Random Projection Neural Network Approximation

Peter Andras
School of Computing and Mathematics
Keele University
Newcastle-under-Lyme, UK
p.andras@keele.ac.uk

***Abstract*— Neural networks are often used to approximate functions defined over high-dimensional data spaces (e.g. text data, genomic data, multi-sensor data). Such approximation tasks are usually difficult due to the curse of dimensionality and improved methods are needed to deal with them effectively and efficiently. Since the data generally resides on a lower dimensional manifold various methods have been proposed to project the data first into a lower dimension and then build the neural network approximation over this lower dimensional projection data space. Here we follow this approach and combine it with the idea of weak learning through the use of random projections of the data. We show that random projection of the data works well and the approximation errors are smaller than in the case of approximation of the functions in the original data space. We explore the random projections with the aim to optimize this approach.**

Keywords—function approximation, high-dimensional, neural network, random projection, weak learning

I. INTRODUCTION

Approximation of functions is a general task in the context of applications of neural networks to complex engineering or data analytics problems (e.g. analysis of multi-sensor data in complex engines [1], assessing the relevance of pieces of text for a certain topic [2], calculating the risk associated with complex investment portfolios [3], etc.). Often the data that the problem involves is very high dimensional (10s / 100s / 1000s of dimensions), which makes solving the problem difficult.

The 'curse of dimensionality' [4] applies often to these function approximation problems, which means that as the dimensionality of the data grows the solution of the problem gets harder. The reason is that as the dimensionality of the data space grows, the sample of the data gets more sparse within the data space, providing less information about how the function behaves over this space. One common way of dealing with this problem is to impose regularisation constraints on the solution (e.g. smoothness, limited variation between points with known function values) [5], however, this approach may limit considerably the correctness of the approximation (i.e. if the space of allowed approximations that conform with the imposed constraints is far from the actual real function). An alternative approach relies on the feature of the data that this usually lays on a manifold within the data space such that the effective dimensionality of this manifold is much lower than the dimensionality of the data space [6]. In such cases the function may be approximated over this lower dimensional manifold, reducing the problem of data sparsity.

Previously, there have been proposals that utilise the assumption about the existence of a low-dimensional data manifold within the high-dimensional data space by projecting the data first into a low-dimensional space and then training a neural network over this projection space instead of training it over the original data space. One approach uses topology-preserving self-organising maps [7] to project the data into a low dimensional space [6], while another uses local linear embedding (LLE) [8] to implement this projection [9].

Recently, there has been considerable interest in using 'weak learners' as an ensemble approach to develop relatively simple solutions to complex machine learning problems [10,11]. One such approach is to use to use a set of random linear projections of the data into a lower dimensional space to reduce the difficulty of the learning step and then combine the resulting 'weak learners' into a more robust solution [12].

Here we propose to combine the last two approaches in the context of approximation of functions defined over high-dimensional data. Assuming that the data relies on a low-dimensional manifold we use random linear projections into a low-dimensional space and then use a neural network to learn the approximation of the function over the projection space. Combining a set of such neural networks provides a robust combination of 'weak learners', which are based on random projections of the data space into the low dimensional space. We investigate options to choose optimally the projection vectors and matrices, however we find that none of the intuitive ideas about optimal choice really works. The results show that the combination of multiple neural networks based on different random projections indeed reduces the variance of the function approximation performance, although it does not improve the approximation performance itself.

The rest of the paper is structured as follows. First we review briefly the related other works. Next we describe in

detail the proposed combination of random projections with neural networks for the approximation of high-dimensional functions. Then we present a set of application examples. Finally the paper is closed by the conclusions section.

II. Related Works

The theoretical approximation properties of neural networks have been established over 20 years ago [13-15]. It is known that neural networks with a single hidden layer of neurons with certain nonlinear activation functions (e.g. sigmoidal, Gaussian) have the property of universal approximation of all continuous functions and all functions that can be approximated arbitrarily correctly by continuous functions [13-15].

Approximation error bounds have been also derived for a range of neural network approximations [16,17]. In general, the approximation bounds get tighter as the number of neurons in the single hidden layer grows, i.e. the approximation bound is proportional to the inverse of the square root of the number of neurons [16]. However, it has also been shown that in many cases the multiplicative constants in the approximation error expressions grow exponentially with the dimensionality of the data [17]. The latter reflects the 'curse of dimensionality', i.e. that as the sparseness of the data grows with the dimensionality, there is more room for error in the approximation of the function.

Given that often the high-dimensional data relies on a much lower dimensional manifold embedded into the high-dimensional space, the idea of learning function approximation over the lower dimensional manifold or some transformation of this has considered by various researchers [6,18]. One approach is to project the manifold into a low-dimensional space of appropriate dimension and then approximate the function over this low-dimensional projection space. Previously, self-organising maps (SOM) [7] and local linear embedding (LLE) [8] have been demonstrated to work well for the projection step [6,9]. The underlying idea of these approaches is that the projection preserves the topological organisation of the data over the manifold, which is expected to help the correct approximation of the target function. However, in order to achieve this topology preserving mapping, both of these approaches involve considerable computational effort to calculate the low dimensional projections. The error bounds for such combined projection neural network approximations have also been established and show that the projection based function approximations are better than the neural network function approximations over the original high-dimensional space [9,18].

The concept of 'weak learners' has been used in various ways over the last 30 years [19,20], generally meaning the combination of not-very-good learning machines (in our context variants of function approximation machines, such as neural networks) such that the combination achieves a much better learning performance than the 'weak' components. For example, random forests with shallow trees in the context of classification [21,22], various 'bagging' and 'ensemble' learning approaches [19,20], feeding the output of 'weaker' learning engines into a 'stronger' learning engine [10], and so on. In general, the assumption is that each 'weak learner' performs a relatively imprecise approximation of the target function, but combining of these imprecise approximations improves the overall approximation as approximation errors cancel out through the combination.

One approach to 'weak learning' in the context of high-dimensional data is to use random linear projections of the data into a low-dimensional space and then perform the learning task over the projection space [12]. A big advantage of this approach is that it does not require computationally expensive data processing to achieve the low-dimensional projection, since it relies on using randomly picked matrices to generate the random linear projections. It has been shown that in the context of linear classification of high-dimensional data the random linear projections work well [12]. Error bounds have been derived demonstrating that the performance of the random projection approach is better than the classification over the high-dimensional original data space [12].

III. Random Projection Neural Networks

We assume that the function approximation task is specified as a set of data points and associated function values, $\{x^i, y_i = f(x^i)\}$, $i = 1,...,n$, $x^i \in \boldsymbol{R}^d$, $y_i \in \boldsymbol{R}$. It is also assumed that the data points reside on a manifold M embedded into the d-dimensional data space $\boldsymbol{R}^d$, such that the intrinsic dimensionality of M is $d' < d$. The approximation of the target function f is performed using neural networks with a single hidden layer, having neurons with a nonlinear activation function (e.g. Gaussian or sigmoidal).

The core idea of projection based neural network function approximation is that if we know d' the dimensionality of M then we can project the data points, for which the target function f has given values, onto a d' dimensional space and perform the approximation of the target function over this space [6]. The expectation is that, given the reduced dimensionality of the space over which we approximate the function, the approximation will be more precise. The reason for this expectation is that the approximation error bounds for approximation over a low-dimensional space are tighter than for neural network approximation over a high-dimensional space [17]. Earlier papers confirm that this expectation is valid and indeed the neural network function approximations over the projection space are significantly better than the neural network approximations over the original data space [6,9,23].

The intrinsic dimensionality of M can be determined with a variety of dimensionality estimation methods. For example we can use the box counting method [24]. For this, we count the number of boxes or balls of decreasing size, required for full coverage of the known part of the manifold (given by the points x^i, $i = 1,...,n$). Then calculate the exponent in the exponential relationship between the box side or ball radius and the number of required boxes or balls. The integer part of this exponent gives the estimated intrinsic dimensionality of the manifold.

Previously used projection based neural network approximations of high-dimensional functions assumed that the preservation of the topological structure of the data space through the projection into the lower dimensional space is

required for the good approximation performance [6,9,23]. To achieve the preservation of the topological structure of the data manifold these methods rely on computationally expensive projections (e.g. self-organising maps or local linear embedding) [7,8].

However, the idea of 'weak learners' [10,12] suggests that similarly good approximation performance may be achieved by combining a set of neural network approximations over the projection space, where each of these approximations use a much simpler projection, which is computationally cheap. Such simple projections can be generated by picking randomly a projection matrix and performing a linear projection of the data space using the projection matrix. Formally this is written as follows.

Let a^{kl} be projection vectors with randomly set components, $a^{kl} \in \boldsymbol{R}^d$, $k = 1,...,p$, $l = 1,...,d'$, where p is the number of random projection matrices that we use. Then a projection matrix is a $d' \times d$ matrix:

$$A^k = (a^{k1}, \ldots, a^{kd'})^T \quad (1)$$

The corresponding projection of data manifold is given by

$$z^{i,k} = A^k \cdot x^i \quad (2)$$

where $i = 1,...,n$ and $z^{i,k} \in \boldsymbol{R}^{d'}$. The neural networks then learn the approximation of the functions defined as

$$y_i = f^{*,k}(z^{i,k}) = f(x^i) \quad (3)$$

Let us denote by $g^{*,k}$ the function implemented by the neural network that learned to approximate the function $f^{*,k}$. Then the combined neural network approximation is given as

$$y^*_i = (1/p) \cdot \sum_{k=1,\ldots,p} g^{*,k}(z^{i,k}) \quad (4)$$

Let us denote by g the function implemented by the neural network that learned to approximate the function f over the original data space, and let us define

$$y^+_i = g(x^i) \quad (5)$$

In all these cases above the neural network approximation of the target functions is a linear combination of nonlinear basis functions, which act as the activation functions of the neurons in the hidden layer of the neural network. The learning process happens by adjusting the summation weights of the nonlinear basis functions using some form of gradient descent (or related) learning algorithm. We note that in principle other parameters of the activation functions of neurons can be adjusted as well, however, to keep the learning process simple, we assume here the basic learning process involving only the adjustment of summation weights.

Following the learning over the projection space and the original data space we expect that the approximation error of the high-dimensional neural network approximation will be larger than the approximation error of the averaged low-dimensional neural network approximation. Formally this is stated as:

$$(1/n) \cdot \sum_{i=1,\ldots,n} (y^+_i - y_i)^2 > (1/n) \cdot \sum_{i=1,\ldots,n} (y^*_i - y_i)^2 \quad (6)$$

Generally, the approximation error over the projection space will be proportional to $r^{d'}$, while the approximation over the original data space will be proportional to r^d, where $r > 1$ is an appropriate constant that depends on the activation functions used in the neurons of the hidden layer of the neural networks [17]. Assuming that we use the same kind of activation functions in the hidden layer neurons and that the number of neurons in the hidden layer is the same, this implies that the approximation error due to the projection distortion should be less than $\alpha \cdot (r^d - r^{d'})$ for some appropriate $\alpha > 0$ in order for equation (6) to be satisfied.

While none of the linear projections through the projection matrices A^k is likely to satisfy the topological mapping requirement that was used in previous works to show the tighter approximation for the low-dimensional case, it is reasonable to expect that each A^k will preserve the topological structure of M for some of its parts, $M^k \subset M$. Thus for any sufficiently small $M' \subset M$ we can set $K(M') \subset \{1,...,p\}$ such that we have that $M' \subset M^k$ for $k \in K(M')$ and $M' \not\subset M^k$ for $k \in \{1,...,p\} - K(M')$. If $p - |K(M')|$ is sufficiently large, i.e. if p is sufficiently large, it is reasonable to expect that the approximation errors due to randomly set projection matrices A^k, $k \in \{1,...,p\} - K(M')$, will cancel out together when summed up. For all projection matrices A^k, $k \in K(M')$ we can use the previous neural network approximation error results [9] that show that when the topological structure of the data manifold is preserved through the projection the above stated condition for the validity of the inequality in equation (6) is satisfied over M'.

Thus, if for M we can find a coverage $C(M) = \{M'_j \mid M'_j \subset M, j \in J\}$, where J is an index set, such that $M \subset \cup_{M' \in C(M)} M'$ and $M' \cap M'' = \emptyset$ if $M' \neq M''$, $M', M'' \in C(M)$ and for each $M' \in C(M)$ we have that $K(M') \neq \emptyset$, then the combined neural network approximation using the set of random linear projections will approximate the target function better than the neural network approximation trained using the original data in the high-dimensional space over the whole manifold M. This condition is likely to be satisfied if the number of random linear projections is sufficiently large.

We note that the proposed approximation of the target function is valid only on and around the manifold on which the data points reside. In fact the target function is expected to be defined only on the manifold. Consequently, extending the approximation well outside of the data manifold is meaningless and it should not be considered, even if purely technically the projection matrices work everywhere in the full original data space, including well outside of the data manifold.

We also note that while the above argument applies to the case of a sufficiently large set of random linear projection matrices, A^k, $k = 1,...,p$, in principle it is also possible that even

for a single random linear projection the neural network approximation over the projection space works better than the neural network approximation over the original high-dimensional data space. This can be the case if the largest $M' \subset M$ such that $K(M') = \{1\}$ (note that there is only one linear projection matrix in this case, i.e. $p = 1$), is sufficiently large, implying that the improved approximation applies for most of M and the potentially larger errors apply only to a small part of M. In this case the overall approximation performance of the neural network trained over the projection space is likely to be better than the approximation performance of the neural network trained over the high-dimensional original data space.

Considering the last note, it might be interesting to try to optimize the projection matrix such that the corresponding largest M' is maximal, while at the same time the computational cost of the optimization is kept low. In principle, the optimization objective is to keep the function over the projection space similar to the function over the original data space, which would be approximately achieved if the mapping would preserve the topological organisation of the data manifold. Given that the projection space is typically multi-dimensional itself, the direct measurement of the match between the original and projected function is computationally very expensive. Thus we could consider proxy optimization objective that apply to the component projection vectors of the projection matrices.

By considering the component projection vectors $a^{kl} \in \boldsymbol{R}^d$, $k = 1,..., p$, $l = 1,...,d'$, the optimisation criteria can be formulated in terms of the resulting function following projection with a chosen projection vector, $f_{kl}: \boldsymbol{R} \rightarrow \boldsymbol{R}$,

$$f_{kl}(u) = (1/|I(u,k,l,\delta)|) \cdot \sum_{i \in I(u,k,l,\delta)} f(x^i) \quad (7)$$

where $I(u,k,l,\delta)$ is the index set for the indices of all x^i such that

$$|a^{kl\,T} \cdot x^i - u| \leq \delta \quad (8)$$

for some appropriately chosen small $\delta > 0$. In effect, this means that the values of $f_{kl}(u)$ are defined as the average of the values of the target function over all x^i-s for which their projection by a^{kl} is close to u. While $\delta = 0$ would be the best choice in principle, in practice this may not work well and this is why we chose a $\delta > 0$.

Naturally the f_{kl} functions are not likely to be very close to the target function, however we may find features of these projection functions that make them better than other projection functions for the purpose of faithful approximation of f over the projection space.

One option is to consider the variance of the values of f_{kl}. In principle if this variance is small, it means that the projection function is close to constant, which may imply that the projection by the corresponding projection vector is not very discriminative in terms of the values of the projection function and may not contribute very much to the faithful approximation of f. Thus, one criterion for the selection of projection vectors is to maximise the variance of the values of the corresponding projection function. Formally, let us define

$$U = \{u \mid \exists x^i : u = a^{kl\,T} \cdot x^i \ , i = 1,...,n\} \quad (9)$$

Then the mean (expectation) and variance of $f_{kl}(u)$ values is given as

$$E(f_{kl}) = (1/|U|) \cdot \sum_{u \in U} f_{kl}(u) \quad (10)$$

$$V(f_{kl}) = (1/|U|) \cdot \sum_{u \in U} (f_{kl}(u) - E(f_{kl}))^2 \quad (11)$$

Then the optimisation aims to maximise $V(f_{kl})$ corresponding to the projection vector a^{kl}.

Another optimization idea for the projection vectors comes from the consideration of independent component analysis (ICA) [25], i.e. each projection vector and corresponding projection function may perform good approximation over some particular part of the data manifold, which may be projected to some particular part of the projection line corresponding to this projection vector, elsewhere on the projection line the projection function values may average to some constant (e.g. 0 or 1). In such case the target function would be approximated as a linear combination of the projection functions. To find projection vectors with this property we look for vectors for which the kurtosis of the projection function values is maximal, i.e. the expectation is that the projection function approximates the target function in some particular value range of the target function. Formally,

$$E_2(f_{kl}) = (1/|U|) \cdot \sum_{u \in U} (f_{kl}(u))^2 \quad (12)$$

$$E_4(f_{kl}) = (1/|U|) \cdot \sum_{u \in U} (f_{kl}(u))^4 \quad (13)$$

$$kurt(f_{kl}) = E_4(f_{kl}) / E_2(f_{kl}) - 3 \quad (14)$$

and we aim to optimise the projection vector a^{kl} by maximising $kurt(f_{kl})$.

Finally, a third optimisation idea is to look for projection vectors for which the Kullback-Leibler distance [26] of the distributions of the values of the target function f and of the values of f_{kl} is minimal, i.e. projection vectors for which the two value distributions are similar. In principle, similar value distributions may indicate similar level of discrimination between values corresponding to different function arguments, thus the projection function may help to construct a good approximation of the target function over the projection space. Formally,

$$\varphi_{kl}(v) = (1/|U|) \cdot |\{u \mid |v - f_{kl}(u)| \leq \varepsilon\}| \quad (15)$$

$$\varphi_f(v) = (1/n) \cdot |\{x^i \mid |v - f(x^i)| \leq \varepsilon\}| \quad (16)$$

$$D_{kl} = -\sum_v \varphi_f(v) \cdot \ln(\varphi_{kl}(v) / \varphi_f(v)) \quad (17)$$

for a set of appropriate v values and an appropriately chosen small $\varepsilon > 0$, such that if $|v - f(x^i)| \leq \varepsilon$ and $|v' - f(x^i)| \leq \varepsilon$ then this implies that $v = v'$, and the same is true in the context of $f_{kl}(u)$. Again, $\varepsilon = 0$ would be the best choice in principle, but in practice may not work well. Then we aim for projection vectors for which D_{kl} is minimal.

Unfortunately, none of the above ideas about optimising the projection vectors yields a straightforward way for the optimisation process, i.e. the optimized quantity is not a sufficiently direct function of the projection vector to allow the effective optimisation of the projection vector. Consequently the practical application of these methods relies on searching for the best vectors among a set of randomly generated vectors. We note that in principle various heuristic optimisation methods (e.g. particle swarm optimisation [27]) may be used for more efficient search, however, all these come with considerably increased computational costs and for this reason such methods are not considered here.

IV. Application Examples

To test the proposed random linear projection neural network approximation of functions defined over high-dimensional data we used a set of functions reported for similar purposes in earlier papers [6,9,23]. We built neural networks with 20 hidden neurons, each with a Gaussian activation function. The neural network architecture was the same for both high-dimensional and low-dimensional approximations.

We generated the high-dimensional data by mapping 5-dimensional random vectors into a 60-dimensional space, following previous papers [6,9,23]. The mapping used a deformed multiple Swiss roll manifold to calculate the 60-dimensional vectors. The relationship between the 5-dimensional $\boldsymbol{z}$ vectors and 60-dimensional $\boldsymbol{x}$ vectors is given by the following equations:

$$\boldsymbol{x}_{3\cdot(j-1)\cdot(10-j)+6\cdot(i-j-1)+1} = \lambda_{3\cdot(j-1)\cdot(10-j)+6\cdot(i-j-1)+1}\cdot\mu\cdot\boldsymbol{z}_j\cdot\cos(\boldsymbol{z}_j) \quad (18)$$

$$\boldsymbol{x}_{3\cdot(j-1)\cdot(10-j)+6\cdot(i-j-1)+2} = \lambda_{3\cdot(j-1)\cdot(10-j)+6\cdot(i-j-1)+2}\cdot\mu\cdot\boldsymbol{z}_i$$

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

$$\boldsymbol{x}_{3\cdot(j-1)\cdot(10-j)+6\cdot(i-j-1)+4} = \lambda_{3\cdot(j-1)\cdot(10-j)+6\cdot(i-j-1)+4}\cdot\mu\cdot\boldsymbol{z}_i\cdot\cos(\boldsymbol{z}_i)$$

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

$$\boldsymbol{x}_{3\cdot(j-1)\cdot(10-j)+6\cdot(i-j-1)+6} = \lambda_{3\cdot(j-1)\cdot(10-j)+6\cdot(i-j-1)+6}\cdot\mu\cdot\boldsymbol{z}_i\cdot\sin(\boldsymbol{z}_i)$$

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

$$\mu = (\sqrt{5}\cdot||\boldsymbol{z}||)^{-1}(2\cdot(1+\exp(-||\boldsymbol{z}||^2)^{-1}-1) \quad (19)$$

and $\lambda_j = 10$ for $j = 1,...,5$, $\lambda_j = 0.1$ for $j = 6,...,10$ and $\lambda_j = 1$ for $j > 10$.

In practice this means that, for any two components $\boldsymbol{z}_i$, $\boldsymbol{z}_j$ of the 5-dimensional vector, with $j<i$, we have six components of the $\boldsymbol{x}$ vector using the deformed Swiss roll equations (18). There are ten such combinations of two components $\boldsymbol{z}_i$, $\boldsymbol{z}_j$ of the 5-dimensional vector, with $j<i$, 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 $\boldsymbol{z}_i$, $\boldsymbol{z}_j$ of the 5-dimensional vector are $3\cdot(j-1)\cdot(10-j)+6\cdot(i-j-1)+t$, where t goes from 1 to 6.

The ten functions that we used for the purpose of functions approximation are adopted from previous papers [6,9,23] with minor variations. The functions are:

1) Squared modulus:

$$f_1(\boldsymbol{x}(\boldsymbol{z})) = ||\boldsymbol{z}||^2 \quad (20)$$

2) Second degree polynomial:

$$f_2(\boldsymbol{x}(\boldsymbol{z})) = (1/500)\cdot\Sigma_{j=1,4}\boldsymbol{z}_j^2\cdot\boldsymbol{z}_{j+1} \quad (21)$$

3) Exponential square sum:

$$f_3(\boldsymbol{x}(\boldsymbol{z})) = (1/5)\cdot\Sigma_{j=1,5}\exp(\boldsymbol{z}_j^2/50) \quad (22)$$

4) Exponential-sinusoid sum:

$$f_4(\boldsymbol{x}(\boldsymbol{z})) = (1/5)\cdot(\Sigma_{j=1,4}\exp(\boldsymbol{z}_j^2/50)\cdot\sin(\boldsymbol{z}_{j+1}) + \exp(\boldsymbol{z}_5^2/50)\cdot\sin(\boldsymbol{z}_1)) \quad (23)$$

5) Polynomial-sinusoid sum:

$$f_5(\boldsymbol{x}(\boldsymbol{z})) = (1/50)\cdot\Sigma_{j=1,5}\boldsymbol{z}_j^2\cdot\cos(j\cdot\boldsymbol{z}_j) \quad (24)$$

6) Inverse exponential square sum:

$$f_6(\boldsymbol{x}(\boldsymbol{z})) = 100\cdot(\Sigma_{j=1,5}\exp(\boldsymbol{z}_j^2/25))^{-1} \quad (25)$$

7) Sigmoidal:

$$f_7(\boldsymbol{x}(\boldsymbol{z})) = 10\cdot(1+\exp(-\Sigma_{j=1,5}\boldsymbol{z}_j/5))^{-1} \quad (26)$$

8) Gaussian:

$$f_8(\boldsymbol{x}(\boldsymbol{z})) = 10\cdot\exp(-\Sigma_{j=1,5}\boldsymbol{z}_j^2/100) \quad (27)$$

9) Linear:

$$f_9(\boldsymbol{x}(\boldsymbol{z})) = \Sigma_{j=1,5}\, j\cdot\boldsymbol{z}_j \quad (28)$$

10) Constant:

$$f_{10}(\boldsymbol{x(z)})=1 \quad (29)$$

To evaluate the neural network approximations we generated 20 different data sets for each function. We also generated 20 additional data sets to assess the effectiveness of the considered projection vector optimisation methods. Each data set consisted of 5000 randomly chosen training data points and 400 randomly chosen test data points. The 5-dimensional vectors for the training and testing sets were selected using uniform sampling of $[-10,10]^5$. The 60-dimensional data vectors were generated as described above.

First, we compared neural networks trained using high-dimensional data with ensembles of 20 neural networks trained with low-dimensional data using 20 different random linear projection matrices and with neural networks trained with low-dimensional data generated by a single random linear projection matrix. For the comparison we calculated the average squared error for the approximations and then averaged these errors over the 20 different data sets. The approximation performance evaluation results are presented in Table I.

The results show that the performance of the neural networks trained with low-dimensional data is statistically significantly (p-value < 0.05 calculated for a t-test for the comparison of mean values) better than the performance of the neural networks trained with high-dimensional data for eight functions (f_1, f_2, f_3, f_6, f_7, f_8, f_9, f_{10}) and comparable (not statistically significantly different) to the performance of the neural networks trained with high-dimensional data for the remaining two functions (f_4, f_5). This applies both to a single neural network trained over the projection space and to an ensemble of neural networks trained over the projection spaces.

These results confirm our expectations and theoretical reasoning in the previous section about the benefits of projecting the high-dimensional data into an appropriate lower dimension. Interestingly, these results also show that improved approximation performance can be achieved for the majority of the considered functions through a simple random projection of the high-dimensional data into the low-dimensional projection space. We note that the performance improvement is achieved for all functions if the projection is performed using a mapping that preserves the topological structure of the data manifold, as reported in earlier papers [9,23]. This means that topology preservation may be important for some, sufficiently complicated, functions, but in many cases of less complicated functions simple randomly chosen linear projections may work sufficiently well to achieve significant improvement of the neural network approximation performance.

The results on the comparison of single linear projection based neural network approximation and the approximations with the ensemble of neural networks based on a set of random linear projections (20 projections in our case) show that the ensemble version achieved statistically significantly better performance only in two cases (f_5, f_{10}) and almost statistically significantly better performance in another two cases (f_2, f_8). In two cases (f_7, f_9) the ensemble performed statistically significantly less well then the selected single projection based neural network. With one exception (f_7) the standard deviations

TABLE I. PERFORMANCE COMPARISON OF NEURAL NETWORKS TRAINED WITH HIGH-DIMENSIONAL AND LOW-DIMENSIONAL DATA

Performance measure: mean squared errors over 20 data sets			
Function	***High-dim Data – average, (standard deviation)***	***Low-dim Data, 1 projection – average, (standard deviation), high-low p-value for t-test***	***Low-dim Data, 20 projections – average, (standard deviation), high-low p-value for t-test, [1–20 p-value for t-test], {1–20 p-value for F-test}***
Squared modulus (f_1)	28787 (33245)	5385 (1477) 0.0053	5639 (1693) 0.0053 [0.9154] {0.7555}
Second degree polynomial (f_2)	33.64 (25.61)	10.81 (5.356) 8.41E-4	8.339 (1.185) 2.98E-4 [0.0584] {7.92E-9}
Exponential square sum (f_3)	5.585 (5.569)	0.6355 (0.167) 8.13E-4	0.6347 (0.1188) 8.05E-4 [0.9142] {0.0798}
Exponential-sinusoid sum (f_4)	0.9921 (0.1927)	1.1387 (0.4438) 0.1871	1.151 (0.2741) 0.091 [0.7574] {0.0028}
Polynomial-sinusoid sum (f_5)	2.434 (0.7923)	2.689 (0.7931) 0.3158	2.251 (0.3505) 0.2975 [0.0244] {2.62E-4}
Inverse exponential square sum (f_6)	50.79 (53.75)	11.17 (2.571) 0.0038	10.66 (2.008) 0.0033 [0.3216] {0.2877}
Sigmoidal (f_7)	138.9 (251.4)	8.432 (0.8278) 0.0315	23.47 (2.293) 0.0526 [1.93E-10] {2.52E-11}
Gaussian (f_8)	11.78 (13.51)	2.87 (0.853) 0.0082	2.449 (0.2945) 0.006 [0.0529] {5.41E-4}
Linear (f_9)	20765 (31258)	1869 (416.4) 0.014	2565 (500.3) 0.017 [2.84E-4] {0.2661}
Constant (f_{10})	0.7191 (0.7634)	4.25E-5 (5.12E-5) 4.72E-4	1.41E-5 (6.68E-6) 4.72E-4 [0.0442] {1.04E-6}

of the ensemble approach are significantly smaller ($p < 0.05$ calculated for the F-test for the comparison of standard deviations) or comparable to the standard deviations of the single projection based neural networks. This indicates that the ensemble approach can reduce the performance variance, but it does not make the approximation performance better on average. This implies that the benefits of averaging over a number of neural network approximations calculated with a set of random linear projections are limited and in most cases a single random projection can achieve as much as the ensemble approach in terms of improvement of the average squared approximation error.

Next, we considered the three approaches to optimise the choice of the projection vector components of the projection matrices. We had 20 different runs for each approach (variance maximisation, kurtosis maximisation, Kullback-Leibler distance minimisation), selecting the best combination of 5 projection vectors from 2000 randomly generated projection vectors. For the performance comparison we calculated the average squared error for each run and then averaged these over the 20 runs. The results of approximation performances for optimized projection matrices are presented in Table II. The corresponding approximation performance data for the randomly picked projection matrix case are shown in Table I.

The results show that in most cases the average performances of the neural networks working with low-dimensional data generated using optimized matrices is numerically better than the performance of neural networks using data projected with a randomly selected projection matrix (an exception is function f_7, for which only one of the three optimized projections leads to a better approximation performance), however the differences in most cases are not statistically significant. The most significantly better than random projection based performances are found for the projection matrices optimized for maximum standard deviation (in the case of functions f_5, f_7, f_8, f_9, f_{10}) – there is only one other case, for projection matrices optimized for maximal kurtosis for function f_8.

The results imply that picking projection vectors that maximize the standard deviation of the projection function values may give better projection matrices than the ones that can be picked randomly. However, the other two optimization approaches (kurtosis maximisation and Kullback-Leibler distance minimisation) seem to be less effective in improving the performance of the neural network approximations using projected low-dimensional data.

TABLE II. PERFORMANCE COMPARISON OF NEURAL NETWORKS TRAINED WITH LOW-DIMENSIONAL DATA FOLLOWING RANDOM PROJECTION AND PROJECTION WITH OPTIMIZED PROJECTION MATRICES – '+' MARKS LARGER MEAN AND STANDARD DEVIATION VALUES FOR OPTIMIZED PROJECTIONS

Performance measure: mean squared errors over 20 data sets			
Function	***Low-dim Data, max variance projection – average, (standard deviation), [random-optimized p-value for t-test], {random-optimized p-value for F-test}***	***Low-dim Data, max kurtosis projection – average, (standard deviation), [random-optimized p-value for t-test], {random-optimized p-value for F-test}***	***Low-dim Data, min KL distance projection – average, (standard deviation), [random-optimized p-value for t-test], {random-optimized p-value for F-test}***
Squared modulus (f_1)	5150 (1578+) [0.6303] {0.7768}	4878 (967.1) [0.208] {0.0722}	5065 (1669+) [0.5252] {0.5998}
Second degree polynomial (f_2)	9 (2.986) [0.1965] {0.0143}	9.405 (2.615) [0.3005] {0.003}	8.523 (2.799) [0.1012] {0.0068}
Exponential square sum (f_3)	0.5529 (0.1051) [0.0704] {0.0502}	0.5804 (0.0936) [0.2076] {0.0153}	0.6154 (0.1445) [0.6857] {0.5345}
Exponential-sinusoid sum (f_4)	0.9227 (0.1093) [0.0465] {9.46E-8}	1.1364 (0.3232) [0.985] {0.1759}	0.9977 (0.1445) [0.1898] {9.37E-6}
Polynomial-sinusoid sum (f_5)	2.228 (0.3366) [0.0244] {4.76E-4}	2.4292 (0.7054) [0.2802] {0.6148}	2.503 (0.3607) [0.3478] {0.0012}
Inverse exponential square sum (f_6)	11.61+ (5.397+) [0.7421] {0.0022}	10.04 (1.79) [0.1175] {0.1233}	10.48 (3.36+) [0.4756] {0.2522}
Sigmoidal (f_7)	4.469 (1.471+) [1.51E-11] {0.0158}	9.584+ (2.004+) [0.0253] {3.25E-4}	11.13+ (3.676+) [0.0043] {2.02E-8}
Gaussian (f_8)	2.298 (0.3896) [0.0111] {0.0012}	2.348 (0.3676) [0.0185] {5.85E-4}	2.479 (0.677) [0.1168] {0.3222}
Linear (f_9)	1185 (384.4) [3.91E-6] {0.7308}	1762 (215) [0.3135] {0.0059}	1877+ (764.8+) [0.9691] {0.011}
Constant (f_{10})	7.3E-6 (5.39E-6) [0.0064] {2.07E-14}	3.43E-5 (6.28E-5+) [0.6556] {0.3808}	2.26E-5 (2.51E-5) [0.1299] {0.003}

V. CONCLUSIONS

In this paper we introduced the use of random linear projections in combination with neural network approximation for improved approximation of functions defined over high-dimensional data. We assumed that the data resides on a lower dimensional manifold and then projected it using a random linear matrix into a low-dimensional space that matches the dimensionality of the data manifold. We have shown that such linear projections work well and in most of the cases of the considered function examples the neural networks using the projected data led to better approximation performance then neural networks that were trained with high-dimensional data.

Somewhat surprisingly we found that single projection matrices worked similarly well as ensembles of projection matrices with their corresponding combination of neural network approximations. We assessed three heuristics for

optimizing the projection matrices and we found that the maximisation of the variance of the values of projection functions corresponding to projection vectors (components of projection matrices) lead in some of the cases to improved performance compared to random picking of the projection matrix.

The results reported here are important since they show that computationally inexpensive, linear projections of the high-dimensional data space into a low-dimensional space work often comparably well as computationally expensive projection methods (e.g. self-organising maps [7], local linear embedding [8]) for neural network approximation of functions defined over high-dimensional data. This opens up an interesting research avenue on exploring the use of random linear projections into low-dimensional spaces of high-dimensional data for a variety of data analytics tasks, including function approximation.

Future work will include the more theoretical analysis of the approximation error for the proposed linear projection based neural network function approximation methodology. Better understanding of the nature of and contributions to the approximation error may unveil more efficient ways of computationally inexpensive optimisation of projection matrices.

REFERENCES

[1] J. Hwangbo, I. Sa, R. Siegwart and M. Hutter, "Control of a quadrotor with reinforcement learning", IEEE Robotics and Automation Letters, vol.2, pp.2096-2103, 2017.

[2] A. Conneau, H. Schwenk, Y. Le Cun, L. Barrault, "Very deep convolutional networks for text classification", Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics, pp.1107-1116, 2017.

[3] J.B. Heaton, N.G. Polson, and J.H. White, "Deep learning for finance: deep portfolios", Applied Stochastic Models in Business and Industry, vol.33, pp.3-12, 2017.

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

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

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

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

[9] P. Andras, "High-Dimensional Function Approximation With Neural Networks for Large Volumes of Data", IEEE Transactions on Neural Networks and Learning Systems, in press – available online, 2017.

[10] J. Tompson, M. Stein, Y. Le Cun, and K. Perlin, "Real-time continuous pose recovery of human hands using convolutional networks", ACM Transactions on Graphics, vol.33, art.169, 2014.

[11] C. Dubout and F. Fleuret, "Adaptive smapling for large scale boosting", Journal of Machine Learning Research, vol.15, pp.1431-1453, 2014.

[12] R.J. Durrant and A. Kaban, "Random projections as regularizers: learning a linear discriminant from fewer observations than dimensions", Machine Learning, vol.99, pp.257-286, 2015.

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

[14] 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.

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

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

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

[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] L. Breiman, "Bagging predictors", Machien Learning, vol.24, pp.123-140, 1996.

[20] T. Dietterich, "An experimental comparison of three methods for constructing ensembles of decision trees: bagging, boosting and randomization", Machine Learning, vol.40, pp.139-157, 2000.

[21] Y. Ganjisaffar, R. Caruana, C.V. Lopes, "Bagging gradient-boosted trees for high precision, low variance ranking models", Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval, pp.85-94, 2011.

[22] E. Ohn-Bar and M.M. Trivedi, "To boost or not to boost? On the limits of boosted trees for object detection", in Proceedings of the 23rd International Conference on Pattern Recognition (ICPR), 2016.

[23] P. Andras, "High-dimensional function approximation using local linear embedding", in Proceedings of the International Joint Conference on Neural Networks, 2015.

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

[25] A. Hyvarinen, "Independent component analysis by minimization of mutual information", Helsinki Univesity of Technology, Report A 46, 1997.

[26] J.M. Joyce, "Kullback-Leibler Divergence", in: M. Lovric (ed) International Encyclopedia of Statistical Science. Springer, Berlin, Heidelberg, 2011.

[27] P. Andras "A Bayesian Interpretation of the Particle Swarm Optimization and Its Kernel Extension", PLoS ONE vol.7, e48710, 2012.