Function Approximation Using Combined Unsupervised and Supervised Learning

Peter Andras, *Senior Member, IEEE*

***Abstract*—Function approximation is one of the core tasks that are solved using neural networks in the context of many engineering problems. However, good approximation results need good sampling of the data space, which usually requires exponentially increasing volume of data as the dimensionality of the data increases. At the same time, often the high dimensional data is arranged around a much lower dimensional manifold. Here we propose the breaking of the function approximation task for high dimensional data into two steps: first the mapping of the high dimensional data onto a lower dimensional space corresponding to the manifold on which the data resides; second the approximation of the function using the mapped lower dimensional data. We use over-complete self-organizing maps for the mapping through unsupervised learning, and single hidden layer neural networks for the function approximation through supervised learning. We also extend the two step procedure by considering support vector machines and Bayesian self-organizing maps for the determination of the best parameters for the nonlinear neurons in the hidden layer of the neural networks used for the function approximation. We compare the approximation performance of the proposed neural networks using a set of functions and show that indeed the neural networks using combined unsupervised and supervised learning outperform in most cases the neural networks that learn the function approximation using the original high dimensional data.**

***Index Terms*—Function approximation, Learning, Neural network, Self-organizing map**

I. Introduction

FUNCTION approximation is a core task in many engineering, economic, and computational problems [1-4]. In general, many kinds of learning tasks (e.g. classification, pattern recognition, prediction) can be formulated as a function approximation task [1, 5]. There are many approaches to function approximation including relatively simple methods, e.g. least squares linear approximation, and many more complex methods, e.g. approximation with splines or neural networks [1, 6-8].

Function approximation with neural networks has strong theoretical foundations. It is well established that neural networks with a single hidden layer can be seen as linear combinations of nonlinear basis functions, and for a wide range of basis function classes (e.g. sigmoidal functions, Gaussian functions) it has been shown that linear combinations of such functions can approximate continuous functions arbitrarily correctly [1, 8-10]. The same result can be extended easily to functions that can be approximated by continuous functions (e.g. step functions, classification functions).

Considering the practical side, the theoretical results do not guarantee the finding of small size neural networks with arbitrary correctness. In some cases the number of neurons in the hidden layer has to be large to achieve good approximation of a given function [11-13]. However, often, a relatively small size neural network can work sufficiently well especially in the case of functions defined on a low dimensional input space (e.g. functions defined over the one or two dimensional real space or some subset of these) [1].

The practical problems with neural network approximation in many cases are caused by the sparseness of the data that is used to learn the approximation of the target function [14-17]. Especially if the input data is high dimensional (e.g. 5 or 10 or even more), a good sample of the data space has to be very large (e.g. millions, billions or more data points). If this is not the case, the generalization ability of the trained neural network (i.e. the ability to approximate sufficiently correctly the target function for input data that has not been seen) will remain poor. In addition to issues caused by the high dimensionality and sparseness of the data sample another common practical problem is due to the uneven distribution of the data sample in the data space [14]. If the available data samples densely a part of the data space but it is very sparse elsewhere, the function learned by the neural network will approximate well the target function only over the subset of the data space that is densely sampled, and its generalization ability over the part of the data space that is sparsely sampled will stay poor.

Often, high dimensional data is arranged around a lower dimensional manifold that is embedded in the high dimensional space [1, 14]. In principle, if the lower dimensional manifold is known the approximation of a function defined on a high dimensional space can be reduced to the approximation of a function on a much lower dimensional space and combination of this with the function that transforms this lower dimensional space into the manifold embedded into the higher dimensional space [18]. However, in

Manuscript received April 28, 2012. (Write the date on which you submitted your paper for review.)

P. Andras is with the School of Computing Science of Newcastle University, Newcastle upon Tyne, NE1 7RU, UK (corresponding author phone: +44-191-2227946; fax: +44-191-2228232; e-mail: peter.andras@ncl.ac.uk).

general the analytical form of the manifold on which the data resides is not known.

Here we propose to use a combination of unsupervised and supervised learning to improve function approximation performance in the case of functions defined on data that can be expected to reside on a low dimensional manifold embedded into a high dimensional space. To deal with the lack of information about the analytical form of the manifold, we use an over-complete self-organizing map (SOM) [19-24] to learn the topographic structure of the unknown manifold. Then we learn the approximation of a function defined over the node space of the SOM. The combined unsupervised and supervised learning allows to improve significantly the approximation of functions defined over high dimensional spaces. We demonstrate this through a selection of example applications. We note that similar approaches of combined application of supervised and unsupervised learning have been already proposed to address various data analysis problems, e.g. dimension reduction using combined support vector machines and independent component analysis [25], setting the basis function parameters for RBF neural networks [26], data mining applications [27], however to the best of our knowledge none of these addresses the issue that we formulated above.

II. Background

A. Unsupervised Learning with Self-Organizing Maps

There are many variants of learning the organization of the data in the data space. Manifold learning algorithms aim to learn the structure or the characteristics of the manifold around which the data resides [14, 28]. Often these characteristics describe the manifold in a local sense and the learning leads to the generation of a patchwork of local models of the manifold [29, 30]. Such learning algorithms are usually classified as unsupervised learning [1, 14] as there is no a priori information about what is the right local model or right set of characteristics of the manifold that is aimed to be learned.

SOMs are one of the alternatives to learn the distribution of the data in the data space [19-21]. The key idea of the SOM is that the data is projected from one space into another such that an appropriate topographic organization of the data is maintained. It is assumed that the projection space corresponds to the manifold around which the data is assumed to reside. The maintenance of the topographic organization of the data means that topological neighborhoods in the projected space defined by the distance metric of this space correspond to topological neighborhoods in the data space. Since the data is assumed to reside around a manifold that has a lower dimension than its embedding space, parts of the manifold may be close in the data space, but distant in the space of the manifold. Thus closeness in the data space does not necessarily imply closeness in the transform space. In accordance, a topological neighborhood in the data space is not necessarily projected into a topological neighborhood in the projected space.

The classic SOM [21] is defined by a set of nodes arranged in a grid according to their position vectors set in the projection space, each node having an associated prototype vector from the data space. When a data point is presented, the SOM nodes compete for the data and the winning node is the one which has its prototype vector closest to the data point. Then the data point is projected onto the position vector of the winning node. In the learning phase, the prototype vector of the winning node, and of the nodes within a local neighborhood of it in the projection space, get moved closer to the data point for which the winning node was chosen. Assuming that there are N nodes in the SOM, each node being defined as

$$node_j : (\theta_j, z_j) \quad \theta_j \in D, z_j \in P, j = 1,\ldots, N \tag{1}$$

where θ_j is the prototype vector and z_j is the position vector of node j, the learning rules are

$$i = \underset{j=1,\ldots,N}{\arg\min}\, d_D(x, \theta_j) \tag{2}$$

$$\begin{aligned} &\theta_j = (1-\gamma)\cdot\theta_j + \gamma \cdot x \\ &\mathit{for}\ j\ \mathit{such\ that}\ d_P(z_j, z_i) < \rho \end{aligned} \tag{3}$$

where x is the data point, γ is the learning rate, and ρ is the radius of the projection space neighborhood of the winning node. This radius is gradually decreased towards zero as the training progresses.

The learning by the SOM can be interpreted as learning of the distribution of the data in the data space. The data distribution is learned as a linear combination of normal distributions centered at the prototype vectors of the nodes of the SOM [19-21]

$$pr(x \mid \Theta) = \sum_{j=1}^{N} \Pr(\theta_j) \cdot pr(x \mid \theta_j) \tag{4}$$

where $\Pr(\theta_j)$ is the probability that the distribution centered around θ_j is the correct distribution of the data and $pr(x \mid \theta_j)$ is the basis distribution centered at θ_j. Assuming normal basis distributions they take the form of

$$pr(x \mid \theta_j) = \frac{1}{(2\pi)^{\frac{d}{2}} \mid C_j \mid^{\frac{1}{2}}} \cdot e^{-\frac{1}{2}\cdot(x-\theta_j)^T C_j^{-1}(x-\theta_j)} \tag{5}$$

where C_j is the covariance matrix of the distribution associated with node j.

The learning rules for the distribution learning, i.e. learning $C_j, \Pr(\theta_j)$ and γ can be derived by minimizing the Kullback-

Leibler distance between the approximating distribution (equation (4)) and the actual distribution estimated as a sum of Dirac δ distributions centered at the data points [19-21].

In certain cases SOMs are build such that they provide an over-complete representation of the projected data by having more nodes than projected data points [22-24]. In such cases the SOM nodes that do not attract data points from the training data act as interpolator approximations of points expected to be in the data space that were not included into the training data [24]. Such SOMs are often used for visualization of the data [22-24].

B. Supervised Learning of Function Approximation with Single Hidden Layer Neural Networks

It has been shown that neural networks with a single hidden layer containing nonlinear neurons can, at least in principle, approximate a very wide range of functional relationships arbitrarily correctly [1, 9, 10]. In general the formal proofs of this property of neural networks are based on showing that the set of linear combinations of functions of some generic form (basis functions) is dense within the set of continuous functions, i.e., for any continuous function f there is an infinite series of functions g_n within the chosen set of functions such that $\lim_{n\to\infty} d(f, g_n) = 0$ for an appropriate metric d on the space of continuous functions [9, 10] . The result also extends to non-continuous functions (e.g. step function) that can be approximated by continuous functions.

This implies that a neural network with neurons in the hidden layer implementing such basis functions and a linear summation output neuron, in principle can approximate any continuous and many discontinuous functions defined on the input space of the hidden neurons. The approximation theory results typically imply a relatively high minimal number of neurons for the worst case scenario approximation [31-35]. Some of these results show that the number of required neurons in many cases grows exponentially or polinomially with the dimensionality of the data [18, 32, 35-37]. However if the parameters of the basis functions represented by the neurons are allowed to vary, for certain classes of approximated functions and appropriate basis functions, the required number of neurons may not be required to grow so quickly or at all with the dimensionality of the data [36-38].

Usually the number and internal parameters of the basis functions of the neurons are fixed and the aim is to learn the linear summation weights of the output neuron. This is done by optimizing these weights through supervised learning and considering the squared error as the objective that is optimized. In principle it is also possible to learn the internal parameters of the basis functions by optimizing the objective with respect to these parameters as well, however, this may complicate very much the learning process in computational terms (see for example [39]).

Often heuristics are used to set the number and internal parameters of hidden neurons on the basis of the size of the data set, the dimensionality of the input data, prior knowledge about the nature of the data, and performance sensitivity to removal of neurons [2-4, 33, 40]. In practice usually a small number of hidden nonlinear neurons give a good approximation of most target functions following parameter learning [1]. More principled approaches suggest to use some form of model selection method based on some model complexity criterion [14].

A principled and generic solution to the problem of the setting the number of hidden neurons and their internal parameters is provided by the support vector machine approach [41]. This approach aims to find the minimal number of support vectors, i.e. data points, which are necessary to approximate the target function with a given level of precision [14, 41, 42]. To include nonlinear basis functions, the support vector machines assume a transformation of the original data into another space (usually a function space) where the corresponding function that has to be approximated is a linear function. Normally this other space is infinite dimensional, however the dealing with infinite dimensional data is avoided through the use of the ‘kernel trick’, which allows the internal product of vectors in the transformed space to be calculated using a nonlinear kernel function that is defined over the original data space. In effect the calculated approximation of the target function is a linear combination of nonlinear kernel functions having one of their argument fixed at one of the support vector data points.

In general, key problems of nonlinear function approximation with neural networks remain that the reliable approximation of functions defined on high dimensional inputs often requires very large data volumes that grow exponentially with the dimensionality of the data [18, 36, 37, 43], and finding simple models of the data with good generalization ability, in the form of neurons networks is very difficult. Often the volume of the available data is very small if the dimensionality of the data is taken into account, i.e. the coverage of the data space is too sparse. On the other side, if the available data amount is sufficiently large, it may be quite difficult to find a neural network with small number of hidden neurons that has good generalized approximation ability.

III. Combined Unsupervised and Supervised Learning of Function Approximation

Let us assume that the task is to learn the approximation of a function defined on multi-dimensional data using a neural network with a fixed set of basis functions, and that the data resides around a lower dimensional manifold embedded into the original data space. The reason for the data not being exactly on this manifold is the possibility of measurement error. Given that the data resides around a low dimensional manifold it is possible that a data sample that is sufficiently dense on the supporting manifold is quite sparse in the context of the embedding original data space. Thus the direct approximation of the function defined on the original data space may suffer from the apparent sparseness of the data.

To avoid the problem of apparent data sparseness, first we may map the data onto a space corresponding to the supporting manifold of the data and then learn the approximation of a function defined on this mapping space. The original function that has to be approximated can be then recovered by composing the mapping of the data onto the lower dimensional mapping space and the learned approximation of the function defined on this mapping space.

It has been shown [18] for a wide range of functions defined on m-dimensional that if the function value depends only on $m' < m$ dimensions of the data then the approximation error of a neural network with fixed basis functions that approximates such functions is proportional to m/m'. Thus if the approximation is performed following the mapping of the data onto the m'-dimensional space, the expected error is reduced. At the same time the constants involved in the estimates of the approximation error for neural networks often depend exponentially or polinomially on the dimensionality of the data [18, 32, 35-37]. Thus by using lower dimensional data following the mapping of the original data into the lower dimensional space, it can be expected that the error of the approximation is reduced.

In formal terms, the original task is to learn the approximation of the function defined by

$$f : D \to \mathbf{R}, f(x_t) = y_t, x_t \in D \tag{6}$$

where D is the original high dimensional data space ($D \subseteq \mathbf{R}^m$), x_t are the sample data points and y_t are the function values for these data points. We assume that $x_t \in M \subset D$ and that there exists a bijective mapping

$$\mu : M \to M', M' \subseteq \mathbf{R}^{m'}, m' < m \,. \tag{7}$$

Then, if we know the mapping $\mu : M \to M'$ we can learn the function f by learning the function g defined as

$$g : M' \to \mathbf{R}, g(z_t) = y_t, z_t = \mu(x_t) \in M' \tag{8}$$

and composing the learned function with the mapping μ, i.e.

$$f(x) = g(\mu(x)), x \in M \tag{9}$$

and $f(x)$ for $x \notin M$ being defined by using a continuous extension of the definition of $f(x)$ over M such that this converges relatively quickly to zero in all directions. In fact, strictly speaking, for points $x \notin M$ the function is not defined, but in order to handle the noise in the measurement of input data it makes practical sense to use a continuous extension of the function defined by equation (9). The speed of convergence to zero of the continuous extension part of the function is defined by the expected range of measurement errors of the data points, e.g. if the errors follow a distribution

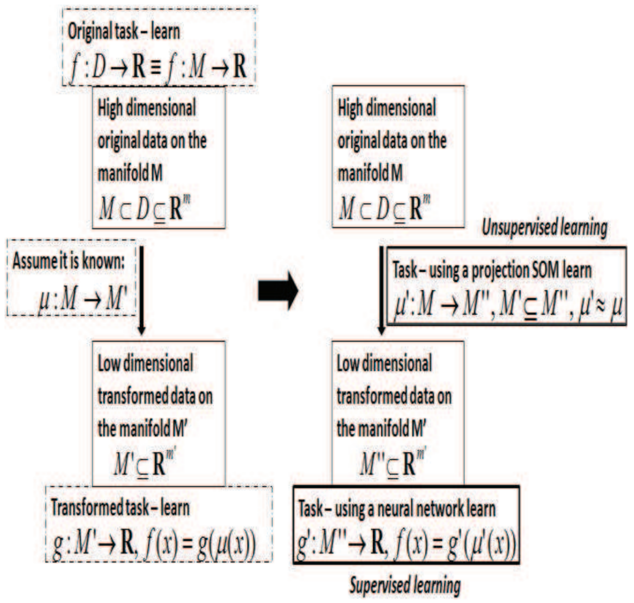
Original task – learn
$f : D \to R \equiv f : M \to R$
High dimensional original data on the manifold M
$M \subset D \subseteq R^m$
Assume it is known:
$\mu : M \to M'$
Low dimensional transformed data on the manifold M'
$M' \subseteq R^{m'}$
Transformed task – learn
$g : M' \to R, f(x) = g(\mu(x))$
High dimensional original data on the manifold M
$M \subset D \subseteq R^m$
Unsupervised learning
Task – using a projection SOM learn
$\mu' : M \to M'', M' \subseteq M'', \mu' \approx \mu$
Low dimensional transformed data on the manifold M'
$M'' \subseteq R^{m'}$
Task – using a neural network learn
$g' : M'' \to R, f(x) = g'(\mu'(x))$
Supervised learning

Figure 1. The proposed combination of unsupervised and supervised learning for function approximation. On the left the segmented line boxes show the original learning task, the key assumption and the transformed learning task. On the right the solid thick line boxes show the proposed actual learning tasks.

with zero mean and standard deviation equal to σ, then the continuous extension part of the function should converge quickly to zero for points beyond the +/-2σ size neighborhood of M in all directions.

In accordance with [18], in the case of approximation with a fixed set of basis functions, and for certain classes of approximated functions, the neural network approximation error for $g(\mu(x))$ is bounded from below by

$$c_{m'} = \frac{C}{16\pi e^{\pi-1} m'} \cdot \left(\frac{1}{2n}\right)^{1/m'} \tag{10}$$

where n is the number of neurons used in the neural network, and C is a constant related to the class of functions to be approximated. The lower bound for the approximation error for neural networks with a fixed set of basis functions approximating $f(x)$ directly in the data space is

$$c_m = \frac{C}{16\pi e^{\pi-1} m} \cdot \left(\frac{1}{2n}\right)^{1/m} \tag{11}$$

where n is the number of neurons, and C is a constant as above. As we noted the constant C often depends exponentially on the dimensionality of the data [32, 36, 37]. Assuming that $C = b^m$, we have that $b^m \cdot (1/m) \cdot (1/2n)^{1/m}$ is increasing if $b > e^{1/(4\ln(2n))}$, implying that we have that $c_{m'} < c_m$ if b satisfies the latter condition, which is very likely, given that $e^{1/(4\ln(2n))} < 1.1$ already for moderate n. This means that the neural network approximation error for $g(\mu(x))$ is likely to be lower than the neural network approximation error for $f(x)$, considering in both cases small size neural networks with fixed sets of basis functions.

In general it is very hard to find the bijective mapping $\mu : M \to M'$ since we do not know the analytical form of M. However, the effective dimensionality of M, and thus the dimensionality of M', i.e. m', can be determined by analyzing the data points x_t (e.g. by using principal component analysis, multi-dimensional scaling, correlation dimension determination, or box-counting Hausdorff dimension determination [44]) . Having this information in principle we can search for an approximation of a bijective mapping of M onto $\mathbf{R}^{m'}$. Note that the mapping $\mu' : M \to \mathbf{R}^{m'}$, which approximates $\mu : M \to M'$ is not necessarily a bijective (and not even an injective) mapping. A such approximate mapping of M onto $\mathbf{R}^{m'}$ can be constructed using an over-complete SOM for which the position vectors are in $\mathbf{R}^{m'}$ and the prototype vectors are in M. Being an over-complete SOM means that the number of the nodes of the SOM is much larger than the number of data points that are considered [22-24].

Using an over-complete SOM implies that in principle it is possible that each data point maps onto a SOM node, such that no other data point maps onto the same SOM node. Thus, at least in principle, using an over-complete SOM makes possible to have an injective approximate mapping $\mu' : M \to \mathbf{R}^{m'}$. However in practice it may be possible that more than one data points map onto a single SOM node in the case of some (most likely few) SOM nodes.

Let us assume that the projection SOM is such that each input data maps onto a unique SOM node. Given that there are many more SOM nodes than input vectors, we can assume that between any two SOM nodes that attract an input vector there are other SOM nods that do not attract input vectors. Thus we can assume that for a given input x there is neighborhood radius ρ_x such that there is only one other input x' such that the node i' that attracts x' is within the ρ_x radius neighborhood of the node i that attracts x. We further assume that the neighborhood radius is decreasing in each step by a factor β (i.e. $\rho_{next} = \beta \cdot \rho_{current}$). We can also assume without loss of generality that the node i' is not in the neighborhood radius for any other nodes attracting an input vector. According to our assumptions the further training of the projection SOM from the perspective of the nodes attracting x and x' is equivalent with the alternate training with only these two inputs. After $2k$ turns of training the prototype vectors of the two nodes are

$$\begin{aligned}&\theta_i^{2k} = (1-\gamma)^{2k} \cdot \theta_i^0 + \gamma \cdot (1 + (1-\gamma)^2 + \ldots \\ &+ (1-\gamma)^{2k}) \cdot x + \gamma \cdot (1-\gamma) \cdot (1 + (1-\gamma)^2 + \ldots \\ &+ (1-\gamma)^{2k-2}) \cdot x'\end{aligned} \tag{14}$$

$$\begin{aligned}&\theta_{i'}^{2k} = (1-\gamma)^{2k} \cdot \theta_{i'}^0 + \gamma \cdot (1 + (1-\gamma)^2 + \ldots \\ &+ (1-\gamma)^{2k}) \cdot x + \gamma \cdot (1-\gamma) \cdot (1 + (1-\gamma)^2 + \ldots \\ &+ (1-\gamma)^{2k-2}) \cdot x'\end{aligned} \tag{15}$$

where θ_i^0 and $\theta_{i'}^0$ are the prototype vectors of the two nodes at the time point when our assumptions become valid. Thus the two prototype vectors become increasingly similar with the training and at one moment it is possible that another node becomes the attractor of one of the input vectors, let say x'.

Let us assume that the node i'' is outside of the neighborhood radius for the node i after $2k_0$ turns of training, but stays within the neighborhood radius of i'. The prototype vector for this node after $2k$ turns of training will be

$$\begin{aligned}&\theta_{i''}^{2k} = (1-\gamma)^{2k} \cdot \theta_{i''}^0 + \gamma \cdot (1 + (1-\gamma)^2 + \ldots \\ &+ (1-\gamma)^{2k_0}) \cdot x + \gamma \cdot (1-\gamma) \cdot (1 + (1-\gamma)^2 + \ldots \\ &+ (1-\gamma)^{2k_0-2}) \cdot x' + \gamma \cdot (1 + (1-\gamma) + \ldots \\ &+ (1-\gamma)^{2k-2k_0}) \cdot x'.\end{aligned} \tag{16}$$

This node takes over the attraction of the data vector x' from the node i', if

$$| x' - \theta_{i''}^{2k} | < | x' - \theta_{i'}^{2k} | . \tag{17}$$

Given that $1-\gamma < 1$ we can ignore the $(1-\gamma)^{2k} \cdot \theta^0$ components for large k, and following algebraic manipulations we find that the above inequality is equivalent to

$$\begin{aligned}&\left| \frac{1-(1-\gamma)^{2k_0+2}}{2-\gamma} \cdot x + \right. \\ &\left. \frac{1-\gamma}{2-\gamma} \cdot \left(1-(1-\gamma)^{2k_0} - (2-\gamma)\cdot(1-\gamma)^{2k-2k_0}\right) \cdot x' \right| < \\ &\left| \frac{1-(1-\gamma)^{2k+2}}{2-\gamma} \cdot x + \frac{1-\gamma}{2-\gamma} \cdot \left(\gamma - (1-\gamma)^{2k}\right) \cdot x' \right| .\end{aligned} \tag{18}$$

Solving this inequality for k_0 and k gives

$$k_0 = r(\gamma) \cdot \ln(| x - x' |) . \tag{19}$$

This implies that the distance between the position vectors of nodes i and i'' is proportional to $| x - x' |$, i.e.

$$\rho_0 = \beta^{k_0} \cdot \rho_x = \alpha \cdot | x - x' | \cdot \rho_x . \tag{20}$$

This shows that the distances in the projection space between the nodes to which data vectors project are proportional to the distances of these data vectors in the data space. Note that ρ_x depends in a similar manner on distances between data points projected in the neighborhood of node i. This means that the proportional distance preservation in the projection space may change in terms of proportionality multipliers between separately mapped topological neighborhoods, but it will be valid within these neighborhoods. Thus the over-complete projection SOM in the optimal case realizes a mapping that locally preserves proportional distances of the data vectors. The projection SOM increasingly preserves the proportional distances between the data points as the number of nodes is increased in the projection SOM.

The use of the SOM guarantees that the topographic structure of M is preserved through the mapping $\mu': M \to \mathbf{R}^{m'}$, and in fact the mapping will map M onto a finite $M'' \subseteq \mathbf{R}^{m'}$ that is an m'-dimensional brick. In general we can assume that $M' \subseteq M''$. Since the topological organization of M does not necessarily match that of a brick, it is likely that the mapping will use only some of the nodes of the over-complete SOM, and some (possibly many) nodes of the SOM will not attract any data points. At the same time, due to the fact that all prototype vectors of the SOM nodes converge towards the input vectors used to train the SOM, the prototype vectors of unused SOM nodes will represent points of the data manifold that were not included into the data sample. Thus the mapping onto the over-complete SOM can be expected to generalize in a faithful manner to points in the data space that were not used for the training of the SOM – i.e. the topology of the data manifold through the mapping $\mu': M \to M''$ will be maintained for previously unseen points from the data space (see equation (1)). Effectively the SOM mapping of the data will approximate M' with a finite m'-dimensional brick lattice.

Having the mapping $\mu': M \to M''$ the function approximation learning task is reduced to the learning of $g: M' \to \mathbf{R}$ expanded to M'', i.e. $g': M'' \to \mathbf{R}$. Given our assumptions about the arrangement of the data the sampling of M'' is much denser than the sampling of D and thus the approximation of $g': M'' \to \mathbf{R}$ is likely to be more precise than the direct approximation of $f: D \to \mathbf{R}$.

To approximate $g': M'' \to \mathbf{R}$ we use a single hidden layer neural network with a sufficiently high number of nonlinear hidden layer neurons with fixed internal parameters. For example, we may use an RBF neural network with fixed Gaussian basis functions as activation functions of the neurons in the hidden layer.

By increasing the number of SOM nodes we get a finer brick lattice representation of M'' by the SOM. More SOM nodes increase the number of SOM nodes which have a single data vector associated with them. This implies the increasing preservation of proportional distances of data points within the mapped topological neighborhoods. We assume that more proportional distance preservation within mapped neighborhoods means more faithful projection of the data manifold into the projection space. Improving the faithfulness of the projection of the data manifold onto the projection space means that a neural network approximation of $g': M'' \to \mathbf{R}$ will get closer to the theoretically possible best approximation of the target function in the m'-dimensional space.

The approximation of $g': M'' \to \mathbf{R}$ is learned using the training data $\mu'(x_t), y_t$. As the injectivity of $\mu': M \to M''$ is not fully guaranteed, it is possible that $\mu'(x_t) = \mu'(x_u), t \neq u$, i.e. two (and possibly more) different data points project to the same SOM node. Thus there will be at least two potentially different y values (i.e. y_t, y_u) associated with this SOM node and its position vector $z_t = z_u$. In effect the neural network will most likely learn the mean value of the y values associated with such SOM nodes and their position vector. Different x and y values may be the result of noisy measurement of the actual data. In such cases the noisy measurements of the data vectors may map onto the same SOM node with position vector z, thus effectively filtering the impact of the noise on the measurement of data vectors.

In summary, it is proposed to learn the approximation of functions defined on high dimensional data spaces by first projecting the data using an over-complete SOM onto a lower dimensional projection space and then learning the approximation of a function defined on this lower dimensional space. The advantage of this approach is that sparse high dimensional data is projected into a low dimensional space where the data set is much less sparse due to the lower dimensionality of the space. The approximation of the function is likely to be more precise through the proposed combined neural network approach than the direct neural network approximation of the function defined over the original data space (in both cases the neural networks use a fixed set of basis functions). The underlying key assumption of the proposed approach is that the data lies around a low dimensional manifold that is embedded into the high dimensional data space. If this key assumption is not satisfied, it is likely that the proposed combined neural network approach to function approximation will not lead to improved results compared to the direct neural network approximation of the function over the original data space.

IV. Support Vector Machine Extension

The combined unsupervised and supervised learning of function approximation proposed in the previous section means that the direct approximation of the function in the high dimensional original data space is replaced by a mapping of the data onto a low dimensional space and the learning of the approximation in this low dimensional space. The use of the over-complete SOM implies that the difference in the complexity and difficulty of the approximation of the target function between the high and the low dimensional approximation is traded for the maintenance of the over-complete SOM that implements the mapping from the high to the low dimensional space.

Since we use an over-complete SOM in general a SOM node is expected to attract a single data point, so we do not expect a compression of the data by this mapping. Thus, the number of separate data points is retained following the mapping onto the lower dimensional space, leaving the complexity of the selection of the size of the approximating neural network unchanged.

The use of the support vector machine approach provides a principled way to choose the number of hidden nodes and the values of the basis function parameters for the approximating neural network. The support vector machine has the form of

$$G(x) = \sum_{t=1}^{N} (\alpha_t - \alpha_t^*) \cdot K(\mu(x), \mu(x_t)) \tag{21}$$

where the calculation of $\mu(x)$ is performed using the SOM mapping of an arbitrary data point x onto the over-complete SOM and $\alpha_t - \alpha_t^* = 0$ if $\mu(x_t)$ is not a support vector.

We note however, that the support vector machine approach may overestimate the minimum number of hidden neurons needed for sufficiently good approximation of the target function. To overcome this problem we may use additional sensitivity analysis to prune the neural network further.

V. Bayesian SOM Approach

An alternative approach to set the internal parameters for the basis functions while keeping the number of neurons low is to use a Bayesian re-mapping of the SOM. This finds an approximation of the distribution of the low dimensional mapped data as a linear combination of a small set of normal distributions. The SOM nodes that are the centers of these normal distributions are then used to define the internal parameters of hidden neurons of the neural network. The Bayesian SOM is set up with a number of nodes that is much smaller than the number of nodes of the projection SOM. If $\mu' : M \to M''$ is the projection SOM mapping, and the nodes of the projection SOM are defined as

$$(z_k, \overline{x_k}),\ k = 1,\ldots,K,\ z_k \in M'',\ \overline{x_k} \in M \tag{22}$$

then the Bayesian SOM is defined as

$$(q_l, \overline{z_l}),\ l = 1,\ldots,L,\ q_l \in M'',\ \overline{z_l} \in M'',\ L << K \tag{23}$$

such that the position vectors of the nodes of the projection SOM (z_k) are mapped onto the Bayesian SOM nodes. The Bayesian SOM nodes also have the attached parameters C_l and $\Pr(q_l)$ that are the covariance matrix and prior probability associated with the node $(q_l, \overline{z_l})$.

The parameters of the Bayesian SOM ($C_l, \Pr(q_l), \overline{z_l}$) are calculated using the equations (2), (3) and the Bayesian SOM learning rules [19-21]. The resulting Bayesian SOM will provide an approximation of the probability density function of the distribution of the position vectors of the projection SOM onto which the original data points are projected. The learned position vectors $\overline{z_l}$ will be used then as the fixed parameters for the hidden neurons of the neural network built for function approximation.

Consequently, the function represented by the neural network will be

$$G(x) = \sum_{l=1}^{L} w_l \cdot g_l(\mu'(x); \overline{z_l}) \tag{24}$$

having randomly chosen initial weights. These weights are then modified through neural network learning.

The advantage of the Bayesian SOM approach is that it provides a small set of fixed internal parameters for the hidden neurons of the approximating neural network, calculated in a principled optimal manner, i.e. they define a good approximation of the probability density function of the distribution of the projections of the data points.

The number of nodes in the Bayesian SOM (L) is not determined in any principled manner, i.e. any number L that seems reasonable may be picked. To deal with the arbitrariness of the picking of L, we may consider a series of L values and pick the one that is optimal in the sense that L is sufficiently small and at the same time the approximation error of the function $G(x)$ is sufficiently small as well. The Bayesian SOM approach combined with the search for the optimal L value deals with the problem of finding the minimal complexity for a sufficiently good approximation of the target function by starting from the low end, i.e. by considering first approximations with potentially too low structural complexity.

VI. Application Examples and Statistical Performance Comparison

We evaluate the proposed combined unsupervised and supervised learning methods for learning the approximation of functions by considering a set of target functions and a selection of neural networks. The original data in all cases is in a 6 dimensional space and it is always situated on a 2 dimensional manifold. The relationship between the 6 dimensional data points $x = (x_1, \ldots, x_6)$ and their corresponding 2 dimensional position $\psi = (\psi_1, \psi_2)$ defines a double Swiss roll in 6 dimensions, and is given by the following equations for $\psi \neq (0,0)$:

$$\begin{aligned} x_1 &= \lambda \cdot \psi_1 \cdot \cos(\psi_1) \\ x_2 &= \lambda \cdot \psi_2 \\ x_3 &= \lambda \cdot \psi_1 \cdot \sin(\psi_1) \\ x_4 &= \lambda \cdot \psi_2 \cdot \cos(\psi_2) \\ x_5 &= \lambda \cdot \psi_1 \\ x_6 &= \lambda \cdot \psi_2 \cdot \sin(\psi_2) \\ \lambda &= \frac{10}{\sqrt{5} \cdot \sqrt{\psi_1^2 + \psi_2^2}} \cdot \left(\frac{2}{1 + e^{-\psi_1^2 - \psi_2^2}} - 1 \right) \end{aligned} \tag{25}$$

for $\psi = (0,0)$ the corresponding 6 dimensional position is $x = (0,...,0)$.

For the learning phase we consider 1000 randomly selected training data points and for testing phase we use 200 randomly selected test data points. The 2 dimensional points, $\psi = (\psi_1, \psi_2)$, were selected from the $[-10,10] \times [-10,10]$ square. The error measure in all cases is the mean squared error (see equation (13)).

The unsupervised neural network in all cases is an over-complete projection SOM with 10,000 nodes that projects the original data points into a 2 dimensional space. For learning the projection SOM we use 1000 epochs of training with all training data. For learning the SOM mapping we use equations (2) and (3) with $\gamma = 0.1$, $\rho_1 = 0.1 \cdot \max_{i,j} || x^i - x^j ||$, and $\rho_{r+1} = 0.99 \cdot \rho_r$.

The neural networks with fixed basis functions that we consider are as follows:

1) radial basis function (RBF) neural network with 20 hidden neurons with randomly set fixed internal parameters, trained on the original data – we name this neural network: RBF;
2) radial basis function (RBF) neural network with 20 hidden neurons with randomly set fixed internal parameters, trained on the data mapped using the projection SOM – we name this neural network: SOM-RBF;
3) radial basis function (RBF) neural network with basis function parameters set using the support vector machine approach and pruned to have the 20 most important neurons resulting from the support vector machine solution, considering the data mapped using the projection SOM – we name this neural network: SVM-RBF;
4) radial basis function (RBF) neural network with basis function parameters set using the Bayesian SOM approach with 20 nodes in the Bayesian SOM, considering the data mapped using the projection SOM, the Bayesian SOM is trained with the same SOM parameters (γ, ρ) that we use for the training of the projection SOM and the initial prior probabilities ($P(q_l)$) are set to be equal – we name this neural network: BSOM-RBF.

The training of all neural networks involves the change of the weights on the outputs of the nonlinear neurons, but no internal parameters of the neurons are modified through learning, i.e. we use fixed basis functions in all cases. Note that for the SVM-RBF and BSOM-RBF neural networks the location of the centre and the width of the Gaussian are necessarily fixed due to method of setting up of these networks.

We consider 10 functions for the purpose of testing the approximation performance of these neural networks. These functions are as follows:

1) Squared modulus: $\psi = (\psi_1, \psi_2)$
$$f(z) = \psi_1^2 + \psi_2^2 \tag{26}$$
2) Polynomial: $\psi = (\psi_1, \psi_2)$
$$f(z) = \frac{1}{500} \cdot \left(4\psi_1^4 + 3\psi_1^3\psi_2 + 2\psi_1\psi_2^2 \right) \tag{27}$$
3) Exponential square sum: $\psi = (\psi_1, \psi_2)$
$$f(z) = e^{\psi_1^2/50} + e^{\psi_2^2/50} \tag{28}$$
4) Exponential-sinusoid sum: $\psi = (\psi_1, \psi_2)$
$$f(z) = e^{\psi_1^2/50} \cdot \sin(\psi_2) + e^{\psi_2^2/50} \cdot \cos(\psi_1) \tag{29}$$
5) Polynomial-sinusoid sum: $\psi = (\psi_1, \psi_2)$
$$f(z) = \frac{1}{100} \cdot \left(\psi_1^2 \cdot \sin(\psi_1) + \psi_2^3 \cdot \cos(2\psi_2) \right) \tag{30}$$
6) Inverse exponential square sum: $\psi = (\psi_1, \psi_2)$
$$f(z) = \frac{10}{e^{\psi_1^2/25} + e^{\psi_2^2/25}} \tag{31}$$
7) Sigmoidal: $\psi = (\psi_1, \psi_2)$
$$f(z) = \frac{10}{1 + e^{-(\psi_1 + \psi_2)/5}} \tag{32}$$
8) Gaussian: $\psi = (\psi_1, \psi_2)$
$$f(z) = 10 \cdot e^{-(\psi_1^2 + \psi_2^2)/100} \tag{33}$$
9) Linear: $\psi = (\psi_1, \psi_2)$
$$f(z) = \psi_1 + 2\psi_2 \tag{34}$$
10) Constant: $\psi = (\psi_1, \psi_2)$

$$f(\psi) = 1 \cdot \tag{35}$$

The functions are set such that their values are in similar ranges over the domain from which the inputs are selected ($\psi \in [-10,10] \times [-10,10]$). Each function approximation learning task was executed 20 times following random initialization of the parameters.

To evaluate the performance of the considered function approximation approaches, first we calculated the approximation performance of a uniformly zero approximation (i.e. the approximation of the function value is zero everywhere) as a benchmark. The uniformly zero approximation corresponds to the default neural network with all weights being set to zero. Then we calculated the empirical mean and standard deviation of the difference between the approximation error of the uniform zero approximation and the neural network approximation for each kind of neural networks. Thus the test set specific performance metric is defined as follows:

$$\eta_{NN}(x_t; t = 1,\ldots,200) = E_0(x_t; t = 1,\ldots,200) - E_{NN}(x_t; t = 1,\ldots,200) \tag{36}$$

where

$$E_{NN}(x_t; t = 1,\ldots,200) = \frac{1}{200} \cdot \sum_{t=1}^{200} (y_t - G_{NN}(x_t))^2 \tag{37}$$

$$E_0(x_t; t = 1,\ldots,200) = \frac{1}{200} \cdot \sum_{t=1}^{200} (y_t - 0)^2 \cdot \tag{38}$$

The empirical mean and empirical standard deviation of the test set specific performance metric are

$$\overline{\eta_{NN}} = \frac{1}{20} \cdot \sum_{k=1}^{20} \eta_{NN}(x_t^k; t = 1,\ldots,200, k = 1,\ldots 20) \tag{39}$$

$$\eta_{NN}^{\sigma} = \sqrt{\frac{1}{20} \cdot \sum_{k=1}^{20} (\eta_{NN}(x_t^k; t = 1,\ldots,200) - \overline{\eta_{NN}})^2} \cdot \tag{40}$$

We used the z-test to test whether the mean differences were significantly different from zero, thus the overall direct performance metric is defined as:

$$\eta_{NN}^{z} = \sqrt{20} \cdot \frac{\overline{\eta_{NN}}}{\eta_{NN}^{\sigma}} \cdot \tag{41}$$

We calculated as final performance metric the z-test p-value corresponding to η_{NN}^{z} for each neural network type that we considered.

A positive mean difference that is statistically significant at the level of a p-value $p<0.05$ indicates that the respective neural network approximation is better than the uniform zero approximation for the target function. The empirical mean values, empirical standard deviations and the results of the z-test significance level calculations are presented in Table I. The approximation performance of a neural network is better if the reported empirical mean value for the network is more positive and more statistically significant.

TABLE I
THE DIFFERENCE BETWEEN THE APPROXIMATION PERFORMANCE OF NEURAL NETWORKS AND THAT OF THE UNIFORM ZERO APPROXIMATION
MEAN VALUE (STANDARD DEVIATION) [Z-TEST P-VALUE], * INDICATES SIGNIFICANCE, I.E. BELOW 0.05 P-VALUE

Function	RBF	SOM-RBF	SVM-RBF	BSOM-RBF
Squared modulus	1519.84 (1404.62) [6.52E-7*]	3000.73 (547.31) [0*]	3872.61 (573.84) [0*]	3353.81 (964.04) [0*]
Polynomial	– 183.55 (260.47) [0.00081*]	– 49.553 (106.468) [0.01869*]	105.33 (73.82) [8.81E-11*]	177.69 (63.57) [0*]
Exponential square sum	14.734 (3.175) [0*]	18.821 (1.519) [0*]	20.517 (1.408) [0*]	19.215 (2.856) [0*]
Exponential-sinusoid sum	– 0.1443 (1.0173) [0.26281]	– 0.0764 (0.4752) [0.23600]	– 0.04511 (0.3336) [0.27269]	0.0166 (0.1626) [0.32364]
Polynomial-sinusoid sum	– 0.3219 (1.6303) [0.18856]	0.2777 (0.9503) [0.09558]	0.3294 (0.6562) [0.01237*]	0.1925 (0.3519) [0.00721*]
Inverse exponential square sum	– 0.7642 (1.1273) [0.00121*]	0.3318 (0.8327) [0.03739*]	1.3571 (0.7420) [1.11E-16*]	2.0276 (0.5878) [0*]
Sigmoidal	25.026 (5.562) [0*]	29.543 (2.231) [0*]	29.082 (2.346) [0*]	18.621 (6.755) [0*]
Gaussian	23.384 (2.998) [0*]	26.016 (3.023) [0*]	28.594 (2.499) [0*]	24.649 (5.712) [0*]
Linear	65.483 (38.979) [2.8E-14*]	88.973 (16.334) [0*]	83.392 (20.621) [0*]	25.056 (22.418) [2.89E-7*]
Constant	0.9827 (0.0185) [0*]	0.9976 (0.0031) [0*]	0.9987 (0.0006) [0*]	0.9258 (0.0878) [0*]

The results show that the RBF neural networks approximate the target functions better than the uniform zero approximation with the exception of polynomial, exponential sinusoid sum, polynomial sinusoid sum and inverse exponential square sum functions. The SOM-RBF neural networks are better than the uniform zero approximation for all considered functions with exception of the polynomial, exponential sinusoid sum, and polynomial sinusoid sum functions. The SVM-RBF and BSOM-RBF neural networks are better than the uniform zero approximation for all considered functions with the exception of the exponential sinusoid sum function. In the case of the exponential sinusoid sum function none of the neural network approximations works statistically significantly differently from the uniform zero approximation. This confirms that in

TABLE II
THE DIFFERENCE BETWEEN THE APPROXIMATION PERFORMANCE OF COMBINED LEARNING NEURAL NETWORKS AND THAT OF THE RBF NEURAL NETWORKS; MEAN VALUE (STANDARD DEVIATION) [Z-TEST P-VALUE], * INDICATES SIGNIFICANCE, BELOW 0.05 P-VALUE

Function	SOM-RBF	SVM-RBF	BSOM-RBF
Squared modulus	1480.89 (1343.14) [4.09E-7*]	2352.77 (1576.94) [1.26E-11*]	1833.97 (1780.76) [2.05E-6*]
Polynomial	134.00 (316.78) [0.02926*]	288.88 (263.47) [4.71E-7*]	361.24 (280.55) [4.24E-9*]
Exponential square sum	4.0868 (3.2636) [1.07E-8*]	5.7829 (3.3816) [1.02E-14*]	4.4811 (4.1574) [7.17E-7*]
Exponential-sinusoid sum	0.0679 (1.1606) [0.39672]	0.0992 (1.0290) [0.33307]	0.1610 (0.9831) [0.23194]
Polynomial-sinusoid sum	0.5997 (1.4523) [0.03238*]	0.6514 (1.4549) [0.02261*]	0.5145 (1.6339) [0.07952]
Inverse exponential square sum	1.0960 (1.2442) [4.08E-5*]	2.1214 (1.0439) [0*]	2.7919 (0.9415) [0*]
Sigmoidal	4.5167 (5.1484) [4.36E-5*]	4.0564 (5.0697) [0.00017*]	– 6.4053 (7.5654) [7.64E-5*]
Gaussian	2.6314 (1.7863) [2.23E-11*]	5.2090 (1.5073) [0*]	1.2641 (5.8834) [0.16829]
Linear	23.490 (37.151) [0.00234*]	17.909 (40.089) [0.02286*]	– 40.426 (45.517) [3.56E-5*]
Constant	0.0149 (0.0187) [0.00018*]	0.0160 (0.0185) [5.37E-5*]	– 0.0568 (0.0927) [0.00307*]

almost all considered cases of function approximation tasks the neural networks that we used can learn the approximation of the target function.

Next we compared performance of the RBF neural networks with the performance of the neural networks based on combined unsupervised and supervised learning. For this, we calculated the differences of approximation errors and used the z-test to check whether the mean differences are significantly different from zero or not. A positive difference that is statistically significant indicates that the RBF neural networks are less good at approximating the target function than the neural networks that use combined learning. The values of empirical mean differences, the corresponding empirical standard deviations, and the corresponding calculated significance levels for the z-test are shown in Table II.

The results show that the SOM-RBF and SVM-RBF neural networks approximate the target function significantly better than the RBF neural networks in the case of all functions for which the neural network approximations are better than the uniform zero approximation. The results also show that the RBF neural networks approximate the sigmoidal, linear and constant functions statistically significantly better than the BSOM-RBF neural networks. These results confirm that the neural networks using the combined unsupervised and supervised learning outperform in most cases the neural networks that learn the function approximation through supervised learning applied directly in the original data space.

Finally we compared the performance of the RBF-SOM neural networks with the performance of the RBF-SVM and RBF-BSOM neural networks in a similar manner as in the previous comparison. Positive values that are statistically significant indicate that the SVM-RBF and BSOM-RBF neural networks are better than the SOM-RBF neural networks for the respective approximation task. The comparison results are shown in Table III.

These results show that the SVM-RBF neural networks are significantly better than the SOM-RBF neural networks for the approximation of the considered squared modulus, polynomial, exponential square sum, inverse exponential square sum and Gaussian functions. The SOM-RBF neural networks are significantly better than the SVM-RBF neural networks for the considered sigmoidal and linear functions, while for the other functions the approximation performances are not significantly different for the two kinds of neural networks. The BSOM-RBF neural networks are significantly better than the SOM-RBF neural networks for the considered polynomial and inverse exponential square sum functions. For the sigmoidal, linear and constant functions the reverse performance relationship is statistically significant, while for the remaining functions the approximation performances are not significantly different. These results show that in particular for simpler target functions trying to set the internal parameters of the nonlinear neurons in some optimal way may be somewhat misleading and comparable or better performance can be achieved by simply random setting of these parameters.

VII. DISCUSSION AND CONCLUSIONS

The paper proposes the use of neural networks trained with combined unsupervised and supervised learning for function approximation tasks. The key idea is that sparse data in a high dimensional space may be mapped into a lower dimensional space that corresponds to the lower dimensional manifold around which the data resides. This can improve the sampling density of the data space and lead to a trained neural network with inputs from the lower dimensional space such that the approximation performance of this neural network is better than the performance of a similar neural network trained on the original high dimensional data. To perform the high dimension to low dimension mapping we propose the use of over-complete self-organizing maps that can approximate an injective mapping. The experimental data presented in the paper confirms that in all considered cases the combined learning neural networks (SOM-RBF networks) have better approximation performance than the neural networks trained with the original high dimensional data (RBF networks).

We also propose the use of support vector machines (SVM-RBF networks) and Bayesian self-organizing maps (BSOM-RBF networks) to find good settings for the fixed parameters of the nonlinear neurons in the neural networks. The experimental results show that the use of such methods to find good parameters for the nonlinear neurons is useful in some cases, but not always. The results indicate that the SVM-RBF networks are more likely to perform better than the SOM-RBF networks than the BSOM-RBF networks. In the case of the SVM-RBF neural networks the determination of the parameters for the neural network basis functions is influenced by the function that is approximated and the distribution of the data. In the case of the BSOM-RBF networks the setting of the parameters is driven purely by the distribution of the projections of the original data into the low dimensional space. This difference in the drivers of the setting of the parameters is likely to be the reason for the difference in the performance of these networks relative to the performance of the SOM-RBF neural networks.

The theoretical arguments for the proposed combined learning neural networks show that it is very important to find a good $\mu' : M \to M''$ SOM mapping in the sense that the dimensionality of the low dimensional space is close to the true dimensionality of the manifold around which the original data is placed and that appropriate parts of the manifold are mapped onto appropriate parts of the low dimensional space in terms of topographic arrangement. If this is not the case, the $g : M'' \to \mathbf{R}$ function that is approximated over the low dimensional space might become more variable over sufficiently large regions of the projection space than the original target function of the approximation task, $f : D \to \mathbf{R}$. Thus the approximation of $g : M'' \to \mathbf{R}$ potentially may become even more complicated and more error prone than the approximation of $f : D \to \mathbf{R}$. Our experimental results show that for the data that we considered the SOM mapping into the lower dimensional space was appropriate. This of course was made easier by the fact that we mapped the high dimensional data into two dimensions which was the correct dimensionality of the manifold of the high dimensional data that we considered.

The experimental results show that the nature of the approximated function is important for achieving improved approximation by neural networks trained on the projected low dimensional data. The considered exponential sinusoidal sum function was equally badly approximated by all neural networks that we built. This indicates that if the variability of the target function is high (i.e. its values change considerably over relatively small regions of the space on which the function is defined) the approximation of the function is not significantly improved by mapping the original data onto a lower dimensional space. Of course, the approximation performance also depends on the complexity of the approximating neural network, and increasing the number of hidden neurons is likely to be more effective in improving the approximation performance using the low dimensional projected data than the original high dimensional data.

TABLE III
COMPARISON OF THE APPROXIMATION PERFORMANCE RESULTS OF SVM-RBF, BSOM-RBF, AND SOM-RBF NEURAL NETWORKS
MEAN VALUE (STANDARD DEVIATION) [Z-TEST P-VALUE], * INDICATES SIGNIFICANCE, BELOW 0.05 P-VALUE

Function	SVM-RBF vs SOM-RBF	BSOM-RBF vs SOM-RBF
Squared modulus	871.885 (631.894) [3.40E-10*]	353.083 (1139.46) [0.08290]
Polynomial	154.884 (127.669) [2.89E-8*]	227.247 (127.848) [8.88E-16*]
Exponential square sum	1.6960 (1.2974) [2.51E-9*]	0.3942 (2.9925) [0.27787]
Exponential-sinusoid sum	0.0313 (0.4045) [0.36546]	0.0930 (0.5027) [0.20387]
Polynomial-sinusoid sum	0.0517 (0.6327) [0.35739]	– 0.0852 (0.9050) [0.33683]
Inverse exponential square sum	1.0253 (0.6184) [6.08E-14]	1.6958 (0.6984) [0*]
Sigmoidal	– 0.4603 (0.6202) [0.00045*]	– 10.922 (7.5645) [5.33E-11]
Gaussian	2.5776 (1.3589) [0*]	– 1.3672 (6.2453) [0.16377]
Linear	– 5.5805 (12.933) [0.02682*]	– 63.916 (23.963) [0*]
Constant	0.0010 (0.0033) [0.07360]	– 0.0717 (0.0879) [0.00013*]

Naturally the proposed combined learning neural networks can work using other kinds of high dimension – low dimension mapping algorithms as well. For example, the mapping between the original and the projection data space could be achieved using multi-dimensional scaling (MDS) [14], principal component analysis (PCA) [1], isometric feature mapping (ISOMAP) [28], or locally linear embedding (LLE) [30]. Any of these methods could be used to generate the projected data that is used to learn the function approximation in the lower dimensional space. However, an advantage of our choice of using over-complete self-organizing maps is the computational simplicity and the guaranteed topography preservation of the mapping.

REFERENCES

[1] S. Haykin, Neural Networks and Learning Machines, Prentice Hall, 2008.
[2] H.-G. Han, Q.-L.Chen, J.-F.Qiao, "An efficient self-organizing RBF neural network for water quality prediction", Neural Networks, vol.24, pp.717-725, 2011.
[3] R.C.J. Minnett, A.T. Smith, W.C. Lennon, R. Hecht-Nielsen, "Neural network tomopgraphy: network replication from output surface geometry", Neural Networks, vol.24, pp.484-492, 2011.
[4] A.-M. Zhou, K.D. Kumar, Z.-G. Hou, X. Liu, "Finite-time altitude tracking control for spacecraft using terminal sliding mode and Chebyshev neural network", IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, vol.41, pp.950-963, 2011.
[5] A.Y. Chervonenkis, "Problems of machine learning", LNCS 6744, pp.21-23, 2011
[6] A. Kryzak and T. Linder, "Radial basis function networks and complexity regularization in function learning", IEEE Transactions on Neural Networks, vol.9, pp.247-256, 1998.
[7] G.-B. Huang, P. Saratchandran, N. Sundararajan, "A generalized growing and pruning RBF (GGAP-RBF) neural network for function approximation, IEEE Transactions on Neural Networks, vol. 16, pp.57-67, 2005.
[8] G.A. Anastassiou, "Multivariate sigmoidal neural network approximation", Neural Networks, vol.24, pp.378-386, 2011.
[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] D.A. Sprecher, "A numerical implementation of Kolmogorov's superpositions I", Neural Networks, vol.9, pp.765-772, 1997.
[13] D.A. Sprecher, "A numerical implementation of Kolmogorov's superpositions II", Neural Networks, vol.10, pp.447-458, 1998.
[14] T. Hastie, R. Tibshirani, J. Friedman, The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Springer, 2009.
[15] 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.
[16] J.H. Friedman, "An overview of predictive learning and function approximation", NATO ASI Series F Computer and System Science 136, 1994.
[17] 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.
[18] 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.
[19] H. Yin, "The self-organizing maps: background, theories, extensions andapplications", Studies in Computational Intelligence, vol. 115, pp.715-762, 2008.
[20] M.M. Van Hulle, "Self-organizing maps", In: G. Rozenberg, T. Baeck, J. Kok (eds.) Handbook of Natural Computing: Theory, Experiments, and Applications, Springer, pp.1-45, 2010.
[21] T. Kohonen, Self-Organizing Maps, Springer, 2001.
[22] H. Yin, "On multidimensional scaling and the embedding of the self-organising maps", Neural Networks, vol.21, pp.160-169, 2008.
[23] H. Yin, "Data visualization and manifold mapping using the ViSOM", Neural Networks, vol.15, pp.1005-1016, 2002.
[24] N. Manukyan, M.J. Eppstein, D.M. Rizzo, "Data-driven cluster reinforcement and visualisation in sparsely-matched self-organising maps", IEEE Transactions on Neural networks and Learning Systems, vol.23, pp.846-853, 2012.
[25] S. Moon, H. Qi, "Hybrid dimensionality reduction method based on support vector machine and independent component analysis", IEEE Transactions on Neural networks and Learning Systems, vol.23, pp.749-761, 2012.
[26] N.B. Karayiannis, G.W. Mi, "Growing radial basis neural networks: merging supervised and unsupervised learning with network growth techniques", IEEE Transactions on Neural Networks, vol.8, pp.1492-1506.
[27] Z. Yao, A.H. Holmborn, T. Eklund, B. Back, "Combining unsupervised and supervised data mining techniques for conducting customer portfolio analysis", LNAI 6171, pp.292-307, 2010.
[28] J. Tenenbaum, V. De Silva, J. Langford, "A global geometric framework for nonlinear dimensionality reduction", Science, vol.290, pp.2319-2323, 2000.
[29] T. Lin and H. Zha, "Riemannian manifold learning", IEEE Transactions on Pattern Analysis and Machine Intelligence, vol.30, pp.796-809, 2008.
[30] S. Roweis and L. Saul, "Nonlinear dimensionality reduction by locally linear embedding", Science, vol.290, pp.2323-2326, 2000.
[31] A.R. Barron, "Approximation and estimation bounds for artificial neural networks", Machine Learning, vol.14, pp.115-133, 1991.
[32] A.R. Barron, "Universal approximation bounds for superpositions of a sigmoidal function", IEEE Transactions on Information Theory, vol.39, pp.930-945, 1993.
[33] A.R. Barron, A. Cohen, W. Dahmen, R.A. DeVore, "Approximation and learning by greedy algorithms", The Annals of Statistics, vol.36, pp.64-94, 2008.
[34] 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.
[35] 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.
[36] G. Gnecco, V. Kurkova, M. Sanguineti, "Some comparisons of complexity in dictionary-based and linear computational models", Neural Networks, vol.24, pp.171-182, 2011.
[37] P.C. Kainen, V. Kurkova, M. Sanguineti, "Dependence of Computational Models on Input Dimension: Tractability of Approximation and Optimization Tasks", IEEE Transactions on Information Theory, vol.58, pp.1203-1214, 2012.
[38] G. Gnecco, V. Kurkova, M. Sanguineti, "Can dictionary-based computational models outperform the best linear ones?", Neural Networks, vol.24, pp.881-887, 2011.
[39] W. Yao, X. Chen, Y. Zhao, M. van Tooren, ""Concurrent subspace width optimization method for RBF neural network modelling", IEEE Transactions on Neural Networks, vol.23, pp.247-259, 2012.
[40] R. Reed, "Pruning algorithms – a survey", IEEE Transactions on Neural Networks, vol.4, pp.740-747, 1993.
[41] V. Vapnik, The nature of Statistical Learning Theory, Springer, 2010.
[42] B. Schölkopf, A.J. Smola, Learning with Kernels: Support Vector machines, Regularization, Optimization and Beyond, MIT Press, 2001.
[43] T. Gerstner, M. Griebel, "Dimension – adaptive tensor – product quadrature", Computing, vol.71, pp.65-87, 2003.
[44] F. Camastra, "Data dimensionality estimation methods: a survey", Pattern Recognition, vol.36, pp.2945-2954, 2003.

Peter Andras (M'95–SM'10) has a BSc in computer science (1995), an MSc in artificial intelligence (1996) and a PhD in mathematical analysis of neural networks (2000), all from the Babes-Bolyai University, Cluj, Romania.

He is a Reader (Associate Professor) in the School of Computing Science, Newcastle University, UK. He has published 2 books and over 100 papers. He works in the areas of complex systems, computational intelligence and computational neuroscience.

Dr. Andras is member of the International Neural Network Society, of the Society for Artificial Intelligence and Simulation of Behaviour, and fellow of the Society of Biology.