural networks trained on the low-dimensional projection space are much better than the approximation performance of the neural networks trained on the original data space. This result supports the use of the combination of unsupervised low-dimensional projection learning with the supervised neural network learning for the purpose of approximation of functions defined on high dimensional data.

The rest of the paper is structured as follows. First we review the relevant related work, namely the approximation of functions with neural networks and the dimension reduction methods. Next we describe in detail the proposed combined use of local linear embedding and supervised training of feed-forward neural networks. Then we present the results and discuss these and their implications. Finally the paper is closed by the conclusions section.

II. Related Work

A. *Neural network approximation of functions*

The theoretical properties of neural network approximation of functions are well established [8-14]. In principle, any continuous function and any function that can be approximated arbitrarily correctly by continuous functions (e.g. step functions) can be approximated arbitrarily correctly by neural networks with a single hidden layer of neurons with nonlinear activation function, for a range of such activation functions (e.g. Gaussian functions, sigmoidal functions). The underpinning theoretical results are usually based on some variant of the Stone-Weierstrass theorem on linear closures of function spaces or on integral representation theorems [8-14].

The key drawback of the theoretical results about the approximation properties of neural networks is that the required number of hidden layer neurons to achieve a given level of precision can be very large. In many cases this number grows exponentially with the dimensionality of the data [12-16]. This means that in practical terms often it is easier and more robust to build good approximations of functions by multiple hidden layer neural networks with relatively few neurons than to follow the theoretical approximation property results and build single hidden layer neural networks. Approaches like extreme learning [17] that rely on very large single hidden layer neural networks in general suffer from excessive structural complexity (i.e. too many neurons and parameters) that makes them potentially non-robust and ready to fit the noise in the data.

Recent works [5,18] show that projections of high-dimensional data points onto a low-dimensional space that corresponds to the manifold around which the data points reside allows to build neural networks that can approximate the original target functions. It is also shown [5,18] that these approximations are likely have better generalization ability than those achieved by building comparable complexity neural networks while using the original data points from the high-dimensional space. These results are important since they show that an initial step of dimension reduction can improve considerably the approximation performance of neural networks trained to approximate target functions defined on high-dimensional data. It should be noted however, that these results imply that the target function is effectively defined around the supporting manifold of the data points and converges to zero as we depart from this manifold. The neural networks obtained through the projection of the data into the low-dimensional space do not generalize well into parts of the original space that are far from the supporting manifold of the training data. However, if we assume that indeed the data resides on such supporting manifolds then the latter constraints does not have any undesirable impact on the approximation properties and performance of the resulting neural networks.

B. *Dimension reduction methods*

As we noted already, high-dimensional data is often arranged around a low-dimensional manifold embedded into the data space. Usually this manifold constitutes a high-dimensional curved nonlinear surface, making the low-dimensional mapping or projection of the data non-trivial. Also, usually the manifold and its true dimensionality is not known in advance.

Dimension reduction methods aim to find the low-dimensional projection of the data points from the manifold in the high-dimensional space onto a low-dimensional space that matches the dimensionality of this manifold [1,6-8,19-21]. Some methods assume the manifold to have a given dimensionality, while others may also include the determination of the estimated dimensionality of the manifold, e.g. principal component analysis [8].

In the simplest case, the data is assumed to reside on a linear sub-space of the data space and principal component analysis is used to determine the dimensionality and the orthogonal basis vectors of this sub-space [8]. This algorithm can be applied in a local sense as well and can be generalized to consider non-linear surfaces as well [21].

One general approach to dimension reduction is the use of self-organizing maps [5,7,20]. Self-organizing maps (SOM) form a Voronoi tessellation of the data space and project each tessellation unit onto a projection vector in the projection space. The SOM adapts the set of vectors that determine the Voronoi tessellation such that the arrangement of the corresponding projection vectors in the projection space matches the topological organization of the data space. The SOM approach assumes that the dimensionality of the projection space is known in advance. The method can be extended to consider non-linear Voronoi tessellations as well using the kernel function approach that transforms the data first into a function space and applies the linear Voronoi tessellation there [22].

Another general approach is to calculate the local linear embedding (LLE) of the data points [6]. This method uses the local neighborhood of each data point to calculate a linear combination of the neighboring data points that approximates the data point. Then these linear weights are used to calculate a low-dimensional mapping of the data points such that the local neighborhood structure is preserved. This is done by retaining of the linear combination weights of the local neighbors and requiring that the approximation of the projected data points by the linear combination of their neighbors is preserved. This method also requires the setting in advance of the dimensionality of the projection space. This method is similar to some extent to the multi-dimensional scaling [19] with the difference that here the local linear weights and linear combination approximation are aimed to be preserved instead of the distances between the data points.

There are a number of other methods used for dimension reduction, a review of these can be found in [19].

III. Function Approximation through Local Linear Embedding

As noted above, our aim is to approximate functions defined on high-dimensional data. Formally, we aim to approximate a function $f:\boldsymbol{R}^n \rightarrow \boldsymbol{R}$, having a sample of data point function value pairs $(\boldsymbol{x}^k, y_k)$, $k=1,m$, where $f(\boldsymbol{x}^k)=y_k$, or more generally, assuming additive measurement errors as well $f(\boldsymbol{x}^k+\boldsymbol{\xi}^k)=y_k+z_k$, where $\boldsymbol{\xi}^k$ is a random vector and z_k is a random number, both following usually a zero mean normal distribution with small value variance and covariance matrix, respectively.

The default option is to approximate the function in the original data space using a neural network with a single hidden layer having neurons with nonlinear (e.g. Gaussian) activation function. In this case we look for the approximation of the target function f in the form of

$$g(\boldsymbol{x})=\Sigma_{i=1,p} w_i \varphi(\boldsymbol{x};\boldsymbol{\theta}_i) \qquad (1)$$

where w_i are linear weights, $\boldsymbol{\theta}_i$ are parameter vectors of the basis functions φ that are used as the activation functions of the hidden neurons, and p is the number of neurons in the neural network. In this context the number of neurons, p, is the structural complexity measure of the neural network.

Note that this approximation of f will approximate the value of this function at all points $\boldsymbol{x}\in\boldsymbol{R}^n$ through interpolation / extrapolation based on the given values of the target function at the sample data points. I.e. the assumption of this approximation is that the target function has defined non-constant values (in general) at least for the points that are within the convex hull of the set of the sample data points. This assumption requires that the set of the basis functions $\varphi(\boldsymbol{x};\boldsymbol{\theta}_i)$ is such that the support sets of their non-constant valued part cover at least the full convex hull of the data points and its immediate neighborhood. Because of the high-dimensionality of the data space this requirement means either that there has to be a large number of such basis functions and consequently a large number of hidden neurons or that the support sets of non-constant parts of the basis functions has to be relatively large. In the first case the structural complexity of the neural network will be high, which will make it prone for over-fitting, i.e. fitting the noise in the data. In the second case the coverage of the data space will be coarse, which makes the approximation prone to under-fitting, i.e. averaging out local variations in the values of the target function.

The number of data points required to achieve comparable sample density increases exponentially with the dimensionality of the space – i.e. if $q>1$ and assuming that the data points sample the n-dimensional cube with edges of length q, the number of data points to provide the same density sample for different n values is q^n. Thus a given size (m) sample of data points provides a coarser sample of the data space as the dimensionality of the data points increases. This combined with the above reasoning about the possibility of under- and over-fitting implies that the default option of approximation of the target function in the high-dimensional data space is likely to have relatively low approximation performance in terms of generalization error.

Assuming that the data points reside around a low-dimensional manifold embedded into the high-dimensional space we can try to find an approximation of the target function restricted to the data manifold. This implies that the approximation over the manifold does not extend by default to other parts of the high-dimensional data space outside of the data manifold. Given that we are really interested in the approximation of the target function only over the data manifold we may simply define an extension of the approximating function to the rest of the high-dimensional data space by some more-or-less natural extension of the approximating function that converges quickly to constant zero outside of the data manifold.

To approximate the target function over the data manifold we need to find this manifold, which in general is not known a priori. This means that in general we do not have an analytical definition of the data manifold and we have to define it experimentally by analyzing the data and by building some approximation of it. In order to learn the target function over the data manifold in general we need first to transform this into a matching unfolded space and approximate the target function over this space.

The approximation of the target function over the data manifold needs in the first step the projection of the data points onto a space that matches the dimensionality of the manifold and the projection should be such that the topological structure of the data manifold is preserved through the projection. The preservation of the topological structure of the data manifold is important in order to learn a valid generalization of the approximated target function beyond the known data points on the data manifold.

To put this formally, first we need to find a projection of the data space $\pi:\boldsymbol{R}^n \rightarrow \boldsymbol{R}^d$, such that for the data manifold $M\subset\boldsymbol{R}^n$ we have that $\forall \boldsymbol{x},\boldsymbol{y}\in M$, if $||\pi(\boldsymbol{x})- \pi(\boldsymbol{y})||<\varepsilon$ then $||\boldsymbol{x}- \boldsymbol{y}||<\delta(\varepsilon)$ for any sufficiently small ε. In addition to this it is also important that the projection retains sufficient discrimination ability between possible data points, i.e. $\forall \boldsymbol{x},\boldsymbol{y}\in M$, if $||\boldsymbol{x}- \boldsymbol{y}||>\varepsilon'$ then $||\pi(\boldsymbol{x})-\pi(\boldsymbol{y})||>\delta'(\varepsilon')$ for sufficiently large ε'. Below the level of discrimination the approximation may lead to a locally constant approximation of the target function, e.g. if for a given $\boldsymbol{x}\in M$ we have that $\forall \boldsymbol{y}\in M$, if $||\boldsymbol{x}-\boldsymbol{y}||<\varepsilon''$, $\pi(\boldsymbol{x})=\pi(\boldsymbol{y})$ then the approximation of the target function for all these $\boldsymbol{y}\in M$ will be a constant function. Note that in principle it is sufficient if π is defined on and around M, which itself is defined by the part of the high-dimensional space where the data points reside – beyond a sufficiently wide neighborhood of M the π may be defined by some default extension of the π defined on M and its neighborhood.

Let us assume that we manage to find a projection $\pi:\boldsymbol{R}^n \rightarrow \boldsymbol{R}^d$ as described above. The revised version of the function approximation task in the low-dimensional space $\boldsymbol{R}^d$ is to approximate the function $h:\boldsymbol{R}^d \rightarrow \boldsymbol{R}$, having a sample of data point function value pairs $(\pi(\boldsymbol{x}^k),y_k)$, $k=1,m$, where $h(\pi(\boldsymbol{x}^k))=f(\boldsymbol{x}^k)=y_k$, or assuming additive measurement errors as well $h(\pi(\boldsymbol{x}^k+\boldsymbol{\xi}^k))=f(\boldsymbol{x}^k+\boldsymbol{\xi}^k)=y_k+z_k$, where $\boldsymbol{\xi}^k$ is a random vector and z_k is a random number, following zero mean normal distributions with small value variance and covariance matrix, respectively. Let us consider $u:\boldsymbol{R}^d \rightarrow \boldsymbol{R}$ to be the approximation of h, then to find the approximation of the original target function f at a point $\boldsymbol{x} \in \boldsymbol{R}^n$ we calculate $u(\pi(\boldsymbol{x}))$. If $\boldsymbol{x}$ is on or around the manifold M this will give an approximate value of the function f applied to $\boldsymbol{x}$. If $\boldsymbol{x}$ is far from M the application the result of this calculation may not be really meaningful. This is consistent with our statement above that we are interested only in the approximation of f on and around M and not beyond a sufficiently wide neighborhood of M.

Finding the projection mapping π that satisfies the requirements stated above is not trivial. For example, SOMs may be used, which have the property of topology preserving mapping [5,7,22], however often SOM-s are used to find a quantization of the original data space, which reduces their discriminative ability. One option is to use over-complete SOMs [5,20], but this has the potential problem of excessive structural complexity, which may make it undesirable if reduced structural complexity is a critical requirement for the approximation solution.

LLE is an attractive method which can provide a topology preserving projection that also has good discrimination ability, while keeping the complexity of the projection relatively low. The key idea of the LLE is that for each data point a constrained linear combination of the nearest neighbor data points is found and then these linear combination weights are used to find the projections of the data points into the lower dimension by constraining the mapping of the data points. Here we provide a brief formal description of the LLE following [23].

For each data point $\boldsymbol{x}^k$ we define the r member nearest neighborhood among the data points as $B_k=\{\boldsymbol{x}^{j(k,i)}|i=1,r\text{: if } ||\boldsymbol{x}^t-\boldsymbol{x}^k|| \leq ||\boldsymbol{x}^{j(k,i)}-\boldsymbol{x}^k||\ \forall i=1,r, \text{ then } \exists a:1\leq a\leq r\text{: } k=j(k,a)\}$. The linear combination weights for $\boldsymbol{x}^k$ and its nearest r-neighborhood are calculated such that the following expression is minimized:

$$E(\boldsymbol{x}^k)=||\ \boldsymbol{x}^k - \Sigma_{i=1,r}\omega_{k,i}\,\boldsymbol{x}^{j(k,i)}|| \tag{2}$$

with the constraint that

$$\Sigma_{i=1,r}\omega_{k,i}=1 \tag{3}$$

This gives the best constrained approximation of the data point $\boldsymbol{x}^k$ using a linear combination of its r nearest neighbors.

Following algebraic calculations [23] the values of $\omega_{k,i}$ are found using the following expressions

$$c_{a,b}=<\boldsymbol{x}^k - \boldsymbol{x}^{j(k,a)},\ \boldsymbol{x}^k - \boldsymbol{x}^{j(k,b)}> \tag{4}$$

$$\omega_{k,i}=(\Sigma_{a=1,r}\ c_{i,a}{}^{-1})/\ (\Sigma_{b=1,r}\ \Sigma_{a=1,r}\ c_{b,a}{}^{-1}) \tag{5}$$

Having the $\omega_{k,i}$ weights for the constrained r-neighborhood linear combinations for each $\boldsymbol{x}^k$, we can calculate the projections of the data points into the low-dimensional space such that we preserve the topological structure of the data manifold. This is achieved by minimizing the following expression

$$E_{full}=\Sigma_{k=1,m}\ ||\ \boldsymbol{y}^k - \Sigma_{i=1,r}\omega_{k,i}\,\boldsymbol{y}^{j(k,i)}|| \tag{6}$$

where $\boldsymbol{y}^k \in \boldsymbol{R}^d$ are the low-dimensional projections of $\boldsymbol{x}^k$, $\omega_{k,i}$ are kept fixed and $\boldsymbol{y}^k$ are optimized.

The vectors $\boldsymbol{y}^k$ are constrained such that

$$\Sigma_{k=1,m}\,\boldsymbol{y}^k = 0 \tag{7}$$

and

$$(1/m)\cdot\Sigma_{k=1,m}\,\boldsymbol{y}^k\,\boldsymbol{y}^{kT} = I^d \tag{8}$$

where I^d is the $d\times d$ identity matrix.

Following algebraic calculations [23] it can be shown that the vectors $\boldsymbol{y}^k$ can be found by determining the eigenvectors of the $m\times m$ matrix Θ, which is defined as

$$\Theta=(I^m-\Psi)^T(I^m-\Psi) \tag{9}$$

where the matrix Ψ has the elements $\psi_{k,j}=\omega_{k,i}$ if $j=j(k,i)$ for some i, $1\leq i\leq r$ and zero otherwise, and I^m is the $m\times m$ identity matrix. The $\boldsymbol{y}^k$ vectors are found by projecting the original data vectors onto the eigenvectors of Θ that correspond to its smallest d non-zero eigenvalues.

To define the full projection π corresponding to the LLE projection calculated using the given data points, we need to extend it to points on the manifold that were not included among the sample of data points. To avoid further optimization calculations we rely on the topological organization preservation nature of the LLE and for any point $\boldsymbol{x} \in M$ we define the $\pi(\boldsymbol{x})$ as follows:

(i) determine the r-neighborhood of $\boldsymbol{x}$ among the given data points $\boldsymbol{x}^k$, be this $B(\boldsymbol{x})=\{\boldsymbol{x}^{j(x,i)},\ i=1,r\}$;

(ii) determine the linear weights $\omega(\boldsymbol{x})_i$ using equations (4) and (5);

(iii) calculate $\pi(\boldsymbol{x})=\ \Sigma_{i=1,r}\,\omega(\boldsymbol{x})_i\cdot\boldsymbol{y}^{j(x,i)}$, where $\boldsymbol{y}^{j(x,i)}$ are the projections of $\boldsymbol{x}^{j(x,i)} \in B(\boldsymbol{x})$.

Having an appropriate projection of the data manifold onto a matching low-dimensional space, we can now build neural networks that approximate the function $h:\boldsymbol{R}^d \rightarrow \boldsymbol{R}$. The approximating function will take the form

$$u(\boldsymbol{y})=\Sigma_{i=1,p}\,w'_i\,\varphi'(\boldsymbol{y};\boldsymbol{\theta}'_i) \tag{10}$$

where $\varphi'(\boldsymbol{y};\boldsymbol{\theta}'_i)$ are the neural network activation functions defined over $\boldsymbol{R}^d$ that match the functions $\varphi(\boldsymbol{x};\boldsymbol{\theta}_i)$ in equation (1).

It has been shown in [18], including the calculation of the error bounds, that manifold projections equivalent to the above described extended LLE projection allow good approximation of the target function by building the approximating function as a neural network using the low-dimensional projections of the data points. In [5] it is argued and demonstrated that the error bounds for the low-dimensional approximation are better than for the high-dimensional approximation, in-line with the intuitive expectation.

Having the same size sample data in the low-dimensional space as in the high-dimensional space, and the same size neural network using lower dimensional inputs instead of high-dimensional inputs, it is expected that the data sample provides a finer grained sample of the data space and the required size of the 'receptive fields' of the neurons is smaller in order to cover the relevant part of the projected data space. These factors are expected to improve the approximation performance of the neural network built using the projected data compared to the approximation performance of the neural network built using the original high-dimensional data.

To summarize the above proposed approach for the approximation of functions defined on high-dimensional spaces, first we use the LLE to build the projection π of the data manifold onto a matching low-dimensional space, next we build neural networks to approximate the target function over the projection space resulting the approximating function u, and finally we use this neural network approximation and the extended LLE projection of the data manifold to calculate the approximation of the original target function f at any point $\boldsymbol{x}$ in the high dimensional data space as $u(\pi(\boldsymbol{x}))$.

IV. Results and Discussion

To evaluate the proposed function approximation approach for functions defined on high-dimensional data we generated synthetic data sets and compared the performance of neural networks trained on high-dimensional data with the approximation performance of the proposed combined projection-approximation approach.

In all cases we used neural networks with 20 hidden units with Gaussian activation functions. The high-dimensional data was 60-dimensional and it was residing on a multiply curled swiss roll like manifold. The corresponding low-dimensional data was 5-dimensional. The relationship between the 5-dimensional $\boldsymbol{y}$ vectors and 60-dimensional $\boldsymbol{x}$ vectors is given by the following equations:

$$\boldsymbol{x}_{3\cdot(j-1)\cdot(10-j)+6\cdot(k-j-1)+1}=\mu\cdot\boldsymbol{y}_j\cdot cos(\boldsymbol{y}_j) \tag{11}$$

$$\boldsymbol{x}_{3\cdot(j-1)\cdot(10-j)+6\cdot(k-j-1)+2}=\mu\cdot\boldsymbol{y}_k$$

$$\boldsymbol{x}_{3\cdot(j-1)\cdot(10-j)+6\cdot(k-j-1)+3}=\mu\cdot\boldsymbol{y}_j\cdot sin(\boldsymbol{y}_j)$$

$$\boldsymbol{x}_{3\cdot(j-1)\cdot(10-j)+6\cdot(k-j-1)+4}=\mu\cdot\boldsymbol{y}_k\cdot cos(\boldsymbol{y}_k)$$

$$\boldsymbol{x}_{3\cdot(j-1)\cdot(10-j)+6\cdot(k-j-1)+5}=\mu\cdot\boldsymbol{y}_j$$

$$\boldsymbol{x}_{3\cdot(j-1)\cdot(10-j)+6\cdot(k-j-1)+6}=\mu\cdot\boldsymbol{y}_k\cdot sin(\boldsymbol{y}_k)$$

where $j<k$, $j=1,5$, and $k=j+1,5$, and

$$\mu=(\sqrt{5}\cdot||\boldsymbol{y}||)^{-1}(2\cdot(1+exp(-||\boldsymbol{y}||^2)^{-1}-1) \tag{12}$$

In other words, for each combination of two components $\boldsymbol{y}_j$, $\boldsymbol{y}_k$ of the 5-dimensional vector, with $j<k$, we define six components of the $\boldsymbol{x}$ vector using the Swiss roll equations (11). There are ten such combinations of two components $\boldsymbol{y}_j$, $\boldsymbol{y}_k$ of the 5-dimensional vector, with $j<k$, so in total we get a 60-dimensional $\boldsymbol{x}$ vector. The indices of the components of the vector $\boldsymbol{x}$ corresponding to the pair of components $\boldsymbol{y}_j$, $\boldsymbol{y}_k$ of the 5-dimensional vector are $3\cdot(j-1)\cdot(10-j)+6\cdot(k-j-1)+t$, where t goes from 1 to 6.

Ten functions were considered for the purpose of functions approximation. The functions are 5-dimensional generalizations of the 2-dimensional functions used in [5] and are given below.

1) Squared modulus:

$$f_1(\boldsymbol{x}(\boldsymbol{y}))=||\boldsymbol{y}||^2 \tag{13}$$

2) Second degree polynomial:

$$f_2(\boldsymbol{x}(\boldsymbol{y}))=(1/500)\cdot\Sigma_{j=1,4}\boldsymbol{y}_j^2\cdot\boldsymbol{y}_{j+1} \tag{14}$$

3) Exponential square sum:

$$f_3(\boldsymbol{x}(\boldsymbol{y}))=(1/500)\cdot\Sigma_{j=1,5}\,exp(\boldsymbol{y}_j^2/50) \tag{15}$$

4) Exponential-sinusoid sum:

$$f_4(\boldsymbol{x}(\boldsymbol{y}))=(1/500)\cdot(\Sigma_{j=1,4}\,exp(\boldsymbol{y}_j^2/50)\cdot sin(\boldsymbol{y}_{j+1}) + exp(\boldsymbol{y}_5^2/50)\cdot sin(\boldsymbol{y}_1)) \tag{16}$$

5) Polynomial-sinusoid sum:

$$f_5(\boldsymbol{x}(\boldsymbol{y}))=(1/50000)\cdot\Sigma_{j=1,5}\boldsymbol{y}_j^2\cdot cos(j\cdot\boldsymbol{y}_j) \tag{17}$$

6) Inverse exponential square sum:

$$f_6(\boldsymbol{x}(\boldsymbol{y}))=10\cdot(\Sigma_{j=1,5}\,exp(\boldsymbol{y}_j^2/25))^{-1} \tag{18}$$

7) Sigmoidal:

$$f_7(\boldsymbol{x}(\boldsymbol{y}))=10\cdot(1+exp(-\Sigma_{j=1,5}\boldsymbol{y}_j/5))^{-1} \tag{19}$$

TABLE I. PERFORMANCE INDICATORS OF NEURAL NETWORKS TRAINED WITH HIGH-DIMENSIONAL DATA

Performance measure: mean squared errors over 20 data sets			
Function	***Average***	***Standard deviation***	***t-test p-value***
Squared modulus (f_1)	17,467.4	21,026.1	0.0457
Second degree polynomial (f_2)	107.25	135.877	0.0051
Exponential square sum (f_3)	0.006605	0.012023	0.0252
Exponential-sinusoid sum (f_4)	0.006205	0.009021	0.0071
Polynomial-sinusoid sum (f_5)	0.005636	0.007459	0.0032
Inverse exponential square sum (f_6)	0.670829	0.804987	0.0057
Sigmoidal (f_7)	254.9053	245.8366	0.0004
Gaussian (f_8)	9.819289	10.10039	0.0064
Linear (f_9)	43,189	77,408.15	0.0297
Constant (f_{10})	0.435177	0.36814	4.21E-5

TABLE II. PERFORMANCE INDICATORS OF NEURAL NETWORKS TRAINED WITH PROJECTED LOW-DIMENSIONAL DATA

Performance measure: mean squared errors over 20 data sets			
Function	***Average***	***Standard deviation***	***t-test p-value***
Squared modulus (f_1)	7,226.65	4,747.98	0.0457
Second degree polynomial (f_2)	11.01783	5.270189	0.0051
Exponential square sum (f_3)	7.58E-5	4.37E-5	0.0252
Exponential-sinusoid sum (f_4)	0.000118	5.98E-5	0.0071
Polynomial-sinusoid sum (f_5)	3.6E-6	1.47E-6	0.0032
Inverse exponential square sum (f_6)	0.109654	0.01565	0.0057
Sigmoidal (f_7)	18.00114	5.913137	0.0004
Gaussian (f_8)	2.893639	0.592092	0.0064
Linear (f_9)	2,505.23	946.9379	0.0297
Constant (f_{10})	2.76E-5	9.46E-5	4.21E-5

8) Gaussian:

$$f_8(\mathbf{x(y)}) = 10 \cdot exp(-\Sigma_{j=1,5} \mathbf{y}_j^2/100) \tag{20}$$

9) Linear:

$$f_9(\mathbf{x(y)}) = \Sigma_{j=1,5}\, j \cdot \mathbf{y}_j \tag{21}$$

10) Constant:

$$f_{10}(\mathbf{x(y)}) = 1 \tag{22}$$

For each approximated function we considered 20 different data sets. The same 20 data sets were used for each target function. Each data set consisted of 5000 randomly chosen training data points and 400 randomly chosen test data points, both the training and testing sets being selected using uniform sampling of $[-10,10]^5$.

For each data set we used the LLE projection calculation to determine the mapping of the 60-dimensional data points into a 5-dimensinal space, i.e. we assumed that we know the dimensionality of the data manifold. The test data points were projected using the above described extension of the LLE projection. For the calculation of the eigenvectors we used the iterative QR matrix factorization method, which requires less memory for the calculations than other methods for the determination of eigenvectors.

For each dataset and for each target function we generated one trained neural network using the high-dimensional data and one trained neural network using the projected low-dimensional data. We calculated the mean squared error for all neural networks and for all data sets and all target functions. Then we calculated the average and standard deviation of the mean squared errors in order to compare the approximation performance of the neural networks for each target functions (i.e. average and standard deviation of performances over the 20 data sets). The results are presented in Tables I and II.

The results show that in all cases the neural network approximation of the target function that used the projected data performed statistically significantly better (at the significance level p=0.05, i.e. all p-values are below 0.05) than the neural network approximations of the target functions using the original high-dimensional data. In seven cases the performance difference is statistically significant even at the level of p=0.01. This confirms our expectations stated earlier that the approximation of the target function using the combination of manifold projection and approximation in the low-dimensional space should work better than the direct approximation of the function in the high dimensional space in terms of generalization performance.

The results presented here are important because they confirm that the combination of manifold projection and low-dimensional approximation works in practice. The available theoretical results [5,18] indicate that the low-dimensional approximation following the manifold projection can be expected to be better than the direct approximation of functions defined over high-dimensional spaces, however these results do not quantify very precisely the scale of improvement. The experimental results reported here indicate that the improvement can be very significant and ask for further theoretical work to get improved theoretical estimates of the bounds of the approximation performance improvements that can be achieved through the combination of manifold projection and low-dimensional approximation of functions defined over high-dimensional spaces.

An important aspect of the manifold projection is that the target function over the projected space should retain its features that characterize it over the data manifold within the high-dimensional space, e.g. the behavior of derivatives, local and global maxima and minima. The topology preservation of the projection and the maintenance of sufficient discrimination ability provide some guarantees about the retaining of the fundamental nature of the target function over the projected space. However, the extent to which the specific features (e.g. derivative behavior) are retained remains unclear. The projection of the manifold onto the low-dimensional space may alter the nature of the reconstituted approximation ($u(\pi(\mathbf{x}))$) of the target function ($f(\mathbf{x})$) and this will be reflected by the transformed target function as well ($h(\pi(\mathbf{x}))$). For example, if SOM-s are used for the projection of the manifold, $u(\pi(\mathbf{x}))$ will approximate $f(\mathbf{x})$ as piece-wise locally constant function, obviously not preserving directly the derivative behavior of the target function. The extended LLE projection that we used in this paper preserves the topological structure of the manifold more smoothly and can be expected to preserve features of the target functions more closely as well. Our results imply that it is important to invest effort into the establishment of theoretical results about the preservation of features of target functions following the manifold projection onto the low-dimensional space in order to provide sound theoretical foundations for such practical applications.

The proposed method involves large volumes of numerical calculations, in particular in relation with the calculation of the eigenvectors of large matrices – the size of the matrices is given by the number of data points. The matrices involved are sparse, which makes the volume of calculations somewhat smaller, but still the choice of the right method for matrix operations is important to avoid running into memory management problems. If the number of data points is very large, which is required if the dimensionality of the original data points is high, careful consideration of matrix operations becomes even more important in order to keep the method numerically feasible.

It can be expected that the proposed method will find useful applications in the context of the use of big data generated by increasingly used and increasingly detailed and sophisticated sensor networks, especially in industrial settings. These large volumes of sensor data derived from complex processes are likely to have many non-trivial relationships between components of the data. Discovering these in itself is a challenge, but having them discovered at least to some extent leads also the challenge of making practically useful the knowledge of these relationships. The combined approximation method proposed here that relies on the low-dimensional projection of data manifolds can make such knowledge really useful when function approximation is used for system identification and control.

V. Conclusions

In this paper a novel approach is presented for the approximation of functions defined over high-dimensional spaces. Assuming that the high-dimensional data points reside on a low-dimensional manifold embedded into the high-dimensional data space, first we use a projection of the data manifold into a matching low-dimensional space and approximate the function in this projection space using neural networks. For the projection of the data manifold we use the local linear embedding (LLE) method [6,23], which satisfies the requirements of maintenance of the topological neighborhood structure of the data manifold and of the preservation of sufficient discrimination ability between points on the data manifold. For calculation of projections of points on the data manifold not included in the determination of the LLE projection, we use a simple extension of the LLE projection based on the approximation of the point using a constrained linear combination of its nearest neighbors from the set of data points used for the determination of the LLE projection.

Intuitively the methodology described here should work well and have better approximation properties than the approximation of the target function using neural networks that work with data from the original high-dimensional data space. Recent theoretical results [5,18] provide the definite support for this intuitive expectation, however we noted throughout the paper that there are still important theoretical details that need investigation and the establishment of rigorous results.

The experimental results show convincingly that the approximation performance of neural networks improves significantly if we use the projected low-dimensional data instead of the original high-dimensional data to train neural networks to approximate a wide range of target functions. This is a very promising results and points to the potential use of the proposed combined function approximation method in the context of system identification and control based on large volumes of sensor data that is increasingly available both from home and industrial environments (e.g. sensors in intelligent home appliances and cars, sensors in complex engines and reactors).

References

[1] T. Hastie, R. Tibshirani, J. Friedman, The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Springer, 2009.

[2] I.M. Johnstone and D.M. Titterington, "Statistical challenges of high-dimensional data", Philosophical Transactions of The Royal Society A, vol.367, pp.4237-4253, 2009.

[3] J.H. Friedman, "An overview of predictive learning and function approximation", NATO ASI Series F Computer and System Science 136, 1994.

[4] J.H. Friedman, "On bias, variance, 0/1 – loss and the curse-of-dimensionality", Data Mining and Knowledge Discovery, vol.1, pp.55-77, 1997.

[5] P. Andras, "Function approximation using combined unsupervised and supervised learning", IEEE Transactions on Neural Networks and Learning Systems, vol.25, pp.495-505, 2014.

[6] S. Roweis, L. Saul, "Nonlinear dimensionality reduction by locally linear embedding.", Science, vol.290, pp.2323-2326, 2000.

[7] T. Kohonen, Self-Organizing Maps, Springer, 2001.

[8] S. Haykin, Neural Networks and Learning Machines, Prentice Hall, 2008.

[9] K. Hornik, "Multilayer feedforward networks are universal approximators", Neural Networks, vol.2, pp.183-192, 1989.

[10] M.B. Stinchcombe, "Neural networks approximation of continuous functional and continuous functions on compactifications", Neural Networks, vol.12, pp.467-477, 1999.

[11] V. Kurkova, "Kolmogorov's theorem and multilayer neural networks", neural Networks, vol.5, pp.501-506, 1992.

[12] K. Hornik, M. Stinchcombe, H. White, P. Auer, "Degree of approximation results for feedforward networks approximating unknown mappings and their derivatives", Neural Computation, vol.6, pp.1262-1275, 1994.

[13] A.R. Barron, “Approximation and estimation bounds for artificial neural networks”, Machine Learning, vol.14, pp.115-133, 1991.

[14] A.R. Barron, “Universal approximation bounds for superpositions of a sigmoidal function”, IEEE Transactions on Information Theory, vol.39, pp.930-945, 1993.

[15] G. Gnecco, “A comparison between fixed-basis and variable-basis schemes for function approximation and functional optimization”, Journal of Applied Mathematics, article ID 806945, 2012.

[16] G. Gnecco, M. Sanguineti, “On a variational norm tailored to variable-basis approximation schemes”, IEEE Transactions on Information Theory, vol.57, pp.549-558, 2011.

[17] G.-B. Huang, Q.-Y. Zhu, C.-K. Siew, “Extreme learning machine: theory and applications”, Neurocomputing, vol.70, pp.489-501.

[18] K. Yu, T. Zhang, Y. Gong, “Nonlinear learning using local coordinate coding”, Advances in Neural Information Processing Systems – NIPS 22, pp.2223-2231, 2009.

[19] F. Camastra, “Data dimensionality estimation methods: a survey”, Pattern Recognition, vol.36, pp.2945-2954, 2003.

[20] H. Yin, “Data visualization and manifold mapping using the ViSOM”, Neural Networks, vol.15, pp.1005-1016, 2002.

[21] J. De Leeuw, "Nonlinear principal component analysis and related techniques." Department of Statistics, UCLA, 2011.

[22] P. Andras, "Kernel-kohonen networks." International Journal of Neural Systems vol.12, pp.117-135, 2002.

[23] L.K. Saul, S.T. Roweis, An introduction to local linear embedding. Accessed at www.cs.nyu.edu/~roweis/lle/papers/lleintro.pdf on 3rd January 2015, 2001.