Journal of Computational Physics 401 (2020) 108939

Contents lists available at ScienceDirect

Journal of Computational Physics

www.elsevier.com/locate/jcp

Reduced-order deep learning for flow dynamics. The interplay between deep learning and model reduction

Min Wang [a], Siu Wun Cheung [a], Wing Tat Leung [b], Eric T. Chung [c], Yalchin Efendiev [d,e,*], Mary Wheeler [b]

[a] Department of Mathematics, Texas A&M University, College Station, TX 77843, USA
[b] Institute for Computational Engineering and Sciences, The University of Texas at Austin, Austin, TX, USA
[c] Department of Mathematics, The Chinese University of Hong Kong, Shatin, New Territories, Hong Kong SAR, China
[d] Department of Mathematics & Institute for Scientific Computation (ISC), Texas A&M University, College Station, TX, USA
[e] Multiscale Model Reduction Laboratory, North-Eastern Federal University, Yakutsk, 677980, Russia

ARTICLE INFO

Article history:
Received 12 March 2019
Received in revised form 4 September 2019
Accepted 6 September 2019
Available online 23 September 2019

Keywords:
Multiscale
Upscaling
Porous media
Deep learning
Dynamics

ABSTRACT

In this paper, we investigate neural networks applied to multiscale simulations of porous media flows and discuss a design of a novel deep neural network model reduction approach for multiscale problems. Due to the multiscale nature of the medium, the fine-grid resolution gives rise to a huge number of degrees of freedom. In practice, low-order models are derived to reduce the computational cost. In our paper, we use a non-local multicontinuum (NLMC) approach [18], which represents the solution on a coarse grid for porous media flows. In specific, we construct a reduced dimensional space which takes into account multi-continuum information. The numerical solutions are then sought in such space. Using multi-layer learning techniques, we formulate and learn input-output maps constructed with NLMC on a coarse grid. We study the features of the coarse-grid solutions that neural networks capture via relating the input-output optimization to ℓ_1 minimization of flow solutions. In proposed multi-layer networks, we can learn the forward operators in a reduced way without computing them as in POD like approaches. We present soft thresholding operators as activation function, which our studies show to have some advantages. With these activation functions, the neural network identifies and selects important multiscale features which are crucial in modeling the underlying flow. Using trained neural network approximation of the input-output map, we construct a reduced-order model for the solution approximation. We use multi-layer networks for the time stepping and reduced-order modeling, where at each time step the appropriate important modes are selected. For a class of nonlinear flow problems, we suggest an efficient strategy. Numerical examples are presented to examine the performance of our method.

© 2019 Elsevier Inc. All rights reserved.

1. Introduction

Modeling of multiscale process in porous media has been of great interest in diverse applied fields. These include flow in porous media, mechanics, biological applications, and so on. However, resolving the fine-scale features of such processes

* Corresponding author.
E-mail addresses: wangmin@math.tamu.edu (M. Wang), tonycsw2905@math.tamu.edu (S.W. Cheung), wleungo@ices.utexas.edu (W.T. Leung), tschung@math.cuhk.edu.hk (E.T. Chung), efendiev@math.tamu.edu (Y. Efendiev), mfw@ices.utexas.edu (M. Wheeler).

https://doi.org/10.1016/j.jcp.2019.108939
0021-9991/© 2019 Elsevier Inc. All rights reserved.

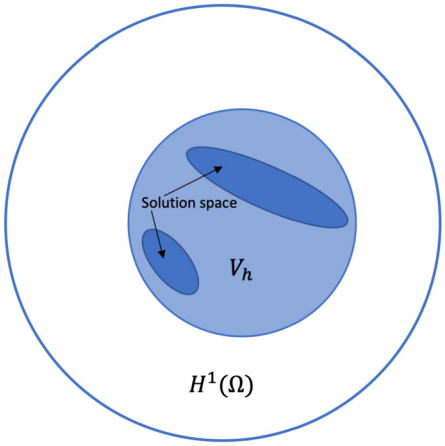


Fig. 1. Illustration of sparsity of our solution.

could be computationally expensive due to scale disparity. Researchers have been working on developing model reduction methods to bridge the information between the scales, in particular, bring the fine-grid information to coarse grid. This generally reduces to constructing appropriate approximation spaces when solving a governing PDEs numerically.

Local model reduction is an important tool in designing reduced-order models for practical applications. Many approaches have been proposed to perform model reduction, which include [3,24,5,23,15,10,11,21,2,20,25,26,33,31,28,14,17,12,13], where the coarse-grid equations are formed and the parameters are computed or found via inverse problems [7,30,8,32,37]. Adopting upscaling ideas, methods such as Generalized Multiscale Finite Element Method (GMsFEM) [23,16], Non-local Multicontinuum Methods (NLMC) [19] have been developed. They share similar ideas in constructing the approximation space, where local problems are solved to obtain the basis functions. The numerical solution is then searched within the space that constituted by such basis. With extra local computation, these multiscale spaces can capture the solution using small degrees of freedom. These methods have been proved to be successful in a number of multiscale problems. Global model reduction methods seeks a reduced dimensional solution approximation using global basis functions. When seeking numerical solutions within a high-dimensional approximation space V_h, the solutions could be sparse. Proper Orthogonal Decomposition (POD) [9] is often used to find a low-dimensional approximate representation of the solutions.

To exploit the advantages of local and global model reduction techniques, global-local model reduction approaches have been explored in the past (see, e.g., [4,27,22,36]). In global-local model reduction methods, the input-output map is approximated using POD or other approaches in combination with multiscale methods. Multiscale methods allow adaptive computing and re-computing global modes [36].

In this paper, we investigate learning strategies for multiscale methods by approximating the input-output map using multi-layer neural networks. Nowadays, neural networks have been commonly used in many applications [34]. In our paper, we use neural networks in conjunction with local reduced-order models. Our studies show that the neural networks provide some type of reduced-order global modes when considering linear problems. In the paper, we investigate these global modes and their relation to spectral modes of input-output map. With a local-global model reduction technique, we limit our search of solution to a smaller zone (see Fig. 1 for an illustration) that fits the solution populations better. Our studies show that the multi-layer network provides a good approximation of coarse-grid input-output map and can be used in conjunction with observation data. Moreover, in multi-layer networks, we can learn the forward operators in a reduced way without computing them as in POD like approaches. In our paper, we use multi-layer network for combined time stepping and reduced-order modeling, where at each time step the appropriate important modes are selected. A soft thresholding neural network is suggested based on our analysis.

Our previous work on neural networks and multiscale methods [35] have mostly focused on incorporating the observed data. In this paper, the key tasks are: (1) reproducing the coarse-scale multiscale model; (2) simultaneously determining the sparse representation of solution; and, (3) identifying the dominant multiscale modes and provide a new basis system which can be used to reduce the multiscale model dimension. There are a few advantages of using neural networks to accomplish these tasks. For (1) & (2), the computation is inexpensive and robust. For (3), the dominant multiscale modes found by the neural network can be generalized to conduct model reduction solutions. Our studies suggest the use of soft thresholding and we observe a relation between the soft thresholding and ℓ_1 minimization.

To handle the nonlinearities in PDEs, we propose the use of clustering. Clustering is an effective pretreatment for learning and prediction. Specifically, the solution of nonlinear PDEs clusters around some nonlinear values of the solution (see Fig. 1 for an illustration). Sometimes, solution clusters can be predicted without computing solutions. For example, if we are interested in the solutions to the nonlinear flow equation

$$\frac{\partial u}{\partial t} - \operatorname{div}(\kappa(t,x,u)\nabla u) = f, \tag{1}$$

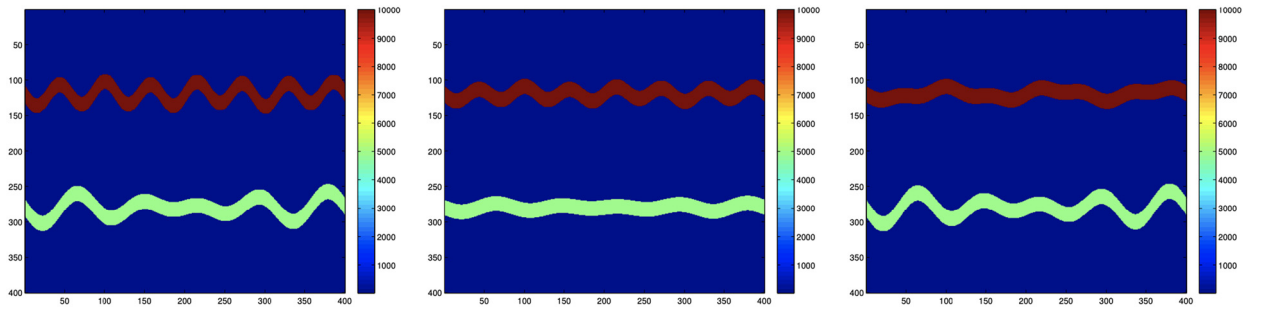


Fig. 2. Illustration of different configurations of $\hat{\kappa}(x)$. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

where $\kappa(t,x,u) = \hat{\kappa}(x)u^2$, and $\hat{\kappa}(x)$ has a limited number of configurations as shown in Fig. 2, then we could anticipate the solutions to accumulate for each configuration of $\hat{\kappa}(x)$. This is an important example of channelized permeability fields that arise in many subsurface applications. Thus, if we can use this prejudged information and divide the training samples into clusters, we are likely to get an accurate model with significantly less training resources.

The main contributions of the paper are the following. (1) We study how neural network captures the important multiscale modes related to the features of the solution. (2) We relate ℓ_1 minimization to model reduction and derive a more robust network for our problems using a soft thresholding. (3) We suggest an efficient strategy for some class of nonlinear problems that arise in porous media applications (4) We use multi-layer networks for combined time stepping and reduced-order modeling, where at each time step the appropriate important modes are selected. We remark that we can use observed data to learn multiscale model as in [35].

The paper is organized as follows. Section 2 will be a preliminary introduction of the multiscale problem with a short review on multiscale methods. A description for general neural network is also provided. Section 3 mainly focuses on discussions on the reduced-order neural network. The structure of the neural network is presented. Section 4 later discuss the proposed neural network from different aspects and propose a way to conduct model reduction with the neural network coefficient. We also present the relation between the soft thresholding neural network and a ℓ_1 minimization problem in this section. Section 5 provides various numerical examples to verify the predictive power of our proposed neural network and provide support to the claims in Section 4. Lastly, the paper is concluded with Section 6.

2. Preliminaries

2.1. Governing equations and local model reduction

We consider a nonlinear flow equation in porous media

$$\frac{\partial u}{\partial t} - \operatorname{div}(\kappa(t,x,u)\nabla u) = f, \tag{1}$$

in the domain $(0,T)\times\Omega$. Here, Ω is the spatial domain, κ is the permeability coefficient which can have a multiscale dependence with respect to space and time, and f is a source function. The weak formulation of (1) involves finding $u \in H^1(\Omega)$ such that

$$\left(\frac{\partial u}{\partial t}, v\right) + (\kappa(t,x,u)\nabla u, \nabla v) = (f, v), \quad \forall v \in H^1(\Omega). \tag{2}$$

If we numerically solve this problem in a m-dimensional approximation space $V_h \subset H^1(\Omega)$, and use an Euler temporal discretization, the numerical solution can be written as

$$u_h(t_n,x) = u_h^n = \sum_{j=1}^{m} \alpha_j^n \phi_j(x), \tag{3}$$

where $\{\phi_j\}_{j=1}^m$ is a set of basis for V_h. Moreover, the problem (2) can then be reformulated as: find $u_h \in V_h$ such that

$$\left(\frac{u_h^{n+1} - u_h^n}{\Delta t}, v_h\right) + (\kappa(t_n,x,u_h^\nu)\nabla u_h, \nabla v_h) = (f, v_h), \quad \forall v_h \in V_h, \tag{4}$$

where $\nu = n$ or $n+1$ corresponds to explicit and implicit Euler discretization respectively. Here, $(\cdot,\cdot)$ denotes the L_2 inner product.

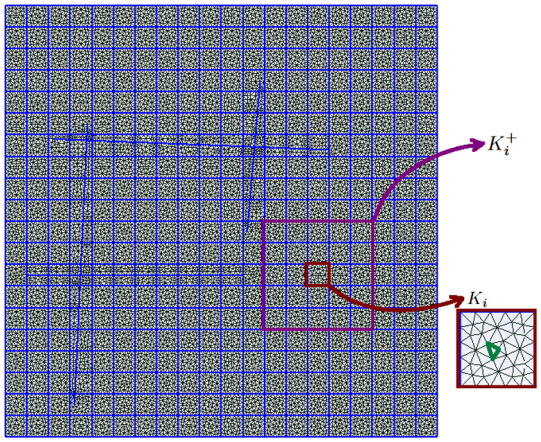


Fig. 3. Illustration of coarse and fine meshes.

For problems with multiple scales, a multiscale basis function for each coarse node is computed following the idea of upscaling, i.e., the problem can be solved with a local model reduction. Instead of using the classic piece-wise polynomials as the basis functions, we construct the local multiscale basis following NLMC [35] and use the span of all such basis function as the approximation space V_h. More specifically, for a fractured media (Fig. 3), the basis functions of (1) can be constructed by the following steps:

- Step 1: Partition of domain

$$\Omega = \Omega_m \bigoplus_n d_n \Omega_{f,n}. \tag{5}$$

Assume $\mathcal{T}^H$ is a coarse-grid partition of the domain Ω with a mesh size H which is further refined into a fine mesh $\mathcal{T}^h$. Denote by $\{K_i| \quad i = 1, \cdots, N\}$ the set of coarse elements in $\mathcal{T}^H$, where N is the number of coarse blocks. We also define the oversampled region K_i^+ for each K_i, with a few layers of neighboring coarse blocks, see Fig. 3 for the illustration of K_i and K_i^+. We further define the set of all fracture segments within a coarse block K_j as $F^j = \{f_n^{(j)} | 1 \leq n \leq L_j\} = \{\cup_n \Omega_{f,n}\} \cap K_j$ and $L_j = dim\{F^j\}$.

- Step 2: Computation of local basis function in K_i^+.

The basis for each over-sampled region $\psi_m^{(i)}$ solves the following local constraint minimizing problem on the fine grid

$$\begin{aligned}
&a(\psi_m^{(i)}, v) + \sum_{K_j \subset K_i^+} \left(\mu_0^{(j)} \int_{K_j} v + \sum_{1 \leq m \leq L_j} \mu_m^{(j)} \int_{f_m^{(j)}} v \right) = 0, \quad \forall v \in V_0(K_i^+), \\
&\int_{K_j} \psi_m^{(i)} = \delta_{ij}\delta_{0m}, \quad \forall K_j \subset K_i^+, \\
&\int_{f_n^{(j)}} \psi_m^{(i)} = \delta_{ij}\delta_{nm}, \quad \forall f_n^{(j)} \in F^{(j)}, \ \forall K_j \subset K_i^+,
\end{aligned} \tag{6}$$

where $a(u, v) = \int_{D_m} \kappa_m \nabla u \cdot \nabla v + \sum_i \int_{D_{f,i}} \kappa_i \nabla_f u \cdot \nabla_f v$, $\mu_0^{(j)}, \mu_m^{(j)}$ are Lagrange multipliers while $V_0(K_i^+) = \{v \in V(K_i^+) | v = 0 \text{ on } \partial K_i^+\}$ and $V(K_i^+)$ is the fine grid space over an over-sampled region K_i^+. By this way of construction, the average of the basis $\psi_0^{(i)}$ equals 1 in the matrix part of coarse element K_i, and equals 0 in other parts of the coarse blocks $K_j \subset K_i^+$ as well as any fracture inside K_i^+. As for $\psi_m^{(i)}, m \geq 1$, it has average 1 on the m-th fracture continua inside the coarse element K_i, and average 0 in other fracture continua as well as the matrix continua of any coarse block $K_j \subset K_i^+$. It indicates that the basis functions separate the matrix and fractures, and each basis represents a continuum.

The resulting multiscale basis space is finally written as $V_h = span\{\psi_m^{(i)} | 0 \leq m \leq L_i, 1 \leq i \leq N\}$. By renumbering the basis, we denote all basis function as $\{\phi_j\}_{j=1}^m$, where $m = \sum_{i=1}^N (L_j + 1)$.

Thus, the coefficient $U^n = (\alpha_1^n, \alpha_2^n, \dots, \alpha_m^n)^T$ satisfies the recurrence relation

$$U^{n+1} = (M + \Delta t A^\nu)^{-1}(MU^n + \Delta t F^n), \tag{7}$$

where M and A^ν are the mass matrix and the stiffness matrix with respect to the basis $\{\phi_j\}_{j=1}^m$, ν can be n or $n+1$ depending on the temporal scheme that we use. We have

$$[M]_{ij} = \int_\Omega \phi_i(x)\phi_j(x)\,dx,$$
$$[A^\nu]_{ij} = \int_\Omega \kappa(t_n, x, u_h^\nu)\nabla\phi_i(x)\cdot\nabla\phi_j(x)\,dx.$$

We claim that a global model reductions can be conducted to the problem described above, as solution $u_h^n(x)$ in many cases can be sparse in V_h even if V_h is a reduced-order space. For instance, $u_h(t_n, x)$ is strongly bonded to initial condition $u_h(t_0, x)$. It can be foreseen that if initial conditions are chosen from a small subspace of $V_h(\Omega)$, $u_h^n(x) = u_h(t_n, x)$ is also likely to accumulate somewhere in V_h. In other words, the distribution of coefficients U^n can hardly expand over the entire $\mathbb{R}^m$ space but only lies in a far smaller subspace.

Other physical restrictions to the problem could also narrow down the space of solution. This indicates that $u_h(t_n, x)$ can be closely approximated with less degrees of freedom. Section 3 and Section 4 will be discussing how to identify dominant modes in the space of U^n using a neural network.

2.2. Neural Network

A neural network is usually used to learn a map from existing data. For example, if we are given samples $\{(x_i, y_i)\}_{i=1}^L$ from the map $\mathcal{F}: X \to Y$, i.e., $\mathcal{F}(x_i) = y_i$ for $1 \le i \le L$, and we would like to predict the value of $\mathcal{F}(x_i)$ for $i = L+1, \cdots, L+M$. With the help of neural network, this problem can be reformulated as the optimization problem. The optimization takes $\{(x_i, y_i)\}_{i=1}^L$ as training samples and produces a proper network coefficient θ^* starting from some initialization chosen at random. More specifically, if the neural network $\mathcal{NN}(\cdot)$ has a feed-forward fully-connected structure, then

$$\mathcal{NN}(x;\theta) = W_n\sigma(\cdots\sigma(W_2\sigma(W_1x + b_1) + b_2)\cdots) + b_n. \tag{8}$$

Here, θ represents all tuneable coefficients in neural network $\mathcal{NN}(\cdot)$ and $\sigma(\cdot)$ is some nonlinear activation function. There are many choices of such nonlinear functions, while the most common ones used are Rectified Linear Units (ReLU) and hyperbolic tangent tanh.

For the network defined above, θ is indeed defined as $\theta = \{W_1, b_1, W_2, b_2, \cdots, W_n, b_n\}$. We will use the output $\mathcal{NN}(x_i)$ to approximate the desired output y_i. The difference between them will be measured using a cost function $\mathcal{C}(\cdot)$. For example, we can take the mean square error and the loss function is defined as

$$\mathcal{C}(\theta) = \frac{1}{N}\sum_{i=1}^N (y_i - \mathcal{NN}(x_i))^2, \tag{9}$$

which measures the average squared difference between the estimated values and the observed values. The neural network is then optimized by seeking θ^* to minimize the loss function.

$$\theta^* = \operatorname*{argmin}_\theta \mathcal{C}(\theta). \tag{10}$$

Numerically, this optimization problem can be solved with a stochastic gradient descent (SGD) type method [29]. By calculating the gradient of the loss function, the coefficient θ is iteratively updated in an attempt to minimize the loss function. This process is also referred as "training". Once the loss is minimized to a satisfactory small level, the neural network parameters θ^* is decided, and further, the overall neural network architecture (15) is constructed. The predictions can then be given as $\mathcal{NN}(x_i;\theta^*)$ for $i = L+1, \cdots, L+M$.

3. Reduced-order neural network

In this section, we present the construction of reduced-order neural network. We propose a reduced-order neural network that can model a time series. Moreover, if there exists a basis that approximates the solution for each time step with sparse coefficients, then the proposed neural network can identify such basis from the training samples. Specifically, Subsection 3.1 will discuss the macroscopic structure of the proposed neural network while Subsection 3.2 will discuss two designs of the sub-network of the network with more details. Subsection 3.3 later assembles the full multi-layer neural network.

(a) Multi-layer reduced-order neural network $\mathcal{NN}(\cdot)$

(b) Sub-network $\mathcal{N}^n(\cdot)$

Fig. 4. Reduced-order neural network structure.

3.1. Reduced-order neural network structure

We propose a reduced-order neural network as shown in Fig. 4a. Here, the full network is constituted by several sub-networks. Each sub-network $\mathcal{N}^n(\cdot)$ is expected to model a one-step temporal evolution from $\mathbf{x}^n$ to $\mathbf{x}^{n+1}$ in a time series $\vec{\mathbf{x}} = [\mathbf{x}^0, \mathbf{x}^1, \cdots, \mathbf{x}^n]$.

Sub-networks should have a general structure as shown in Fig. 4b. The specific design will vary depending on the problem we are modeling (see discussions in Subsection 3.2). The sub-network is built in a way that the input $\mathbf{x}^n$ will be first feed into a multi-layer fully-connected network named as "operation layer". This layer is intended for the neural network to capture the map between two consecutive time steps. The output of the operation layer is then fed into a soft thresholding function to impose sparsity to the solution coefficient. Lastly, the data will be processed with a "basis transform layer" in which a new basis set will be learned. With the basis set, one can represent the solution with sparse coefficients assuming such representation exists.

3.2. Sub-network

In this subsection, we present two designs of the sub-network $\mathcal{N}^n(\cdot)$. One is for modeling linear dynamics while the other is designed for nonlinear dynamics. One can choose from these two options when assembling the full network depending on the dynamics of interest. Both sub-network designs are intended to learn a new set of basis and then impose the sparsity to the solution coefficient in the new system while learning dynamics.

3.2.1. Sub-network for linear process

We first present the sub-network for modeling linear dynamics. It can be used to model the one-step flow described in (1), where we define the sub-network for the n-th time step as

$$\mathcal{N}^n(\mathbf{x}^n; \theta_n) := W_n^2 S_\gamma(W_n^1 \cdot \mathbf{x}^n + b_n). \tag{11}$$

Here, for the sub-network parameter $\theta_n = (W_n^1, W_n^2, b_n)$, W_n^1 and b_n are in the operation layer and W_n^2 works as the basis transformation layer. S_γ is the soft thresholding function defined point-wise as $S_\gamma : \mathbb{R} \to \mathbb{R}$

$$S_\gamma(x) = \text{sign}(x)(|x| - \gamma)_+ \begin{cases} x - \gamma & \text{if } x \geq \gamma, \\ 0 & \text{if } -\gamma < x < \gamma, \\ x + \gamma & \text{if } x \leq -\gamma. \end{cases} \tag{12}$$

We further require W_n^2 to be an orthogonal matrix, i.e. $(W_n^2)^T \cdot (W_n^2) = I$. To this end, we train the network (11) with respect to the cost function

$$\mathcal{C}^n(\theta_n) = \|\mathbf{x}^{n+1} - \mathcal{N}^n(\mathbf{x}^n; \theta_n)\|_2^2 + \eta_n \|(W_n^2)^T \cdot (W_n^2) - I\|_1 \tag{13}$$

with a penalty on the orthogonality constraint of W_n^2. η is a hyper coefficient for adjusting the weight of the ℓ_1 regularization term. Here, $\mathbf{x}^n$ is the input of $\mathcal{N}^n(\cdot; \theta_n)$, and $\mathcal{C}^n(\cdot)$ measures the mismatch between true solution $\mathbf{x}^{n+1}$ and the prediction $\mathcal{N}^n(\mathbf{x}^n)$ while forcing W_n^2 to be orthogonal. We remark that this cost functions is only a part of the full cost function that we will be discussed in Subsection 3.3.

With such a design, the trained neural sub-network $\mathcal{N}^n(\cdot; \theta_n^*)$ will be able to model the input-output map specified by the training data while producing a matrix W_n^2 whose columns forms an orthogonal basis in $\mathbb{R}^m$.

3.2.2. Sub-network for nonlinear process

Similar to the linear case, a sub-network for nonlinear process can also be designed. When the output $\mathbf{x}^{n+1}$ is non-linearly dependent on input $\mathbf{x}^n$, we make use of a d_n-layer feed-forward neural network $\tilde{\mathcal{N}}(\cdot)$ to approximate the input-output map. $\tilde{\mathcal{N}}(\cdot)$ will work as the operation layer of the sub-network. The output of $\tilde{\mathcal{N}}(\cdot)$ is then processed with soft-thresholding and a basis transformation layer W_n^2. We define the sub-network to be

$$\begin{aligned}\mathcal{N}^n(\mathbf{x}^n;\theta_n) &:= W_n^2 S_\gamma(\tilde{\mathcal{N}}(\mathbf{x}^n;\theta_n)),\\ \tilde{\mathcal{N}}(\mathbf{x}^n;\theta_n) &:= W_n^{1,d_n}(\cdots(\sigma(W_n^{1,3}\sigma(W_n^{1,2}\sigma(W_n^{1,1}\mathbf{x}^n+b_n^1)+b_n^2)\cdots+b_n^{d_n}),\end{aligned} \tag{14}$$

where $\theta_n = \left(W_n^{1,1}, W_n^{1,2}, \ldots, W_n^{1,d_n}, W_n^2, b_n^1, b_n^2, \ldots, b_n^{d_n}\right)$ is the sub-network parameter, and $\sigma(\cdot)$ is a nonlinear activation function. The cost function for training the sub-network parameter is again defined in (13). We also remark that if $d_n = 1$ and $\sigma = \mathbf{1}$ is the point-wise identify function, then the network structure (14) is reduced to (11).

Another approach to reduce the difficulty of reproducing a nonlinear process is clustering. Instead of using a single network to approximate complicated nonlinear relations, we use different networks for different data cluster. As discussed in the Section 1, the clusters of the solutions can be predicted. Thus, separate the training samples by cluster can be an easy and effective way to accurately recover complicated process. Discussions on clustering and numerical examples are presented in Section 5.5.

3.3. Multi-layer reduced order network

Now we construct the full neural network $\mathcal{NN}(\cdot;\theta)$ by stacking up the sub-network $\mathcal{N}^n(\cdot;\theta_n)$. More precisely, $\mathcal{NN}(\cdot;\theta):\mathbb{R}^m \to \mathbb{R}^m$ is defined as:

$$\mathcal{NN}(\mathbf{x}^0;\theta) := \mathcal{N}^n(\cdots\mathcal{N}^1(\mathcal{N}^0(\mathbf{x}^0;\theta_0);\theta_1)\cdots;\theta_n), \tag{15}$$

where $\mathcal{N}^n(\cdot;\theta_n)$ is defined as in (11) or (14) depending the linearity of the process, and $\theta = (\theta_0, \theta_1, \ldots, \theta_n)$ is the full network parameter. We use such $\mathcal{NN}(\cdot;\theta)$ to approximate time series $\vec{\mathbf{x}} = [\mathbf{x}^0, \mathbf{x}^1, \cdots, \mathbf{x}^{n+1}]$. Denote the output of $(t+1)$-fold composition of sub-network $\mathcal{N}^t$ as

$$\mathbf{o}^{t+1} := \mathcal{N}^t(\cdots\mathcal{N}^1(\mathcal{N}^0(\mathbf{x}^0;\theta_0);\theta_1)\cdots;\theta_t). \tag{16}$$

Then $\mathbf{o}^{t+1}$ works as a prediction of the solution at time $t+1$. The full cost function is then defined as

$$\mathcal{C}(\theta) = \sum_{t=0}^{n} \|\mathbf{o}^{t+1} - \mathbf{x}^{t+1}\|_2^2 + \eta_t \|(W_t^2)^T \cdot (W_t^2) - I\|_1, \tag{17}$$

where $\vec{\mathbf{x}}$ is the true time sequence, and η_t is hyper-parameter stands for the weight of the regularizer while θ represents all tuneable parameters of $\mathcal{NN}(\cdot)$. Each layer (sub-network) corresponds to a one-step time evolution of the dynamics.

Suppose we have L training samples,

$$\vec{\mathbf{x}}_i = [(\mathbf{x}_i^0), (\mathbf{x}_i^1, \mathbf{x}_i^2, \cdots, \mathbf{x}_i^{n+1})], \qquad 1 \le i \le L.$$

The optimal parameter θ^* of $\mathcal{NN}(\cdot)$ is then determined by optimizing the cost function $\mathcal{C}(\theta)$ subject to this training set $\{\vec{\mathbf{x}}_i\}_{i=1}^L$ as discussed Section 2.2. Once θ^* is decided, predictions can be made for testing samples $[\mathbf{x}_i^1, \cdots, \mathbf{x}_i^{n+1}]$ by $\mathcal{NN}(\mathbf{x}_i^0), i > L$.

4. Discussions and applications

In this section, we discuss some theoretical aspects of proposed neural networks. Specifically, we use the proposed network to model fluid dynamics in heterogeneous media, as described in (1). First, we relate the soft thresholding network with ℓ_1 optimization problem. Secondly, we explore how learned coefficients of neural network are related to the map that is being approximated. Thirdly, based on the understanding of proposed neural network, we present a way to utilize the trained network coefficient to construct a reduced-order model.

Specifically, we consider a one-layer neural network for single-step linear dynamics

$$\mathcal{NN}(\mathbf{x};\theta) = \mathcal{N}(\mathbf{x};\theta) := W^2 S_\gamma(W^1\mathbf{x} + b), \tag{18}$$

omitting the indices for time step n. We train the neural network $\mathcal{NN}(\cdot;\theta)$ with data pairs $\{(U_i^n, U_i^{n+1})\}_{i=1}^L$, which are NLMC solution coefficients (see Section 2.1 for more details of data generation) to (1) at t^n and t^{n+1}, respectively. More precisely, we consider the linear case of (1), when $\kappa(t,x)$ is independent of u_h^n. In this case, the one step fluid dynamics (7) is indeed a linear revolution such that A^n is only dependent on time. Further, we denote $\hat{W}_n$ and $\hat{b_n}$ by

$$\begin{aligned}\hat{W} &= (M + \Delta t A^n)^{-1} M, \\ \hat{b} &= (M + \Delta t A^n)^{-1} \Delta t F^n.\end{aligned} \tag{19}$$

Then

$$U_i^{n+1} = \hat{W} U_i^n + \hat{b}, \quad 1 \le i \le L, \tag{20}$$

for all training samples that we use in this section. We further define the linear map between U_i^n and U_i^{n+1} as $\hat{\mathcal{L}}(\cdot) : \mathbb{R}^m \to \mathbb{R}^m$

$$\hat{\mathcal{L}}(\mathbf{x}) := \hat{W}\mathbf{x} + \hat{b}. \tag{21}$$

From now on, when there is no risk of confusion, we drop the optimal network parameter θ^* in the trained network $\mathcal{NN}(\cdot)$. We expect $\mathcal{NN}(\cdot)$ to learn the map $\hat{\mathcal{L}}(\cdot)$ from the data while extracting a system W^2 in which the data U_i^{n+1} can be represented sparsely.

4.1. Sparsity and ℓ_1 minimization

To understand the trained neural network $\mathcal{NN}(\cdot)$, we first assume the following.

Assumption 1. *For a subspace $S \subset \mathbb{R}^m$, there exist some orthogonal matrix $\hat{W}^2 \in \mathbb{R}^{m\times m}$ such that $(\hat{W}^2)^T \hat{\mathcal{L}}(\mathbf{x})$ can be approximated by a sparse vector in $\mathbb{R}^m$ for any $\mathbf{x} \in S$. More precisely, there exist an approximation error function $\epsilon(\cdot) : \mathbb{N} \to \mathbb{R}^+$, such that for any $\mathbf{x} \in S$, there exist a corresponding s-sparse vector $\mathbf{y}_s \in \mathbb{R}^m$ satisfies*

$$\|(\hat{W}^2)^T \hat{\mathcal{L}}(\mathbf{x}) - \mathbf{y}_s\|_2 \le \epsilon(s). \tag{22}$$

We then take $\{U_i^n\}_{i=1}^L$ from S and define $U_i^{n+1,\hat{W}^2} := (\hat{W}^2)^T U_i^{n+1}$ for all training pairs (U_i^n, U_i^{n+1}) of $\mathcal{NN}(\cdot)$. By Assumption 1, there exist s-sparse vector $U_i^s \in \mathbb{R}^m$ such that

$$||U_i^{n+1,\hat{W}^2} - U_i^s||_2 \le \epsilon(s) \tag{23}$$

for $1 \le i \le L$.

Additionally, for any orthogonal matrix $W^2 \in \mathbb{R}^{m\times m}$, we define $U_i^{n+1,W^2} := (W^2)^T U_i^{n+1}$ and denote it as $[\alpha_{i,1}^{n+1,W^2}, \alpha_{i,2}^{n+1,W^2}, \cdots, \alpha_{i,m}^{n+1,W^2}]^T$. Recall (3), and the corresponding numerical solution $u_{h,i}^{n+1}$ at time step $n+1$ can be written as

$$u_{h,i}^{n+1} = \Phi U_i^{n+1},$$

letting $\Phi = [\phi_1, \phi_2, \cdots, \phi_m]$ be the multiscale basis functions, and U_i^{n+1} be the corresponding coefficient vector. By the orthogonality of W^2, $u_{h,i}^{n+1}$ can be further written as

$$u_{h,i}^{n+1} = \Phi(W^2) U_i^{n+1,W^2} = \Phi^{W^2} U_i^{n+1,W^2}, \tag{24}$$

with Φ^{W^2} defined as

$$\Phi^{W^2} := \Phi W^2. \tag{25}$$

We further denote the columns of Φ^{W^2} as $\phi_j^{W^2}$'s. Then $\{\phi_j^{W^2}\}_{j=1}^m$ is actually a new set of multiscale basis functions such that $u_{h,i}^{n+1}$ can be written as

$$u_{h,i}^{n+1} = \sum_{j=1}^m \alpha_{i,j}^{n+1,W^2} \phi_j^{W^2}. \tag{26}$$

If W^2 is taken to be $\hat{W}^2$ as in Assumption 1, we will obtain

$$u_{h,i}^{n+1} = \sum_{j=1}^m \alpha_{i,j}^{n+1,\hat{W}^2} \phi_j^{\hat{W}^2},$$

where the coefficient $U_i^{n+1,\hat{W}^2} = [\alpha_{i,1}^{n+1,\hat{W}^2}, \alpha_{i,2}^{n+1,\hat{W}^2}, \cdots, \alpha_{i,m}^{n+1,\hat{W}^2}]^T$ can be closely approximated by a sparse vector U_i^s.

We then claim that our proposed neural network $\mathcal{NN}(\cdot)$ is able to approximate such $\hat{W}^2$ and a sparse approximation of the output from data. This is guaranteed by the following lemma from [6]:

Lemma 4.1. *We define* $\hat{\mathcal{N}}(\cdot), \mathbb{R}^m \to \mathbb{R}^m$ *as*

$$\hat{\mathcal{N}}(\mathbf{x}) := S_\gamma(W\mathbf{x} + b).$$

The output of $\hat{\mathcal{N}}(\mathbf{x})$ *is the solution of the* ℓ_1*-regularized problem*

$$\mathbf{y}^* = \underset{\mathbf{y}\in\mathbb{R}^m}{argmin} \frac{1}{2}\|\mathbf{y} - (W\mathbf{x} + b)\|_2^2 + \gamma\|\mathbf{y}\|_1 \tag{27}$$

by proximal gradient update.

Proof. It is straightforward to see that the directional derivative of the residual with respect to $\mathbf{y}$ is given by $\mathbf{y} - (W\mathbf{x} + b)$. On the other hand, the soft thresholding operator is the proximal operator of the ℓ_1 norm, i.e.

$$S_\gamma(x) = \text{prox}_{\gamma\|\cdot\|_1}(\mathbf{x}) = \underset{\mathbf{y}\in\mathbb{R}^m}{\text{argmin}} \left(\frac{1}{2}\|\mathbf{y} - \mathbf{x}\|_2^2 + \gamma\|\mathbf{y}\|_1 \right).$$

With a zero initial guess, the 1-step proximal gradient update of (27) with a step size γ is therefore

$$\mathbf{y}^* = \hat{\mathcal{N}}(\mathbf{x}) = S_\gamma(W\mathbf{x} + b). \quad \square$$

Thus, for the one-step neural network $\mathcal{NN}(\cdot)$ defined in (18), Lemma 4.1 implies that

$$\mathcal{NN}(\mathbf{x}) = \underset{\mathbf{y}\in\mathbb{R}^m}{\text{argmin}} \frac{1}{2}\|\mathbf{y} - W^2W^1\mathbf{x} + W^2b\|_2^2 + \gamma\|(W^2)^T\mathbf{y}\|_1. \tag{28}$$

That is to say, the output of the trained neural network $\mathcal{NN}(\cdot)$ is actually the solution to a ℓ_1 optimization problem. We further define the linear operator $\mathcal{L}(\cdot) : \mathbb{R}^m \to \mathbb{R}^m$ as

$$\mathcal{L}(\mathbf{x}) := W^2W^1\mathbf{x} + W^2b. \tag{29}$$

Equation actually (28) implies that

$$\mathcal{L}(\cdot) \approx \mathcal{NN}(\cdot) \tag{30}$$

as $\mathcal{NN}(\mathbf{x})$ minimizes $\|\mathbf{y} - W^2W^1\mathbf{x} + W^2b\|_2^2$. Moreover, the output of $\mathcal{NN}(\cdot)$ is sparse in the coordinate system W^2 as it also minimized the $\|(W^2)^T\mathbf{y}\|_1$ term.

S_γ is widely used in ℓ_1-type optimization for promoting sparsity and extracting important features as discussed above. It is therefore also brought into network to extract sparsity from the training data. For other network defined with activation functions such as ReLU, we also remark there's an correlation between S_γ and ReLU. Recall its definition in (12):

$$S_\gamma(x) = \text{sign}(x)(|x| - \gamma)_+ \begin{cases} x - \gamma & \text{if } x \geq \gamma, \\ 0 & \text{if } -\gamma < x < \gamma, \\ x + \gamma & \text{if } x \leq -\gamma, \end{cases}$$

and that of $\text{ReLU} : \mathbb{R} \to \mathbb{R}$

$$\text{ReLU}(x) = \max\{x, 0\} = \begin{cases} x & \text{if } x \geq 0, \\ 0 & \text{if } x < 0. \end{cases}$$

We have the following:

One can explicitly represent the soft thresholding operator S_γ by the ReLU function as

$$S_\gamma(x) = \text{ReLU}(x - \gamma) - \text{ReLU}(-x - \gamma) \quad \forall x \in \mathbb{R}, \tag{31}$$

or in an entry-wise sense, one can write

$$S_\gamma(\mathbf{x}) = J_m\text{ReLU}(J_m^T\mathbf{x} - \gamma\mathbf{1}_{2m}) \quad \forall \mathbf{x} \in \mathbb{R}^m, \tag{32}$$

where $J_m = [I_m, -I_m]$. Activation functions $S_\gamma(\cdot)$ can thus be easily implemented with the help of ReLU. Further, it also means that our proposed neural network is only a special class of neural networks that are defined with ReLU.

4.2. Linear operator $\hat{\mathcal{L}} \approx \mathcal{NN}$

For neural network $\mathcal{NN}(\cdot)$ as defined in (18)

$$\mathcal{NN}(\mathbf{x}) = W^2 S_\gamma(W^1\mathbf{x} + b),$$

we claim the following

Lemma 4.2. *We assume Assumption 1 holds. There exist a set of parameters $(W^1, W^2, b) \in \mathbb{R}^{m\times m} \times \mathbb{R}^{m\times m} \times \mathbb{R}^m$ such that*

$$\|\mathcal{NN}(\mathbf{x}) - \hat{\mathcal{L}}(\mathbf{x})\|_2 \le 2\epsilon(s) + s^{\frac{1}{2}}\gamma, \quad \forall x \in S. \tag{33}$$

Proof. By Assumption 1, there exist some orthogonal matrix $\hat{W}^2 \in \mathbb{R}^{m\times m}$ such that for all $\mathbf{x} \in S$, we have

$$\|(\hat{W}^2)^T\hat{\mathcal{L}}(\mathbf{x}) - \mathbf{y}_s\|_2 \le \epsilon(s), \tag{34}$$

where $\mathbf{y}_s$ is an s-sparse vector. Next, we consider $W^1 = (\hat{W}^2)^T\hat{W}$, $W^2 = \hat{W}^2$ and $b = (\hat{W}^2)^T\hat{b}$. We recall the definition of $\hat{\mathcal{L}}(\cdot)$

$$\hat{\mathcal{L}}(\mathbf{x}) := \hat{W}\mathbf{x} + \hat{b}.$$

The difference between $\mathcal{NN}(\mathbf{x})$ and $\hat{\mathcal{L}}(\mathbf{x})$ can then be estimated by

$$\begin{aligned}
\|\mathcal{NN}(\mathbf{x}) - \hat{\mathcal{L}}(\mathbf{x})\|_2 &= \|\hat{W}^2 S_\gamma\Big((\hat{W}^2)^T\hat{W}\mathbf{x} + (\hat{W}^2)^T\hat{b}\Big) - \hat{\mathcal{L}}(\mathbf{x})\|_2 \\
&= \|\hat{W}^2\Big(S_\gamma\Big((\hat{W}^2)^T\hat{W}\mathbf{x} + (\hat{W}^2)^T\hat{b}\Big) - (\hat{W}^2)^T\hat{\mathcal{L}}(\mathbf{x})\Big)\|_2 \\
&= \|\hat{W}^2\Big(S_\gamma\Big((\hat{W}^2)^T\hat{\mathcal{L}}(\mathbf{x})\Big) - (\hat{W}^2)^T\hat{\mathcal{L}}(\mathbf{x})\Big)\|_2 \\
&= \|S_\gamma\Big((\hat{W}^2)^T\hat{\mathcal{L}}(\mathbf{x})\Big) - (\hat{W}^2)^T\hat{\mathcal{L}}(\mathbf{x})\|_2.
\end{aligned}$$

Since $|S_\gamma(z_2) - S_\gamma(z_1)| \le |z_2 - z_1|$, $\forall z_1, z_2 \in \mathbb{R}$, we have

$$\|S_\gamma\Big((\hat{W}^2)^T\hat{\mathcal{L}}(\mathbf{x})\Big) - S_\gamma(\mathbf{y}_s)\|_2 \le \|(\hat{W}^2)^T\hat{\mathcal{L}}(\mathbf{x}) - \mathbf{y}_s\|_2 \le \epsilon(s).$$

Thus, we obtain

$$\begin{aligned}
&\|S_\gamma\Big((\hat{W}^2)^T\hat{\mathcal{L}}(\mathbf{x})\Big) - (\hat{W}^2)^T\hat{\mathcal{L}}(\mathbf{x})\|_2 \\
\le&\|S_\gamma\Big((\hat{W}^2)^T\hat{\mathcal{L}}(\mathbf{x})\Big) - S_\gamma(\mathbf{y}_s)\|_2 + \|S_\gamma(\mathbf{y}_s) - \mathbf{y}_s\|_2 + \|\mathbf{y}_s - (\hat{W}^2)^T\hat{\mathcal{L}}(\mathbf{x})\|_2 \\
\le&2\epsilon(s)_+ \|\mathbf{y}_s - S_\gamma(\mathbf{y}_s)\|.
\end{aligned}$$

Since $|(S_\gamma(\mathbf{y}_s) - \mathbf{y}_s)_i| \le \gamma$, we have

$$\|S_\gamma(\mathbf{y}_s) - \mathbf{y}_s\|_2^2 = \sum_{(\mathbf{y}_s)_i \neq 0} |(S_\gamma(\mathbf{y}_s) - \mathbf{y}_s)_i|^2 \le s\gamma^2,$$

and therefore we have

$$\|\mathcal{NN}(\mathbf{x}) - \hat{\mathcal{L}}(\mathbf{x})\|_2 \le 2\sigma(s) + s^{\frac{1}{2}}\gamma,$$

letting $W^1 = (\hat{W}^2)^T\hat{W}$, $W^2 = \hat{W}^2$ and $b = (\hat{W}^2)^T\hat{b}$. □

Since $\mathcal{NN}(\cdot)$ is trained with (U_i^n, U_i^{n+1}), where $U_i^n \in S$ and $\hat{\mathcal{L}}(U_i^n) = U_i^{n+1}$, we have

$$\mathcal{NN}(\mathbf{x}) \approx \hat{\mathcal{L}}(\mathbf{x}), \quad \mathbf{x} \in S. \tag{35}$$

More specifically, this approximation error $\|\mathcal{NN}(\mathbf{x}) - \hat{\mathcal{L}}(\mathbf{x})\|_2$ is small for all $\mathbf{x} \in S$ providing sufficient training. Therefore, by Lemma 4.2, we claim the trained parameters closely approximate the optimal choice to guarantee the small error indicated in (33), i.e.

$$W^2 \approx \hat{W}^2, \quad W^2W^1 \approx \hat{W}, \quad \text{and} \quad W^2b = \hat{b}.$$

However, due to the high dimension of $\hat{W}$, full recovery of $\hat{W}$ requires enormous number of training and is thus impractical. However, by enforcing $\mathcal{NN}(\mathbf{x}) = \hat{\mathcal{L}}(\mathbf{x})$ for $\mathbf{x} \in S$, the neural network learns a set of parameters $W^2W^1 \neq \hat{W}$, and $W^2b \neq \hat{b}$ such that they function similarly as $\hat{\mathcal{L}}(\cdot)$ on the subset S in the sense of linear operator. A validation of this is later provided in Subsection 5.1. Recall definition of $\mathcal{L}(\cdot)$ in (29) and the fact that it can approximate $\mathcal{NN}(\cdot)$ as in (30), we claim the linear operator have the following property:

$$\mathcal{L}(\mathbf{x}) \approx \hat{\mathcal{L}}(\mathbf{x}) \quad \forall \mathbf{x} \in S. \tag{36}$$

In the following subsection, we further construct a reduced-order model with the help of $\mathcal{L}(\cdot)$.

4.3. Model reduction with W^2

In this subsection, we further assume the s-sparse vector $\mathbf{y}_s$ in Assumption 1 has non-zero entries only at fixed coordinates for all $\mathbf{x}$ in S. That is to say, we have a fix reordering $\{j_k\}_{k=1}^s$ for $\{1, 2 \cdots m\}$, such that $(\mathbf{y}_s)_{j_k} = 0$ for $s+1 \leq k \leq m$.

Then, we will be able to utilize the coordinate system $W^2 \approx \hat{W}^2$ learned through training network to construct a reduced-order operator $\mathcal{L}_s(\cdot)$, such that it can approximate the linear map $\hat{\mathcal{L}}(\cdot)$ and maps $\mathbf{x}$ in S to a s-sparse vector in $\mathbb{R}^m$. To do so, we first define $\mathcal{L}(\cdot)$ form learned coefficients of $\mathcal{NN}(\cdot)$ as in last subsection, and $\mathcal{L}_s(\cdot)$ will be exactly a truncation of it.

Moreover, let the new basis set $\{\phi_j^{W^2}\}_{j=1}^m$ be defined with trained coefficient W^2 as in (25). When truncating W^2 in $\mathcal{L}(\cdot)$, we also determine the dominant basis among $\{\phi_j^{W^2}\}_{j=1}^m$ simultaneously. Thus, we can view the model reduction from another aspect that we actually drop the basis with less significance and represent the solution with only the dominant multiscale modes.

To construct such $\mathcal{L}_s(\cdot)$, we follow the steps:

1. Find the dominant coordinates of outputs $\mathcal{NN}(\cdot)$ in the system W^2.
   (a) Compute W^2 system coefficients of $\mathcal{NN}(U_i^n)$ for all training samples by
   $$O_i^{n+1,W^2} := (W^2)^T \mathcal{NN}(U_i^n), \quad 1 \leq i \leq L,$$
   where i refers to the sample index. Notice $(W^2)^T\mathcal{NN}(\mathbf{x})$ is sparse for $\mathbf{x} \in S$ by (28), therefore O_i^{n+1,W^2} is also sparse.
   (b) Calculate the quadratic mean of $\{O_i^{n+1,W^2}\}_{i=1}^L$ over all samples, coordinate by coordinate:
   $$S_j := \frac{1}{L}\left(\sum_{i=1}^{L} |O_{i,j}^{n+1,W^2}|^2\right)^{\frac{1}{2}}, \quad 1 \leq j \leq m.$$
   (c) Sort the quadratic mean value S_j in descending order and denoted the reordered sequence as $\{S_{j_k}\}_{k=1}^m$.
2. Keep the dominant j_k-th columns of W^2 for $k = 1, \cdots, s$. Then let the rest columns be zero. Thus, we construct a reduced-order coordinate system $W^{2,s} \in \mathbb{R}^{m\times m}$. Consequently, $\mathbf{y} = \hat{\mathcal{L}}(\mathbf{x})$ for any $\mathbf{x} \in S$ can be approximated with the reduced-order system W^2 as an s-sparse vector $\mathbf{y}^{W^{2,s}} := (W^{2,s})^T\mathbf{y}$, and
   $$\mathbf{y}^{W^{2,s}} \approx (W^2)^T\mathbf{y}. \tag{37}$$
   Consequently, for training/testing samples, we have $U_i^{n+1,W^{2,s}} \approx U_i^{n+1,W^2}$. Thus, corresponding function $u_{h,i}^{n+1}$ can be approximated with only basis $\{\phi_{j_k}^{W_2} | 1 \leq k \leq s\}$ correspond to dominant multiscale modes.
   $$u_{h,i}^{n+1} \approx \Phi^{W^2} U_i^{n+1,W^{2,s}} = \sum_{k=1}^{s} \alpha_{i,j_k}^{n+1,W^2} \phi_{j_k}^{W^2}. \tag{38}$$
3. We finally define the reduced linear operator $\mathcal{L}_s(\cdot) : \mathbb{R}^m \to \mathbb{R}^m$ as
   $$\mathcal{L}_s(\mathbf{x}) = W^{2,s}W^1 \cdot \mathbf{x} + W^{2,s}b, \quad \mathbf{x} \in \mathbb{R}^m. \tag{39}$$
   Here, the output of $\mathcal{L}_s(\cdot)$ is an s-sparse vector in $\mathbb{R}^m$.

This algorithm is designed based on the fact that $O_i^{n+1,W^2} \approx (W^2)^T U_i^{n+1} = U_i^{n+1,W^2}$ as $\mathcal{NN}(\cdot)$ is fully trained with (U_i^n, U_i^{n+1}). Thus, the order of S_j can reflect not only the significance of the coordinates of the output of $\mathcal{NN}(\cdot)$ but also that of $(W^2)^T\mathcal{L}(\mathbf{x})$ for all $\mathbf{x}$ in S. Moreover, the existence of the sparse approximation $\mathbf{y}_s$ to $(\hat{W}^2)^T\hat{\mathcal{L}}(\mathbf{x})$ as described in Assumption 1 guarantees the effectiveness of the ordering.

We then claim that this reduced-order linear operator $\mathcal{L}_s(\cdot)$ can approximate the true input-output map $\hat{\mathcal{L}}(\cdot)$ on S: Since $\mathcal{L}_s(\cdot)$ is simply a truncation of $\mathcal{L}(\cdot)$, we have:

$$\mathcal{L}_s \longrightarrow \mathcal{L}, \quad \text{as} \quad s \to m. \tag{40}$$

Moreover, recall (36)

$$\hat{\mathcal{L}}(\mathbf{x}) \approx \mathcal{L}(\mathbf{x}), \quad \forall \mathbf{x} \in S,$$

it implies the following

$$\hat{\mathcal{L}}(\mathbf{x}) \approx \mathcal{L}_s(\mathbf{x}), \quad \forall \mathbf{x} \in S. \tag{41}$$

This property of $\mathcal{L}_s(\cdot)$ provides us a way to represent the projected vectors $\hat{\mathcal{L}}(\mathbf{x})$ for $\mathbf{x} \in S$ using a vector with only s nonzero coefficients, which corresponds to a reduced multiscale model to represent the class of solution u_h^{n+1} that we are interested in.

Numerical examples are presented in Section 5.3 to verify this claim, from which we actually observe that s can be taken as a fraction of the original number of multiscale basis m to give the approximation in (41).

5. Numerical experiment

In this section, we present numerical examples in support of the previous discussion on the reduced-order neural network. Specifically, Section 5.1 demonstrates the $\mathcal{L}(\cdot)$ and $\hat{\mathcal{L}}(\cdot)$ functions similarly on a subspace by comparing the eigenvalues of the two operators; Section 5.2 demonstrates that a one-layer soft thresholding neural network can accurately recover a linear dynamics with a sparse coefficient vector; Section 5.3 then uses the learned coefficient W^2 from the one-layer neural network to conduct model reduction as describe in Section 4.3; Section 5.4 later presents the predicting results for multi-layer reduced-order neural network which corresponds to Section 3.3; and Section 5.5 applies the clustering scheme to nonlinear process modeling. All the network training are performed using the Python deep learning API TensorFlow [1].

In our numerical experiments, we use channelized/fractured permeability fields, which occur in subsurface applications. We use a fixed permeability in all simulations throughout the paper except for Section 5.5. We remark that we have investigated the performance of neural networks when we have different permeability fields. If we are dealing with different permeability fields or a family of permeability fields, the training needs to include representative data sets. In general, we did not test our proposed neural networks for many different data sets and we cant make any definite conclusions about the generalization capability. This is partly due to CPU time required for training for PDE models that we run. We plan to consider generalizations in our future works.

5.1. $\hat{\mathcal{L}} \approx \mathcal{L}$ in a subspace of $\mathbb{R}^m$

We recall the one-layer neural network for a single-step linear dynamics in (18)

$$\mathcal{NN}(\mathbf{x}) := W^2 S_\gamma (W^1 \mathbf{x} + b),$$

and the definition of $\hat{\mathcal{L}}(\cdot)$ and $\mathcal{L}(\cdot)$ in (21) and (29) respectively:

$$\hat{\mathcal{L}}(\mathbf{x}) := \hat{W}\mathbf{x} + \hat{b}, \qquad \mathcal{L}(\mathbf{x}) = W^2 W^1 \mathbf{x} + W^2 b,$$

where $\hat{W}_n$ and $\hat{b}_n$ are defined as in (19), while W_n^2, W_n^1 are trained parameters of $\mathcal{NN}(\cdot)$. We also recall (36):

$$\mathcal{L}|_S \approx \hat{\mathcal{L}}|_S,$$

for $S \subset \mathbb{R}^m$.

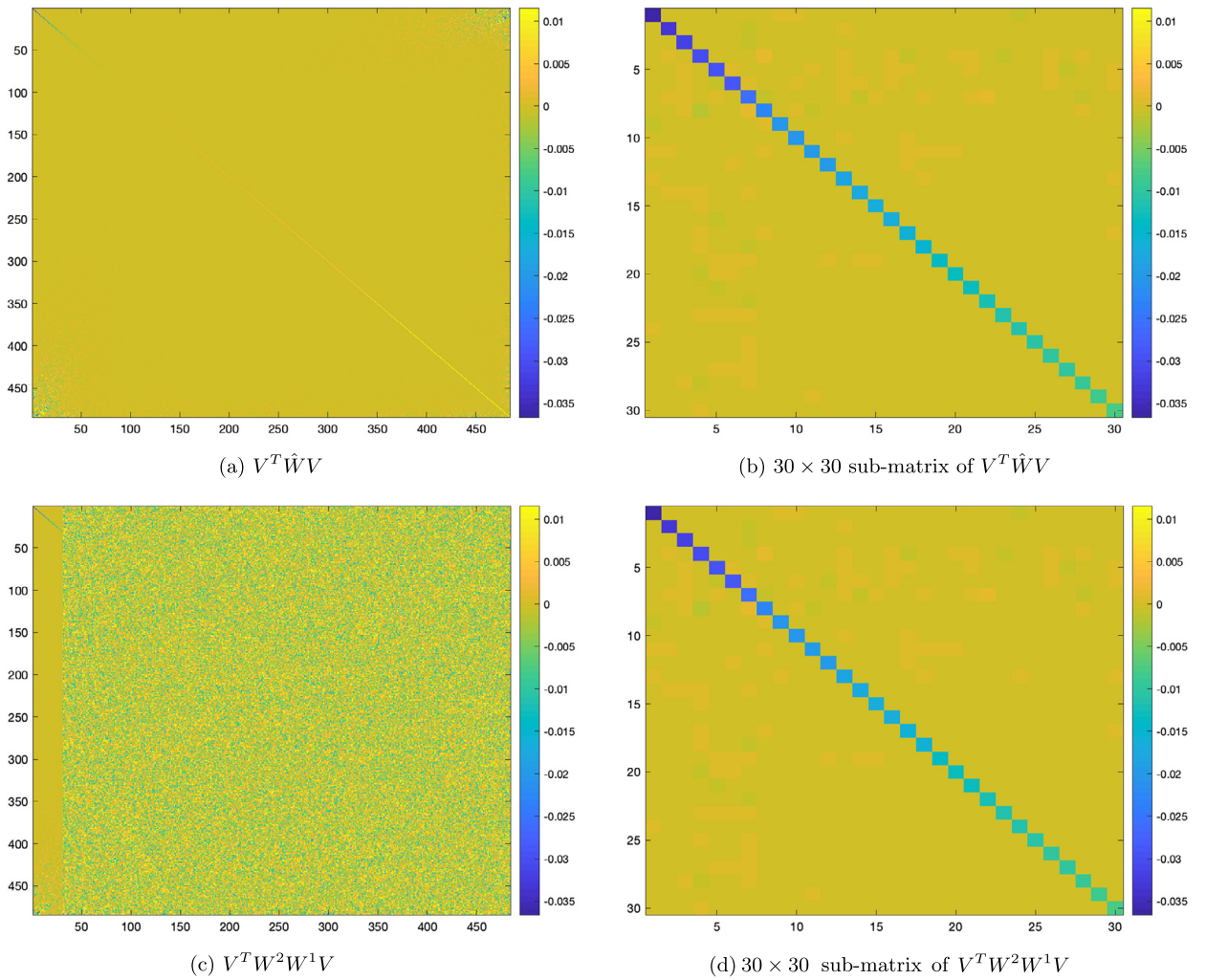
To support this claim, we design a special subspace $S \subset \mathbb{R}^m$. For $r < m$, we then let

$$S = V_r := \text{span}\{v_i, 1 \le i \le r\} \quad \subset \quad \text{span}\{v_i, 1 \le i \le m\} = \mathbb{R}^m, \tag{42}$$

where $\{v_i\}_{i=1}^m$ are eigenvectors of $\hat{W}$ corresponding to eigenvalues λ_i in descending order. We also define matrix V as

$$V := [v_1, v_2, \cdots, v_m]. \tag{43}$$

We then randomly pick training input $U \in V_s$ such that $U = \sum_{i=1}^r c_i v_i$. The $\mathcal{NN}(\cdot)$ is then trained with $(U, \hat{\mathcal{L}}(U))$-like training pairs, and we obtain a corresponding operator $\mathcal{L}(\cdot)$ with trained coefficients. The linear operators of $\mathcal{L}(\cdot)$ and $\hat{\mathcal{L}}(\cdot)$ are compared by their eigenvalues, i.e. $V^T \hat{W} V$ and $V^T W^2 W^1 V$. By the definition of V, the former will exactly be a diagonal matrix with λ_i be its diagonal value. We expect $W^2 W^1$ functions similarly to $\hat{W}$ on V_r, and further the r-by-r sub-matrix of $V^T W^2 W^1 V$ should be similar to that of $V^T \hat{W} V$.



(a) $V^T\hat{W}V$ (b) 30×30 sub-matrix of $V^T\hat{W}V$

(c) $V^TW^2W^1V$ (d) 30×30 sub-matrix of $V^TW^2W^1V$

Fig. 5. $\hat{W}$ and W^2W^1 function similarly on S, where $r = 30$.

Fig. 5 compares $V^TW^2W^1V$ and $V^T\hat{W}V$ for the case when V_r is constructed letting $r = 30$. We can tell that the first 30×30 submatrix are very much alike. That is to say, despite the fact that the operator $\mathcal{L}(\cdot)$ and $\hat{\mathcal{L}}(\cdot)$ are different on $\mathbb{R}^m$, their behavior on the subspace V_r are the same. Moreover, Fig. 6, shows that such similarity only exist in V_r as for the i_{th} diagonal values of $V^TW^2W^1V$ and that of $V^T\hat{W}V$ distinct when $i > s$. This also makes sense as the operator $\mathcal{L}(\cdot)$ is defined from the trained parameters of $\mathcal{NN}(\cdot)$ where only subspace V_r is visible to the network.

5.2. One-layer reduced order neural network

In this example, we consider the one-layer reduced-order neural network as defined in (18). We use this neural network to predict a one-step fluid dynamics, where the data are taken to be NLMC solution coefficients to (1) in the form (U_i^0, U_i^1). We fix $\kappa(t, x)$ and $f(t, x)$ among samples, thus all data describes linear dynamics for different initial conditions. In specific, we use NLMC solutions to different source functions as U_i^0's.

We take 2% out of all data pairs as testing samples and the remaining 98% as training samples and use only the training sample to train $\mathcal{NN}(\cdot)$. We then evaluate the neural network by examining the accuracy of the following approximation for the unseen testing samples:

$$U^1 \approx \mathcal{NN}(U^0).$$

The ℓ_2 relative error of the prediction is computed by

$$\frac{||U^1 - \mathcal{NN}(U^0)||_2}{||U^1||_2}. \tag{44}$$

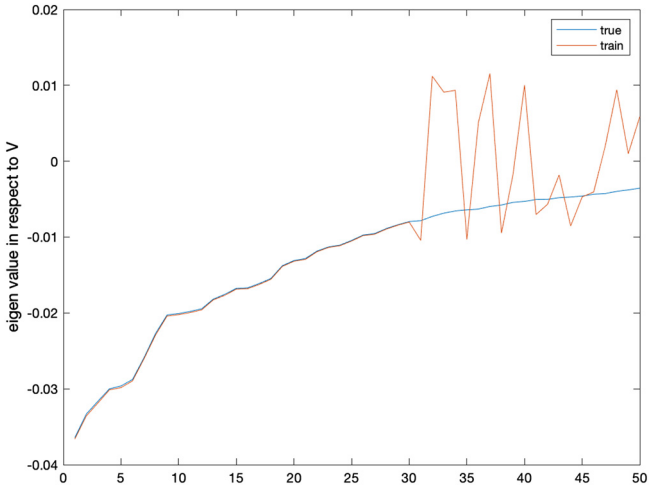


Fig. 6. Comparison of eigen-values of $\hat{W}$ and W^2W^1 when $r = 30$.

Table 1
ℓ_2 relative error of $\mathcal{NN}(\cdot)$ prediction.

Sample index	Error (%)
#1	0.25
#2	0.43
#3	10.02
#4	9.91
#5	3.90
#6	8.18
#7	17.27
#8	1.57
#9	1.13
#10	0.76
Mean	5.34

Table 1 is the error table for the case when we use 500 data pairs with 490 to be training samples and 10 to be testing samples. These data are generated with different choice of initial condition U_i^0. To match the realistic physical situation, we took all initial conditions to be the NLMC terminal pressure of a mobility driven nonlinear flow process.

From Table 1, we can see that the prediction of our proposed network $\mathcal{NN}(U^0)$ is rather effective with an average ℓ_2 error of 5.34%.

We also verify that U^1 is sparse in the learned W^2-system for all data (training and testing). We first reorder the columns of W^2 by their dominance as discussed in Section 4.3, then compute the corresponding W^2-system coefficients U^{1,W^2}, which should be a roughly decreasing vector. From Fig. 7, we can tell that the W^2-system coefficients U^{1,W^2} are sparse. This can be an reflection of successful learning of $\hat{W}^2$ in Assumption 1. Moreover, only a few dominant modes are needed to recover the solution as the quadratic averages of coordinates $|U_j^{1,W^2}|$ decays fast when $j > 100$.

In a word, the proposed neural network can indeed learn the dominant multiscale modes needed to represent U^1 from training data while properly reproduce the map between U^0 and U^1.

5.3. Model reduction with W^2

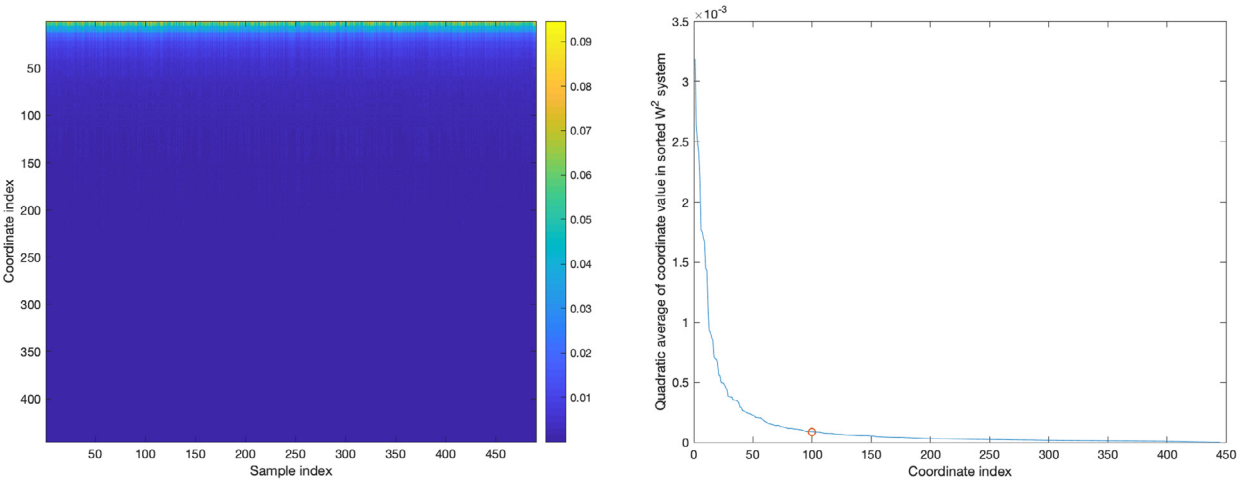
As discussed in Section 4.3, we would like to use the reduced-order system $W^{2,s}$ to further conduct model reduction. The reduced-order solution coefficient is defined with the reduced-order linear operator $\mathcal{L}_s(\cdot)$:

$$U_{sN}^{n+1} = \mathcal{L}_s(U^n). \tag{45}$$

Noticing that U_{sN}^{n+1} is the coefficient of u_h^{n+1} in the original basis system $\{\phi_j\}_{j=1}^m$ while it is sparse in W^2-system, i.e., U_{sN}^{n+1,W^2} is sparse and has maximum s nonzero elements.

The numerical experiments are conduced based on a one-layer neural network as defined in (18) for a one-step linear dynamics. We would like to compare the following coefficient vectors:

$$U_{sN}^1 := \mathcal{L}_s(U^0) = W^{2,s}W^1 \cdot U^0 + W^{2,s}b,$$



(a) U^{1,W^2} for all data samples (b) quadratic average of U^{1,W^2} among all data samples

Fig. 7. Sparsity of the output of $\mathcal{NN}(U^0)$ in W^2.

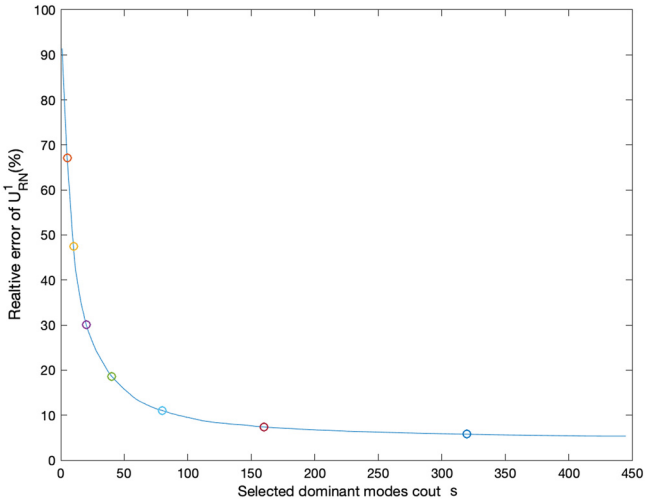


Fig. 8. Decay of relative error $\frac{||U^1_{sN}-U^1_{\text{true}}||_2}{||U^1_{\text{true}}||_2}$ as s grows.

$$U^1_{\text{true}} = \hat{W}U^0 + \hat{b},$$

and

$$U^1_N := \mathcal{NN}(U^0).$$

Here U^1_{true} is the true solution to (20), while U^1_N is the prediction of $\mathcal{NN}(\cdot)$.

Fig. 8 shows the error decay of U^1_{sN} compared to U^1_{true}. As s grows, the error gets smaller. This figure actually verified (41), i.e. the reduced operator $\mathcal{L}_s(\cdot)$ can approximate $\hat{\mathcal{L}}(\cdot)$. Moreover, this approximation gets more accurate as s gets larger. The error in Fig. 8 at $s = m = 445$ is also expected that it is a consequence of the training error. Besides, we observe that the error decays fast when $s > 40$ for our training samples.

Table 2 further facilitates such conclusion. The error of U^1_{sN} is less than 12% when $s \geq 80$. We can thus represent the multiscale solution u^1_h using $s = 80$ basis with little sacrifice in the solution accuracy. We notice that the order of the reduce operator $\mathcal{L}_s(\cdot)$ is only around 18% that of the original multiscale model.

We lastly present the comparison between U^1_{true}, U^1_N, and U^1_{sN}. From Table 3, we can tell that for a single testing sample, we have

$$\frac{||U^1_{sN}-U^1_{\text{true}}||_2}{||U^1_{\text{true}}||_2} > \frac{||U^1_N-U^1_{\text{true}}||_2}{||U^1_{\text{true}}||_2}, \quad \frac{||U^1_{sN}-U^1_{\text{true}}||_2}{||U^1_{\text{true}}||_2} > \frac{||U^1_{sN}-U^1_N||_2}{||U^1_N||_2},$$

Table 2
Decay of error $\frac{||U^1_{sN}-U^1_{\text{true}}||_2}{||U^1_{\text{true}}||_2}$ with respect to number of selected dominant modes in $W^{2,s}$.

Sample index	s						
	5 (1.12%)	10 (2.24%)	20 (4.48%)	40 (8.96%)	80 (17.92%)	160 (35.84%)	320 (71.68%)
#1	66.71	46.80	29.02	17.01	8.44	3.53	1.11
#2	66.55	46.58	29.03	17.41	9.48	4.62	1.59
#3	66.35	46.77	29.70	18.59	12.26	10.92	10.16
#4	67.95	48.58	31.23	19.87	12.59	10.03	9.90
#5	66.22	46.31	28.99	17.73	10.49	6.49	4.34
#6	66.47	46.76	29.40	17.89	10.86	9.12	8.33
#7	69.59	51.33	36.01	27.42	22.33	18.62	17.29
#8	66.60	46.61	28.81	16.39	7.47	3.86	1.96
#9	66.90	47.13	29.21	16.94	8.02	3.58	1.67
#10	67.14	47.39	29.44	17.23	8.23	3.19	1.32
Mean	67.05	47.43	30.08	18.65	11.02	7.40	5.77

Table 3
Relative error percentage of solutions obtained in full W^2 system and reduced-order system $W^{2,s}$ for $s = 100$.

Sample index	$\frac{||U^1_N-U^1_{\text{true}}||_2}{||U^1_{\text{true}}||_2}$	$\frac{||U^1_{sN}-U^1_{\text{true}}||_2}{||U^1_{\text{true}}||_2}$	$\frac{||U^1_{sN}-U^1_N||_2}{||U^1_N||_2}$
#1	0.25	6.48	6.41
#2	0.43	7.65	7.55
#3	10.02	11.59	5.81
#4	9.91	11.28	5.83
#5	3.90	8.93	8.38
#6	8.18	10.00	5.80
#7	17.27	20.98	10.25
#8	1.57	5.85	5.83
#9	1.13	6.13	6.03
#10	0.76	6.17	6.12
Mean	5.34	9.50	6.80

which is as expected since $\frac{||U^1_N-U^1_{\text{true}}||_2}{||U^1_{\text{true}}||_2}$ and $\frac{||U^1_{sN}-U^1_N||_2}{||U^1_N||_2}$ are exactly the two components of error $\frac{||U^1_{sN}-U^1_{\text{true}}||_2}{||U^1_{\text{true}}||_2}$. They stand for neural network prediction error and $\mathcal{L}_s$ truncation error, respectively. The latter can be reduced by increasing s, while the former one can only be improved with more effective training.

5.4. Multi-layer reduced order Neural Network

In this example, we use a multi-layer reduced-order neural network $\mathcal{NN}(\cdot)$ to predict multi-step fluid dynamics. Recall (15), it is defined as

$$\mathcal{NN}(\mathbf{x}^0) := \mathcal{N}^n(\cdots \mathcal{N}^1(\mathcal{N}^0(\mathbf{x}^0))).$$

The input of $\mathcal{NN}(\cdot)$ is taken to be U^0, the initial condition, while the outputs are the collection of outputs at n_{th}-layer sub-network $\mathcal{N}^n(\cdot)$ which correspond to the true values $[U^1, U^2, \cdots, U^9]$. U^{n+1} are all taken to be NLMC solutions of (1) at time step n for $n = 0, \cdots, 8$. Prediction accuracy is measured with ℓ_2 relative error that defined similar to (44).

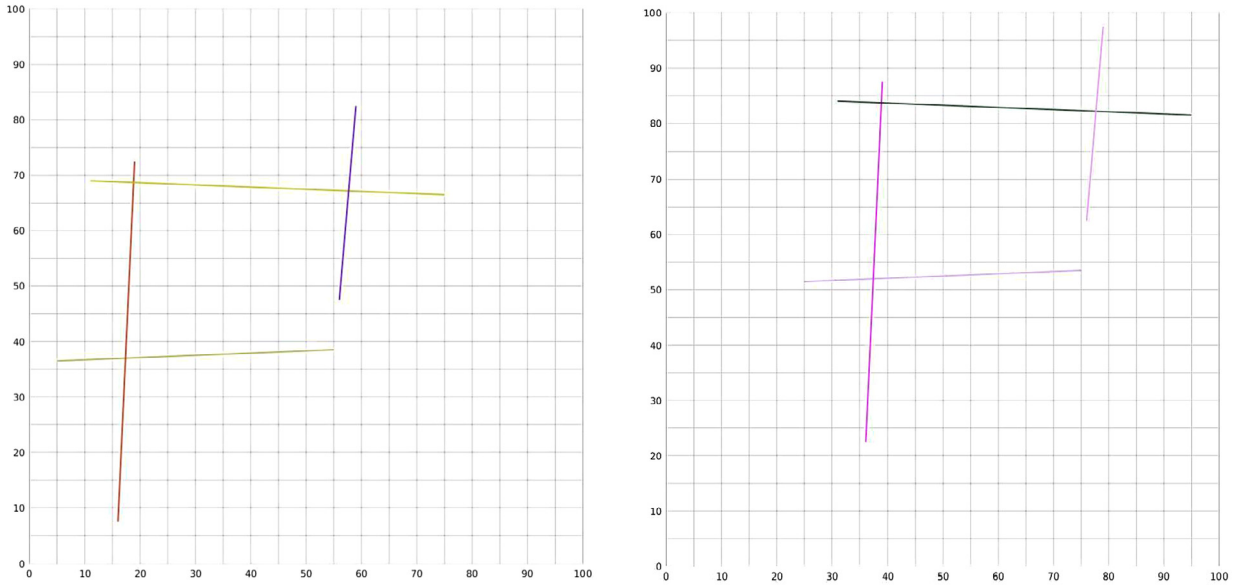
In Table 4, the columns shows the prediction error for $U^{n+1}, n = 0, 1, \cdots 8$ where the average error is computed for all time steps among testing samples which are less than 10% in average. Therefore, we claim that the proposed multi-layer reduced order neural network $\mathcal{NN}(\cdot)$ is effective in the aspect of prediction. We also claim that the coefficient U^{n+1,W^2_n} for $n = 0, 1, \cdots, 8$ are sparse in the independent systems W^2_n. These systems are again learned simultaneously by training $\mathcal{NN}(\cdot)$.

5.5. Clustering

In this experiment, we aim to model the fluid dynamics correspond to two different fractured media as shown in Fig. 9. More specifically, the permeability coefficient of matrix region have $\kappa_m = 1$ and the permeability of the fractures are $\kappa_f = 10^3$. The one-step NLMC solutions pairs (U^0, U^1) are generated following these two different configurations of fracture are then referred as "Cluster 1" and "Cluster 2" (see Fig. 10 for an illustration). We will then compare the one-step prediction of networks $\mathcal{NN}_1, \mathcal{NN}_2$ with that of $\mathcal{NN}_{\text{mixed}}$. The input are taken to be U^0, which are chosen to be the terminal solution of mobility driven 10-step nonlinear dynamics, while the output is an approximation of U^1.

Table 4
ℓ_2 relative error of prediction of U^{n+1} using $\mathcal{NN}(\cdot)$.

Sample index	U^1	U^2	U^3	U^4	U^5	U^6	U^7	U^8	U^9
#1	1.62	1.17	1.69	1.91	1.95	1.92	1.92	1.91	1.96
#2	3.33	1.86	2.10	1.98	2.15	1.53	1.04	0.94	0.77
#3	11.62	13.32	9.39	9.57	8.89	10.36	11.67	11.86	12.97
#4	9.74	9.00	4.21	3.45	3.46	3.17	2.93	2.87	2.81
#5	5.63	4.68	2.65	2.28	2.52	1.89	1.49	1.74	1.91
#6	9.54	11.50	9.30	9.70	9.14	10.50	11.73	12.00	13.09
#7	21.46	14.82	5.42	4.24	3.27	5.06	6.52	6.77	7.81
#8	5.72	1.40	0.67	0.77	1.11	1.49	1.95	2.54	3.20
#9	4.03	2.16	3.21	3.66	3.47	4.36	5.15	5.35	5.97
#10	4.62	1.01	3.14	3.84	3.81	4.47	5.05	5.24	5.68
Mean	7.73	6.09	4.18	4.14	3.98	4.47	4.95	5.12	5.62



(a) Cluster 1 (b) Cluster 2

Fig. 9. Fracture networks for two clusters.

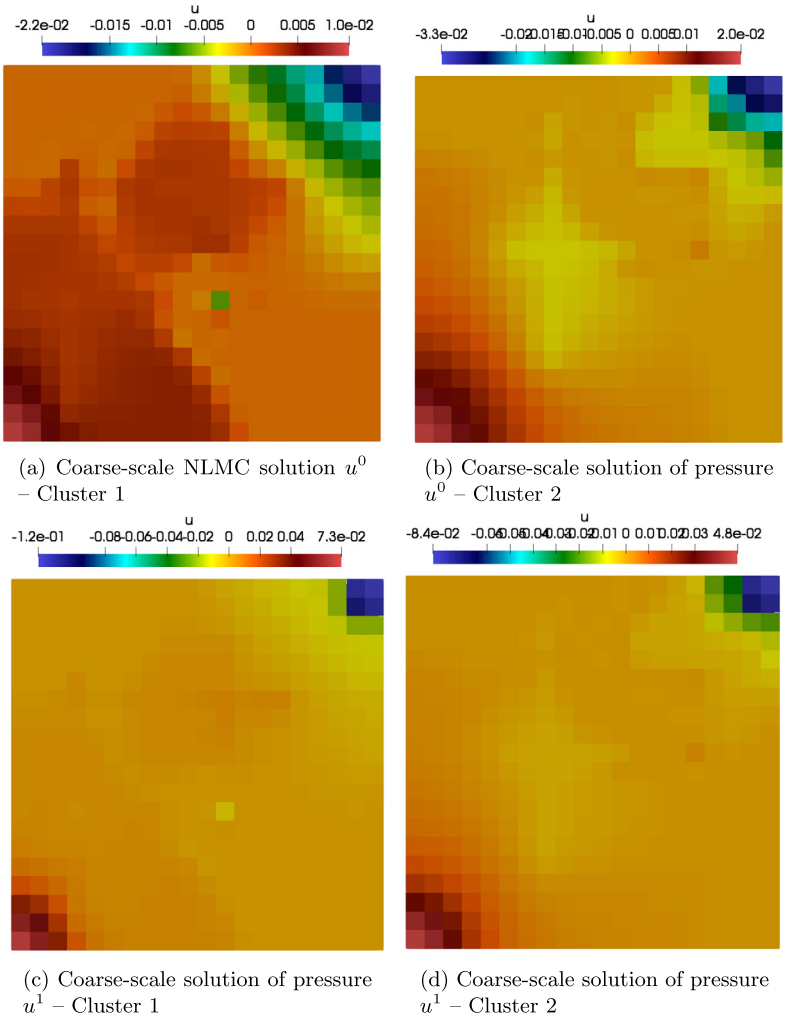
$\mathcal{NN}_1, \mathcal{NN}_2$ and $\mathcal{NN}_{\text{mixed}}$ share same one-layer soft thresholding neural network structure as in (18) while the first two network are trained with data for each cluster separately and the latter one is trained with mixed data for two clusters.

Fig. 9 shows the fracture network we use to generate data for two clusters, while Fig. 10 shows an example of solution U^0 and U^1 for each cluster. As observed from the profiles of U^0 and U^1, we can see that these the solutions are very different due to the translation of the fractures. Moreover, since the data resulted from both clusters have non-uniform map between U^0 and U^1, the mixed data set can be considered as obtained from a nonlinear map.

Table 5 demonstrates the comparison of the prediction accuracy when the network is fed with a single cluster data and when the network is fed with a mixed data. This simple treatment can significantly improve the accuracy. For Cluster 1, the average prediction error of $\mathcal{NN}_1$ is around 7.45% while that of the $\mathcal{NN}_{\text{mixed}}$ is around 36.64%. Similar contrast can also be observed for Cluster 2.

6. Conclusion

In this paper, we discuss a novel deep neural network approach to model reduction approach for multiscale problems in porous media. To numerically solve multiscale problems, a fine grid needs to be used and results in a huge number of degrees of freedom. To this end, non-local multicontinuum (NLMC) upscaling [18] is used as a dimensionality reduction technique. In porous media flow dynamics problems, multiscale solutions at consecutive time instants are regarded as an input-output mechanism and learnt from deep neural networks techniques. By exploiting a relation between a soft-thresholding neural network and ℓ_1 minimization, multiscale features of the coarse-grid solutions are extracted using neural networks. This results in a new neural-network-based construction of reduced-order model, which involves extracting appropriate important modes at each time step. We also suggest an efficient strategy for a class of nonlinear flow problems.



(a) Coarse-scale NLMC solution u^0 – Cluster 1

(b) Coarse-scale solution of pressure u^0 – Cluster 2

(c) Coarse-scale solution of pressure u^1 – Cluster 1

(d) Coarse-scale solution of pressure u^1 – Cluster 2

Fig. 10. Example solution pairs for two different clusters.

Table 5
Prediction error of $\mathcal{N}_1$, $\mathcal{N}_2$ and $\mathcal{N}_{\text{mixed}}$.

(a) Relative ℓ_2 prediction error(%) of $\mathcal{N}_1$ and $\mathcal{N}_2$

Sample index	Cluster 1 error	Cluster 2 error
#1	0.87	0.26
#2	3.19	0.45
#3	13.62	0.79
#4	12.36	0.49
#5	7.64	0.43
#6	10.69	1.00
#7	19.08	0.53
#8	2.57	0.92
#9	0.77	0.46
#10	3.70	0.38
Mean	7.45	0.57

(b) Relative ℓ_2 prediction error(%) of $\mathcal{N}_{\text{mixed}}$

Sample index	Cluster 1 error	Cluster 2 error
#1	36.25	84.83
#2	35.08	84.20
#3	28.20	111.05
#4	44.60	69.35
#5	32.32	89.30
#6	30.03	106.41
#7	48.61	57.76
#8	35.63	88.46
#9	36.89	88.67
#10	38.78	83.85
Mean	36.64	86.39

Finally, we present numerical examples to demonstrate the performance of our method applied to some selected problems. In the future, we plan to explore new design of convolutional neural networks associated with each continua and interactions between continua.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

The research of Eric Chung is partially supported by the Hong Kong RGC General Research Fund (Project 14317516) and the CUHK Direct Grant for Research 2016-17. YE would like to thank the partial support from National Science Foundation 1620318. YE would also like to acknowledge the support of Mega-grant of the Russian Federation Government (N 14.Y26.31.0013).

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, Xiaoqiang Zheng, TensorFlow: Large-scale machine learning on heterogeneous systems, 2015, Software available from tensorflow.org.
[2] Assyr Abdulle, Yun Bai, Adaptive reduced basis finite element heterogeneous multiscale method, Comput. Methods Appl. Mech. Eng. 257 (2013) 203–220.
[3] G. Allaire, R. Brizzi, A multiscale finite element method for numerical homogenization, SIAM J. Multiscale Model. Simul. 4 (3) (2005) 790–812.
[4] Manal Alotaibi, Victor M. Calo, Yalchin Efendiev, Juan Galvis, Mehdi Ghommem, Global–local nonlinear model reduction for flows in heterogeneous porous media, Comput. Methods Appl. Mech. Eng. 292 (2015) 122–137.
[5] T. Arbogast, Implementation of a locally conservative numerical subgrid upscaling scheme for two-phase Darcy flow, Comput. Geosci. 6 (2002) 453–481.
[6] Amir Beck, Marc Teboulle, A fast iterative shrinkage-thresholding algorithm for linear inverse problems, SIAM J. Imaging Sci. 2 (1) (2009) 183–202.
[7] I. Bilionis, N. Zabaras, Solution of inverse problems with limited forward solver evaluations: a Bayesian perspective, Inverse Probl. 30 (015004) (2013).
[8] Ilias Bilionis, Nicholas Zabaras, Bledar A. Konomi, Guang Lin, Multi-output separable Gaussian process: towards an efficient, fully Bayesian paradigm for uncertainty quantification, J. Comput. Phys. 241 (2013) 212–239.
[9] Hervé Bourlard, Yves Kamp, Auto-association by multilayer perceptrons and singular value decomposition, Biol. Cybern. 59 (4–5) (1988) 291–294.
[10] Donald L. Brown, Daniel Peterseim, A multiscale method for porous microstructures, arXiv preprint arXiv:1411.1944, 2014.
[11] E. Chung, Y. Efendiev, S. Fu, Generalized multiscale finite element method for elasticity equations, Int. J. Geomath. 5 (2) (2014) 225–254.
[12] E. Chung, Y. Efendiev, W.T. Leung, Generalized multiscale finite element method for wave propagation in heterogeneous media, SIAM Multicale Model. Simul. 12 (2014) 1691–1721.
[13] E. Chung, W.T. Leung, A sub-grid structure enhanced discontinuous Galerkin method for multiscale diffusion and convection-diffusion problems, Commun. Comput. Phys. 14 (2013) 370–392.
[14] E.T. Chung, Y. Efendiev, W.T. Leung, M. Vasilyeva, Y. Wang, Online adaptive local multiscale model reduction for heterogeneous problems in perforated domains, Appl. Anal. 96 (12) (2017) 2002–2031.
[15] E.T. Chung, Y. Efendiev, G. Li, An adaptive GMsFEM for high contrast flow problems, J. Comput. Phys. 273 (2014) 54–76.
[16] Eric Chung, Yalchin Efendiev, Thomas Y. Hou, Adaptive multiscale model reduction with generalized multiscale finite element methods, J. Comput. Phys. 320 (2016) 69–95.
[17] Eric Chung, Maria Vasilyeva, Yating Wang, A conservative local multiscale model reduction technique for Stokes flows in heterogeneous perforated domains, J. Comput. Appl. Math. 321 (2017) 389–405.
[18] Eric T. Chung Efendiev, Wing Tat Leung, Maria Vasilyeva, Yating Wang, Non-local multi-continua upscaling for flows in heterogeneous fractured media, arXiv preprint arXiv:1708.08379, 2018.
[19] Eric T. Chung, Yalchin Efendiev, Wing Tat Leung, Maria Vasilyeva, Yating Wang, Non-local multi-continua upscaling for flows in heterogeneous fractured media, J. Comput. Phys. (2018).
[20] Martin Drohmann, Bernard Haasdonk, Mario Ohlberger, Reduced basis approximation for nonlinear parametrized evolution equations based on empirical operator interpolation, SIAM J. Sci. Comput. 34 (2) (2012) A937–A969.
[21] W. E, B. Engquist, Heterogeneous multiscale methods, Commun. Math. Sci. 1 (1) (2003) 87–132.
[22] Y. Efendiev, J. Galvis, E. Gildin, Local-global multiscale model reduction for flows in highly heterogeneous media, J. Comput. Phys. 231 (2012) 8100–8113.
[23] Y. Efendiev, J. Galvis, T.Y. Hou, Generalized multiscale finite element methods (GMsFEM), J. Comput. Phys. 251 (2013) 116–135.
[24] Y. Efendiev, J. Galvis, X.H. Wu, Multiscale finite element methods for high-contrast problems using local spectral basis functions, J. Comput. Phys. 230 (2011) 937–955.
[25] Jacob Fish, Wen Chen, Space–time multiscale model for wave propagation in heterogeneous media, Comput. Methods Appl. Mech. Eng. 193 (45) (2004) 4837–4856.
[26] Jacob Fish, Rong Fan, Mathematical homogenization of nonperiodic heterogeneous media subjected to large deformation transient loading, Int. J. Numer. Methods Eng. 76 (7) (2008) 1044–1064.
[27] M. Ghommem, M. Presho, V.M. Calo, Y. Efendiev, Mode decomposition methods for flows in high-contrast porous media. Global–local approach, J. Comput. Phys. 253 (2013) 226–238.
[28] Patrick Henning, Mario Ohlberger, The heterogeneous multiscale finite element method for elliptic homogenization problems in perforated domains, Numer. Math. 113 (4) (2009) 601–629.
[29] Diederik P. Kingma, Jimmy Ba Adam, A method for stochastic optimization, arXiv preprint, arXiv:1412.6980, 2014.
[30] X. Ma, M. Al-Harbi, A. Datta-Gupta, Y. Efendiev, A multistage sampling approach to quantifying uncertainty during history matching geological models, SPE J. 13 (10) (2008) 77–87.
[31] Ana-Maria Matache, Christoph Schwab, Two-scale fem for homogenization problems, ESAIM: Math. Model. Numer. Anal. 36 (04) (2002) 537–572.
[32] A. Mondal, Y. Efendiev, B. Mallick, A. Datta-Gupta, Bayesian uncertainty quantification for flows in heterogeneous porous media using reversible jump Markov Chain Monte-Carlo methods, Adv. Water Resour. 33 (3) (2010) 241–256.
[33] H. Owhadi, L. Zhang, Metric-based upscaling, Commun. Pure Appl. Math. 60 (2007) 675–723.
[34] Jürgen Schmidhuber, Deep learning in neural networks: an overview, Neural Netw. 61 (2015) 85–117.

[35] Yating Wang, Siu Wun Cheung, Eric T. Chung, Yalchin Efendiev, Min Wang, Deep multiscale model learning, arXiv preprint, arXiv:1806.04830, 2018.
[36] Yanfang Yang, Mohammadreza Ghasemi, Eduardo Gildin, Yalchin Efendiev, Victor Calo, et al., Fast multiscale reservoir simulations with pod-deim model reduction, SPE J. 21 (06) (2016) 2–141.
[37] Yinhao Zhu, Nicholas Zabaras, Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification, J. Comput. Phys. 366 (2018) 415–447.