ResNet-based isosurface learning for dimensionality reduction in high-dimensional function approximation with limited data

Guannan Zhang
Computer Science and Mathematics Division
Oak Ridge National Laboratory
zhangg@ornl.gov

Jacob Hinkle
Computational Science and Engineering Division
Oak Ridge National Laboratory
hinklejd@ornl.gov

Abstract

We developed a novel ResNet-based isosurface learning method for dimensionality reduction in high-dimensional function approximation. Existing methods, including sliced inverse regression [11], active subspace methods [4], ridge approximation [5, 7], reduce the dimensionality by learning an affine/linear transformation; our contribution is to extend such transformation to the nonlinear regime. Specifically, we exploited the reversible ResNets [2, 9] to learn the target functions’ isosurfaces and approximately parameterize the isosurfaces in low-dimensional spaces, so as to greatly increase the anisotropy of the original input-output map. Since the ResNet plays a different role in our method, a new loss function was designed for training the ResNets, such that a trained network can capture the nonlinearity of the isosurface. The effectiveness of our approach is demonstrated by applying it to three 2-dimensional functions for illustrating the nonlinearity of the transformation, as well as to two 20-dimensional functions for showing the improved approximation accuracy with the use of the nonlinear transformation.

1 Introduction

High-dimensional function approximation arises in a variety of science and engineering applications, where scientists or engineers relies on accurate and fast-to-evaluate approximations to replace complex and time-consuming physical models (e.g., high-resolution multiscale fluid simulation models), so as to accelerate scientific discovery or engineering design/manufacture. In most of those applications, training and validation data need to be generated by running expensive simulations (e.g. multiscale fluid dynamics codes), so that we are often in the regime of having limited data. Thus, a new approximation strategy needs to address the challenge imposed by high dimensionality and limited data in order to be deemed successful.

One way to overcome the challenge is to develop dimensionality reduction methods that can build a transformation of the input space to increase the anisotropy of the input-output map. In this work, we assume that the function has a scalar output and the input consists of high-dimensional uncorrelated variables, such that there is no intrinsically low-dimensional structure of the input manifold. In this case, instead of analyzing the input or output manifold separately, we needs to perform dimensionality reduction by learning the low-dimensional structure of the isosurface of the target function. Several types of methods have been developed for this purpose, including sliced inverse regression [11], active subspace methods [4], ridge approximation [5, 7]. Despite many successful applications of those methods, their main drawback is that they use affine/linear isosurface structures. When the isosurface of the target function is highly nonlinear, e.g., $f(x) = \sin(||x||^2)$, the number of active dimensions cannot be reduced by linear transformations. An initial attempt of nonlinear isosurface learning was conducted in [1] by analyzing local structures of isosurfaces, where its main drawback is the high online cost.
In this effort, we exploited reversible residual neural networks (ResNets)\[2,9\] to learn the target functions’ isosurfaces and approximately parameterize the isosurfaces in low-dimensional spaces, so as to greatly increase the anisotropy of the original input-output map. Reversible architectures have been developed in the literature\[8,10,6\] with the purpose of reducing memory usage in backward propagation, while our motivation is to exploit the reversibility to build bijective nonlinear transformations. Since the ResNet is used for a different purpose, we designed a novel loss function for training the ResNets, such that a well trained network can capture the nonlinearity of the isosurface. The key idea is to utilize the samples of the function gradient in the loss function to promote the objective that the trajectories of the transformation are perpendicular to the gradient direction, i.e., on the tangent plane. In addition, we also added a constraint about the determinant of the Jacobian matrix of the transformation to avoid singularity. The effectiveness of our approach is demonstrated by applying it to three 2-dimensional functions to illustrate the nonlinear nature of the transformation, as well as to two 20-dimensional functions to show the improved approximation accuracy.

2 Problem setting

We are interested in approximating a \(d\)-dimensional multivariate function of the form

\[
y = f(x), \quad x \in \Omega \subset \mathbb{R}^d,
\]

where \(\Omega\) is a bounded domain in \(\mathbb{R}^d\), the input \(x := (x_1, x_2, \ldots, x_d)^\top\) is a \(d\)-dimensional vector, and the output \(y\) is a scalar value. \(\Omega\) is equipped with a probability density function \(\rho: \mathbb{R}^d \rightarrow \mathbb{R}^+,\) i.e.,

\[
0 < \rho(x) < \infty, \quad x \in \Omega \quad \text{and} \quad \rho(x) = 0, \quad x \notin \Omega,
\]

and all the components of \(x\) are assumed to be uncorrelated, i.e., \(\mathbb{E}[x_i x_j] = 0\) for \(i \neq j\). The target function \(f\) is assumed to be differentiable, i.e., \(f \in C^1(\Omega)\), and square-integrable with respect to \(\rho\), i.e., \(\int_\Omega f^2(x) \rho(x) dx < \infty\). Our ultimate goal is to build an approximation of \(f\), denoted by \(\tilde{f}\), such that the error \(f - \tilde{f}\) is smaller than a prescribed threshold \(\varepsilon > 0\), i.e.,

\[
\| (f(x) - \tilde{f}(x)) \|_{L^2(\Omega)} < \varepsilon,
\]

where \(\| \cdot \|_{L^2}\) is the \(L^2\) norm under the probability measure \(\rho\).

Such function approximation problem arises in a variety of science and engineering applications. For example, an economically important problem to petroleum industry is how to optimize a large number of control parameters of a production system to maximize the oil production rate. Since the fine-scale oil reservoir and production simulation software is usually very time-consuming, it is necessary to develop an efficient emulator to provide accurate forecast of the production rate given samples of the parameters. In this context, \(\tilde{f}\) in \[2\] is the desired emulator where \(x\) consists of the control parameters such as the number, depth, position of wells, water injection rates and schedule, etc., and the output \(y\) is the production rate.

The main challenge in approximating \(f(x)\) is the curse of dimensionality and limited data. Since the output \(y\) is already a scalar and the input \(x\) consists of uncorrelated components, most standard manifold learning methods are not applicable. There are several existing methods targeting at this problem, including sliced inverse regression\[11\], active subspace\[4\], ridge approximation\[3\], where the dimensionality reduction could be achieved by learning and optimizing a linear transformation \(A\), such that only a small number of the transformed variable \(z = Ax\) are active. However, when the isosurface of \(f\) is nonlinear, e.g., \(f(x) = \sin(||x||_2^2)\), the dimension cannot be effectively using a linear transformation. This issue motivated us to develop a nonlinear isosurface learning approach for dimensionality reduction.

3 New method: isosurface learning via reversible ResNets

The goal of isosurface learning is to construct a bijective nonlinear transformation, denoted by

\[
z = g(x) \in \mathbb{R}^d \quad \text{and} \quad x = g^{-1}(z),
\]

such that the composite map \(y = f \circ g^{-1}(z)\) has a very small number of active input variables. In other words, the transformed vector \(z \in \mathbb{R}^d\) can be split into two parts, i.e., \(z = (z_{\text{act}}, z_{\text{inact}})\), such
that, for any fixed active variables in \(z_{\text{act}}\), the trajectory of \(x = g^{-1}(z_{\text{inact}}|z_{\text{act}})\) moves in a small neighborhood of an isosurface of \(f(x)\). As such, we can ignore \(z_{\text{inact}}\) and only approximate the low-dimensional map from \(z_{\text{act}}\) to \(y\).

To proceed, we need two types of data for training \(g\), i.e., samples of the function values and its gradients, denoted by

\[
\Xi_{\text{train}} := \left\{ \left(x^{(s)}, f(x^{(s)}), \nabla f(x^{(s)}) \right) : s = 1, \ldots, S \right\},
\]

where \(\{x^{(s)} : s = 1, \ldots, S\}\) are drawn from \(\rho(x)\), and \(\nabla f(x^{(s)})\) denotes the gradient of \(f\) at \(x^{(s)}\). We realized that \(\nabla f\) is usually not accessible in many real-world applications. In those cases, the gradient information can be approximated by finite difference or other gradient estimation methods.

### 3.1 The isosurface learning model

In this work, we defined the isosurface learning model, i.e., the nonlinear transformation \(g\) in Eq. \((3)\), as a reversible ResNet, developed in \([2,9]\). This type of ResNets was inspired by the connection between ResNets and the following Hamiltonian dynamical system, i.e.,

\[
\begin{align*}
\dot{u}(t) &= K^T_1(t) \sigma(K_1(t)u(t) + b_1(t)), \\
\dot{v}(t) &= -K^T_2(t) \sigma(K_2(t)u(t) + b_2(t)),
\end{align*}
\]

where \(u(t)\) and \(v(t)\) are partitions of the features, \(\sigma\) is the activation function. A reversible ResNet with \(N\) layers can be obtained by discretizing the above dynamical system using Verlet methods, i.e.,

\[
\begin{align*}
u_{n+1} &= u_n + h K^T_{n,1} \sigma(K_{n,1} v_n + b_{n,1}), \\
v_{n+1} &= v_n - h K^T_{n,2} \sigma(K_{n,2} u_{n+1} + b_{n,2}),
\end{align*}
\]

for \(n = 0, 1, \ldots, N-1\), where \(h\) is the “time step”. Since \(u_n, v_n\) can be immediately obtained given \(u_{n+1}, v_{n+1}\), the ResNet in Eq. \((4)\) is reversible by definition.

To define \(g : x \mapsto z\), we split \(x\) evenly into \(u_0\) and \(v_0\), and split \(z\) accordingly into \(u_N\) and \(v_N\), i.e.,

\[
x := \begin{bmatrix} u_0 \\ v_0 \end{bmatrix} \quad \text{where} \quad u_0 := (x_1, \ldots, x_{|d/2|})^\top, \quad v_0 := (x_{|d/2|+1}, \ldots, x_d)^\top,
\]

\[
z := \begin{bmatrix} u_N \\ v_N \end{bmatrix} \quad \text{where} \quad u_N := (z_1, \ldots, z_{|d/2|})^\top, \quad v_N := (z_{|d/2|+1}, \ldots, z_d)^\top,
\]

such that \(g\) is defined by the map \((u_0, v_0) \mapsto (u_N, v_N)\) from the input of the \(N\)-layer ResNets in Eq. \((4)\) to its output, i.e.,

\[
x = \begin{bmatrix} u_0 \\ v_0 \end{bmatrix} \xrightarrow{g} \begin{bmatrix} u_N \\ v_N \end{bmatrix} = z.
\]

It has been proved in \([9]\) that the ResNet in Eq. \((4)\) is guaranteed to be stable, so that we can use a very deep architecture to build a highly nonlinear transformation to capture the isosurface of \(f\).

### 3.2 The loss function

Since the ResNet plays a different role in our method, a new loss function was designed for training the ResNets \(g\). Our loss function includes three components. The first component is defined to guarantee a trained \(g\) captures the isosurface of \(f\). Our observation is that, for any given \((x^*, z^*)\) satisfying \(z^* = g(x^*)\), if \(x = g^{-1}(z)\) moves along a tangent direction of the isosurface of \(f\) under any perturbation of \(z_i\), then the function \(f(x)\) will not change with \(z_i\) in the neighborhood of \(z^*\). Guided by such observation, we write out the Jacobian matrix of \(g^{-1} : z \mapsto x\) as

\[
J_{g^{-1}}(z) = [J_1(z), J_2(z), \ldots, J_d(z)] \quad \text{with} \quad J_i(z) := \begin{bmatrix} \frac{\partial x_1}{\partial z_i}(z), \ldots, \frac{\partial x_d}{\partial z_i}(z) \end{bmatrix}^\top
\]
where the $i$-th column $J_i$ describes how $x$ moves under the perturbation of $z_i$. Hence, we can re-state our observation in another way (illustrated in Figure 1), i.e., $f(x)$ will not change with $z_i$ in the neighborhood of $z^*$ if

$$J_i(z^*) \perp \nabla f(x^*) \iff \langle J_i(z^*), \nabla f(x^*) \rangle = 0,$$

(7)

where $\langle \cdot, \cdot \rangle$ denotes the inner product. As such, the first component of the loss function is defined by

$$L_1 := \sum_{i=1}^{S} \sum_{s=1}^{d} \left[ \omega_i \left( \frac{J_i(z^{(s)})}{\|J_i(z^{(s)})\|_2}, \nabla f(x^{(s)}) \right) \right]^2,$$

(8)

where $\omega_1, \omega_2, \ldots, \omega_d$ are user-defined weights determining how strict the condition in Eq. (7) is enforced for each dimension. A naive example is that $\omega := (0, 1, 1, \ldots, 1)$, meaning the objective is to train the transformation $g$ such that the output of $f \circ g^{-1}(z)$ only changes with $z_1$, i.e., the intrinsic dimension of $f \circ g^{-1}(z)$ is one. Note that we only normalize $J_i$ in (10), but not $\nabla f$, such that $L_1$ will not penalize too much in the regions where $\nabla f$ is very small. An extreme example is that $L_1 = 0$ if $f$ is a constant function.

The second component of the loss function is to guarantee that the nonlinear transformation $g$ is non-singular. To do this, we utilized the determinant of the Jacobian matrix $J_{g^{-1}}$ to define $L_2$, i.e.,

$$L_2 := (\det(J_{g^{-1}}) - 1)^2,$$

(9)

which will push the transformation away from singularity. In addition, we also wanted to impose the uncorrelated structure on the transformed variable $z$, in order to make it easier to approximate the composite function $f \circ g^{-1}$. To do so, let $V \in \mathbb{R}^{d \times d}$ denote the empirical correlation matrix of $z$ based on $\Xi_{\text{train}}$, i.e., the sample correlation between $z_i$ and $z_j$ is described by the $i,j$-th entry of $V$ as

$$[V]_{i,j} = \frac{\sum_{s=1}^{S} (z_i^{(s)} - \overline{z_i})(z_j^{(s)} - \overline{z_j})}{\sqrt{\sum_{s=1}^{S} (z_i^{(s)} - \overline{z_i})^2 \sum_{s=1}^{S} (z_j^{(s)} - \overline{z_j})^2}},$$

where $\overline{z_i}, \overline{z_j}$ are the sample means of $z_i, z_j$. Then we introduce the third component $L_3$ into the loss function, i.e.,

$$L_3 := \|V - I\|_F = \left( \sum_{i,j=1}^{d} |[V]_{i,j}|^2 \right)^{1/2},$$

(10)

where $\| \cdot \|_F$ is the Frobenius norm and $I$ is the $d$-dimensional Identity matrix. Basically, $L_3$ promotes that all the off-diagonal entries of the correlation matrix of $z$ to be zero.

In summary, the entire loss function is defined by

$$L := \lambda_1 L_1 + \lambda_2 L_2 + \lambda_3 L_3,$$

(11)

where $\lambda_1, \lambda_2, \lambda_3$ are user-defined constants to balance the three terms.

### 3.3 Approximation of the composite function

After the transformation $g$ is trained, the composite low-dimensional function $f \circ g^{-1}$ can be approximated with many existing methods, e.g., polynomials or simple neural networks. In this work, we used fully connected neural networks with a single hidden layer to approximate $f \circ g^{-1}$. Our observation is that the single hidden layer network can be viewed as a ridge function [12], which is an appropriate choice for approximating highly anisotropic functions. An alternative is to use the sampling strategy proposed in [3] to build a ridge approximation to $f \circ g^{-1}(z)$.
4 Numerical examples

We evaluate our method on three 2-dimensional functions for visualizing the effect of our nonlinear transformation $g$ in (3), as well as two 20-dimensional functions to compare the performances of our method with brute-force polynomial and neural network approximations. All the neural network algorithms were written using Pytorch and tested on a 2017 Macbook Pro with a 3.3 GHz Intel Core i7 CPU. To make use of the automatic differentiation in Pytorch, we implemented a customized loss function in Pytorch, where the entries of the Jacobian matrix $J$ was computed using finite difference schemes.

4.1 Tests on two-dimensional functions

Here we applied the new method to the following three two-dimensional functions:

\[
\begin{align*}
    f_1(x) &= \frac{1}{2} \sin(2\pi(x_1 + x_2)) + 1 \quad \text{for } x \in [0, 1] \times [0, 1], \\
    f_2(x) &= \exp(-(x_1 - 0.5)^2 - x_2^2) \quad \text{for } x \in [0, 1] \times [0, 1], \\
    f_3(x) &= x_1^3 + x_2^3 + 0.2x_1 + 0.6x_2 \quad \text{for } x \in [-1, 1] \times [-1, 1].
\end{align*}
\]

We used one ResNet architecture for the three functions. Specifically, $u$ and $v$ in Eq. (4) are one-dimensional variables (as the total dimension is two); the number of layers is $N = 10$, i.e., 10 blocks of the form in Eq. (4) are connected; $K_n,1, K_n,2$ are $2 \times 1$ matrices; $b_{n,1}, b_{n,2}$ are 2-dimensional vectors; the activation function is $\tanh(\cdot)$; the time step $h$ is set to 0.25; stochastic gradient descent method is used to train the ResNets with the learning rate being 0.01; no regularization is applied to the network hyperparameters; the weights in Eq. (8) is set to $\omega = (0, 1)$; $\lambda_1 = \lambda_2 = \lambda_3 = 1$ for the loss function in Eq. (11); the training set includes 100 uniformly distributed samples in $\Omega$, and the validation set includes 2000 uniformly distributed samples in $\Omega$.

The results are shown in Figure 2 to 4. For $f_1$, it is known that the optimal transformation is a 45 degree rotation of the original coordinate system. Figure 2(c) shows that the trained ResNet can approximately recover the 45 degree rotation, which demonstrates that our method can also recover linear transformation. The isosurfaces of $f_2$ and $f_3$ are nonlinear, and our method successfully captured such nonlinearity, as shown in Figure 3(b), 4(b).

![Figure 2](image)

**Figure 2.** Results for $f_1(x) = 0.5 \sin(2\pi(x_1 + x_2)) + 1$. (a) The relationship between the function value and the first component of the transformed variable $z_1$ showing that $f_1 \circ g^{-1}$ is insensitive to $z_2$; (b) The gradient field (gray arrows) and the $J_2(z)$ field (black arrows) showing the orthogonality promoted by the loss function $L_1$; (c) the transformation of a cartesian mesh to the $z$ space via the trained ResNet $z = g(x)$.

4.2 Tests on 20-dimensional functions

Here we applied the new method to the following two 20-dimensional functions:

\[
\begin{align*}
    f_4(x) &= \sin(x_1^2 + x_2^2 + \cdots + x_{20}^2) \quad \text{for } x \in [0, 1]^{20}, \\
    f_5(x) &= \prod_{i=1}^{20} (1.2 - x_i^2)^{-1} \quad \text{for } x \in [0, 1]^{20}.
\end{align*}
\]

We used one ResNet architecture for the three functions. Specifically, $u$ and $v$ in Eq. (4) are 10-dimensional variables, respectively; the number of layers is $N = 30$, i.e., 30 blocks of the form in
Figure 3. Results for $f_2(x) = \exp(-(x_1 - 0.5)^2 - x_2^2)$. (a) The relationship between the function value and the first component of the transformed variable $z_1$ showing that $f_2 \circ g^{-1}$ is insensitive to $z_2$; (b) The gradient field (gray arrows) and the $J_2(z)$ field (black arrow) showing the orthogonality proposed by the loss function $L_1$; (c) the transformation of a cartesian mesh to the $z$ space via the trained ResNet $z = g(x)$.

Figure 4. Results for $f_3(x) = x_1^3 + x_2^3 + 0.2x_1 + 0.6x_2$. (a) The relationship between the function value and the first component of the transformed variable $z_1$ showing that $f_3 \circ g^{-1}$ is insensitive to $z_2$; (b) The gradient field (gray arrows) and the $J_2(z)$ field (black arrows) showing the orthogonality proposed by the loss function $L_1$; (c) the transformation of a cartesian mesh to the $z$ space via the trained ResNet $z = g(x)$.

Eq. (10) are connected; $K_{n,1}, K_{n,2}$ are $20 \times 10$ matrices; $b_{n,1}, b_{n,2}$ are 20-dimensional vectors; the activation function is tanh; the time step $h$ is set to 0.25; stochastic gradient descent method is used to train the ResNets with the learning rate being 0.01; the weight decay rate is set to 0.01 to regularize the ResNet training; $\lambda_1 = 10$ and $\lambda_2 = \lambda_3 = 1$ for the loss function in Eq. (11); the training set includes 500 uniformly distributed samples in $\Omega$, and the validation set includes 5,000 uniformly distributed samples in $\Omega$.

Figure 5. Relative sensitivity indices of the transformed function $f_4 \circ g^{-1}(z)$ for $f_4$ in Eq. (15) with various choices of anisotropic weight $w = (w_1, \ldots, w_{20})$ in Eq. (8): (a) 1 active dimension with $w_i = 0, w_i = 1$ for $i = 2, \ldots, 20$; (b) 5 active dimensions with $w_i = 0$ for $i = 1, \ldots, 5$, $w_i = 1$ for $i = 6, \ldots, 20$; (c) 10 active dimensions with $w_i = 0$ for $i = 1, \ldots, 10$, $w_i = 1$ for $i = 11, \ldots, 20$; (d) 15 active dimensions with $w_i = 0$ for $i = 1, \ldots, 15$, $w_i = 1$ for $i = 16, \ldots, 20$. 
Figure 6. Relative sensitivity indices of the transformed function $f_5 \circ g^{-1}(z)$ for $f_5$ in Eq. (16) with various choices of anisotropic weight $w = (w_1, \ldots, w_{20})$ in Eq. (8): (a) 1 active dimension with $w_1 = 0, w_i = 1$ for $i = 2, \ldots, 20$; (b) 5 active dimensions with $w_i = 0$ for $i = 1, \ldots, 5, w_i = 1$ for $i = 6, \ldots, 20$; (c) 10 active dimensions with $w_i = 0$ for $i = 1, \ldots, 10, w_i = 1$ for $i = 11, \ldots, 20$; (d) 15 active dimensions with $w_i = 0$ for $i = 1, \ldots, 15, w_i = 1$ for $i = 16, \ldots, 20$.

Figure 7. Illustration of polynomial approximation accuracy for $f_4$ in Eq. (15): (a) sorted polynomial coefficients of the third-order total degree space obtained by least squares projection with 5,000 samples; (b) approximated value v.s. exact value using 5,000 validation samples for the least squares approximation with 5,000 training samples; (c) approximated value v.s. exact value using 5,000 validation samples for compressive sensing approximation with 500 training samples.

Figure 8. Illustration of polynomial approximation accuracy for $f_5$ in Eq. (16): (a) sorted polynomial coefficients of the third-order total degree space obtained by least squares projection with 5,000 samples; (b) approximated value v.s. exact value using 5,000 validation samples for the least squares approximation with 5,000 training samples; (c) approximated value v.s. exact value using 5,000 validation samples for compressive sensing approximation with 500 training samples.

The effectiveness of the dimensionality reduction of our method is shown in Figure 5 and 6 for $f_4$ and $f_5$, respectively, with various choices of the weight $w$ in the loss function $L_1$ in Eq. (8). The significance of each transformed variable $z_i$ is described by normalized sensitivity indices calculated.
by sampling the derivatives of $f \circ g^{-1}$. The definition of $w$ provides the target anisotropy of the transformed function $f \circ g^{-1}$. For example, when we set $w_1 = 0$ and $w_i = 1$ for $i = 2, \ldots, 20$, we intended to reduce the number of active dimensions from 20 to 1. As shown in Figure 5(a) and 6(a), our method successfully reduced the sensitivities of $z_i$ for $i = 2, \ldots, 20$ to two orders of magnitude smaller than the active variable $z_1$. In fact, the intrinsic dimension of $f_4$ is 1 if transformed to the hyper-spherical coordinate system, and our method can approximately capture such transformation. On the other hand, our approach also has the flexibility to control the number of active dimensions, as shown in Figure 5(b)-(d) and 6(b)-(d) in which we allowed 5, 10, and 15 active dimensions, respectively.

Figure 9. For target function $f_4(x)$ in Eq. 15: comparison of performances of single hidden layer neural networks with and without using our dimensionality reduction approach: (a) error obtained by using a neural network with 10 hidden neurons to approximate the transformed function $f_4 \circ g^{-1}(z)$; (b) error obtained using a neural network with 10 hidden neurons to approximate the original function $f_4(x)$; (c) error obtained using a neural network with 20 hidden neurons to approximate the original function $f_4(x)$; (d) the decay of the RMSE for the case in (a); (e) the decay of the RMSE for the case in (b); (f) the decay of the RMSE for the case in (c).

Next, we show the improvement of approximation accuracy obtained by our method. Figure 7-8 illustrate the performance of Legendre polynomials in approximating $f_4$ and $f_5$. We use third-order total-degree space with 1,771 degrees of freedom. Figure 7(a) and 8(a) show the sorted coefficients obtained by least squares projection with 5,000 training samples. Due to slow decay of the coefficients, the target functions do not have the desired sparsity/anisotropy to get accurate polynomial approximations, as shown in Figure 7(b,c) and 8(b,c). On the other hand, as shown in Figure 9(b,c,e,f) and 10(b,c,e,f), when training a neural network with a single hidden layer to approximate the original function $f_4(x)$ and $f_5(x)$ using 500 training data, there are clearly overfitting issues, and the approximation accuracy is not satisfactory. When using our method, the overfitting issue is resolved and the approximation accuracy satisfies the usual requirement of a wide range of scientific applications.

1The 5,000 training samples were only used to get an accurate least squares projection, all the neural network trainings only used 500 training samples.

2We used the case of having only one active dimension to generate the results in Figure 9 and 10.
Figure 10. For target function $f_5(x)$ in Eq. 16: Comparison of performances of single hidden layer neural networks with and without using our dimensionality reduction approach: (a) error obtained by using a neural network with 10 hidden neurons to approximate the transformed function $f_5 \circ g^{-1}(z)$; (b) error obtained using a neural network with 10 hidden neurons to approximate the original function $f_5(x)$; (c) error obtained using a neural network with 20 hidden neurons to approximate the original function $f_5(x)$; (d) the decay of the RMSE for the case in (a); (e) the decay of the RMSE for the case in (b); (f) the decay of the RMSE for the case in (c).

5 Conclusion and future work

We developed a novel ResNet-based isosurface learning method for dimensionality reduction in high-dimensional function approximation. With a exclusively design loss function, the ResNet-based nonlinear transformation can effectively learn the nonlinearity of the target function’s isosurface, so that the input dimension can be significantly reduced. There are several research directions we will pursue in the future. The first the scalable implementation. Since our loss function is more complicated than the standard loss functions, it will require extra effort to improve the efficiency of the backward propagation. The second direction is to extend the current method to a gradient-free version. This will make our approach applicable to broader applications where gradient information is not accessible.

Acknowledgements

This material was based upon work supported by the Artificial Intelligence Initiative at the Oak Ridge National Laboratory (ORNL). ORNL is operated by UT-Battelle, LLC., for the U.S. Department of Energy under Contract DE-AC05-00OR22725.

References

[1] Robert A. Bridges, Chris Felder, and Chelsey Hoff. Dimension Reduction Using Active Manifolds. arXiv e-prints, page arXiv:1802.04178, Feb 2018.
[2] Bo Chang, Lili Meng, Eldad Haber, Lars Ruthotto, David Begert, and Elliot Holtham. Reversible architectures for arbitrarily deep residual neural networks. In AAAI Conference on Artificial Intelligence, 2018.
[3] Albert Cohen, Ronald Devore, Dominique Picard, and Gerard Kerkyacharian. Capturing ridge functions in high dimension from point queries. Constructive Approximation, 35:225–243, 2012.
[4] Paul G Constantine. *Active Subspaces*. Emerging Ideas for Dimension Reduction in Parameter Studies. Society for Industrial and Applied Mathematics, Philadelphia, PA, March 2015.

[5] Ronald DeVore, Guergana Petrova, and Przemyslaw Wojtaszczyk. Approximation of Functions of Few Variables in High Dimensions. *Constructive Approximation*, 33(1):125–143, June 2010.

[6] Laurent Dinh, David Krueger, and Yoshua Bengio. NICE - Non-linear Independent Components Estimation. *CoRR*, 2014.

[7] Massimo Fornasier, Karin Schnass, and Jan Vybiral. Learning Functions of Few Arbitary Linear Parameters in High Dimensions. *Foundations of Computational Mathematics*, 12(2):229–262, February 2012.

[8] Aidan N Gomez, Mengye Ren, Raquel Urtasun, and Roger B Grosse. The Reversible Residual Network - Backpropagation Without Storing Activations. *NIPS*, 2017.

[9] Eldad Haber and Lars Ruthotto. Stable architectures for deep neural networks. *Inverse Problems*, 34:014004, Jan 2018.

[10] Michael Hauser and Asok Ray. Principles of Riemannian Geometry in Neural Networks. *NIPS*, 2017.

[11] Ker-Chau Li. Sliced Inverse Regression for Dimension Reduction. *Journal of the American Statistical Association*, 86(414):316–327, June 1991.

[12] Allan Pinkus. Approximation theory of the MLP model in neural networks. *Acta Numerica*, 8:143–195, 1999.