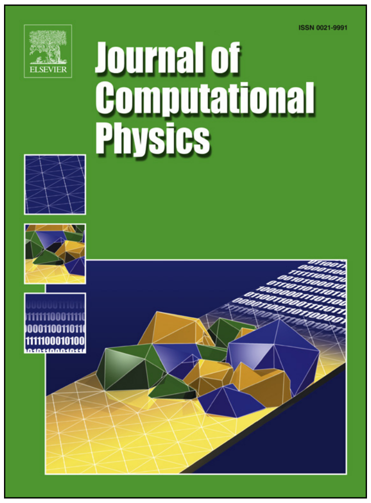


Deep multiscale model learning

Yating Wang, Siu Wun Cheung, Eric T. Chung, Yalchin Efendiev, Min Wang

PII: S0021-9991(19)30776-4
DOI: https://doi.org/10.1016/j.jcp.2019.109071
Reference: YJCPH 109071

To appear in: *Journal of Computational Physics*

Received date: 15 June 2018
Revised date: 15 August 2019
Accepted date: 27 October 2019

Highlights

- Combine multiscale model reduction and deep learning.
- Use sufficient coarse simulation data and limited fine observed data in training.
- Derive surrogate coarse-grid models which take into account observed data.
- The multiscale concepts provide appropriate information for the design of DNN.
- Incorporate fine observation data can improve the coarse grid model.

Deep Multiscale Model Learning

Yating Wang[*] Siu Wun Cheung[†] Eric T. Chung[‡] Yalchin Efendiev[§]
Min Wang[¶]

November 20, 2019

Abstract

The objective of this paper is to design novel multi-layer neural networks for multiscale simulations of flows taking into account the observed fine data and physical modeling concepts. Our approaches use deep learning techniques combined with local multiscale model reduction methodologies to predict flow dynamics. Using reduced-order model concepts is important for constructing robust deep learning architectures since the reduced-order models provide fewer degrees of freedom. We consider flow dynamics in porous media as multi-layer networks in this work. More precisely, the solution (e.g., pressures and saturation) at the time instant $n+1$ depends on the solution at the time instant n and input parameters, such as permeability fields, forcing terms, and initial conditions. One can regard the solution as a multi-layer network, where each layer, in general, is a nonlinear forward map and the number of layers relates to the internal time steps. We will rely on rigorous model reduction concepts to define unknowns and connections between layers. It is critical to use reduced-order models for this purpose, which will identify the regions of influence and the appropriate number of variables. Furthermore, due to the lack of available observed fine data, the reduced-order model can provide us sufficient inexpensive data as needed. The designed deep neural network will be trained using both coarse simulation data which is obtained from the reduced-order model and observed fine data. We will present the main ingredients of our approach and numerical examples. Numerical results show that using deep learning with data generated from multiscale models as well as available observed fine data, we can obtain an improved forward map which can better approximate the fine scale model.

1 Introduction

Many processes have multiple scales and uncertainties at the finest scales. These include, for example, porous media processes, where the media properties can vary over many scales. Constructing models on a computational coarse grid is challenging. Many multiscale methods [24, 23, 39, 34, 22] and solvers are designed to construct coarse spaces and resolve unresolved scales to a desired accuracy via additional computing. In general, for nonlinear problems and in the presence of observed solution-related data, multiscale models are challenging to construct [25, 8, 3]. However, the idea of multiscale methods can give a guidance to construct robust computational models by combining multiscale concepts with deep learning methodologies. This is an objective of this paper.

In this work, we consider multiscale methods for nonlinear PDEs and incorporate the observed fine data to modify thecoarse-grid model. This is a typical situation in many applications, where multiscale methods are often used to guide coarse-grid models. These approximations, e.g., typically involve a form of the coarse-grid equations [2, 24, 4, 23, 14, 7, 10, 22, 1, 21, 26, 27, 43, 41, 31, 13, 15, 11, 12], where the coarse-grid

[*]Department of Mathematics, Texas A&M University, College Station, TX 77843, USA (`wytgloria@math.tamu.edu`)
[†]Department of Mathematics, Texas A&M University, College Station, TX 77843, USA (`tonycsw2905@math.tamu.edu`)
[‡]Department of Mathematics, The Chinese University of Hong Kong, Shatin, New Territories, Hong Kong SAR, China (`tschung@math.cuhk.edu.hk`)
[§]Department of Mathematics & Institute for Scientific Computation (ISC), Texas A&M University, College Station, Texas, USA & Multiscale Model Reduction Laboratory, North-Eastern Federal University, Yakutsk, Russia, 677980 (`efendiev@math.tamu.edu`)
[¶]Department of Mathematics, Texas A&M University, College Station, TX 77843, USA (`wangmin@math.tamu.edu`)

equations are formed and the parameters are computed or found via inverse problems [5, 40, 6, 42, 50]. As was shown in [18, 17], the form of upscaled and multiscale equations can be complicated, even for linear problems. To condition these models to the limited observed fine data, we propose a multi-layer neural network, which uses multiscale concepts as well as available data.

In this work, we will use the non-local multi-continuum approach (NLMC), developed in [17, 18]. This approach identifies the coarse-grid quantities in each cell and their connectivity to neighboring variables. The approach derives its foundation from the Constraint Energy Minimizing Generalized Multiscale Finite Element Method (CEM-GMsFEM) [16]. It has a convergence rate H/Λ, where H denotes the coarse grid size, and Λ is the minimal eigenvalue that the corresponding eigenvector is not included in the local multiscale space.Using the concept of CEM-GMsFEM, NLMC defines new basis functions such that the degrees of freedom have physical meanings. This makes the coarse grid solutions obtained from NLMC represent the solution averages in coarse degrees of freedom. One do not necessarily need to downscale the coarse scale quantities solved from the coarse equation onto the fine grid. Therefore, NLMC will be chosen as our multiscale method.

Deep learning has attracted a lot of attention in a wide class of applications and gained great success in many computer vision tasks including image recognition, language translation and so on [37, 32, 30]. Deep Neural Network is one particular branch of artificial neural network algorithm under the concept of machine learning. Neural networks are information processing systems inspired by the biological nervous systems and animal brains. In an artificial neural network, there are a collection of connected units called artificial neurons, which are analogous to axons in the brain of an animal or human. Each neuron can transmit a signal to another neuron through the connections. The receiving neuron will then process the signal and transmit the signal to downstream neurons, etc. Although it is unclear how learning in biological systems occurs, in modern neural network systems, the backpropagation algorithms combined with gradient methods make it possible to update the weights of network by calculating the gradient of the loss function, and thus train the model. Techniques like max pooling, stochastic descent further make the training of deep neural networks surprisingly effective.More network structures like convolutional neural network (CNN) and recurrent neural networks (RNN) are designed targeting various tasks. Many researches have focused on learning the expressivity of deep neural nets theoretically [20, 33, 19, 47, 44, 29]. For example, there are numerous results to investigate the universal approximation property of neural networks and show the ability of deep networks in approximations of a rich class of functions. The structure of a deep neural network is usually a composition of multiple layers, with several neurons in each layer. In deep learning, the first layer learns primitive features, these features are then fed to the next layer, which trains itself to recognize more complex features. As layers stack up, this process is repeated in successive layers until the system can reliably recognize phonemes or objects. In between layers, some activation functions are needed, which can be a nonlinear transformation from an input neuron to an output neuron. That output is then used as an input in the next layer in the network. The composition structure of the deep nets is important for approximating complicated functions. This encourages many works utilizing deep learning in solving partial differential equations and model reductions. For example, in the work [49] the authors numerically solve Poisson problems and eigenvalue problems in the context of the Ritz method based on representing the trial functions by deep neural networks. In [35], a neural network was proposed to learn the physical quantity of interest as a function of random input coefficients; the accuracy and efficiency of the approach for solving parametric PDE problems was shown. In the work [50], the authors study deep convolution networks for surrogate models. In [38], the authors build a connection between residual networks (ResNet) and the characteristic equation of the transport equation. This work proposes a continuous flow model for ResNet and shows an alternative perspective to understand deep neural networks. Other new developments on the approximation of PDE include extensive work on physics informed neural networks (PINN) [45, 46]. PINN is successfully applied to solve both forward and inverse problems constrained to respect the law of physics that govern the observed data. The trained neural networks can be used as physics-informed surrogate models in the observed data-sufficient case. We remark that, in our approach, reduced-order model is used to efficiently generate more training data, which helps to overcome the high cost of observed fine data acquisition. Moreover, it guides us to identify the regions of influence and the appropriate number of variables in the training.

In this work, we will bring together machine learning and novel multiscale model reduction techniques to design/modify upscaled models and to train coarse-grid discrete systems. This will also allow alleviating some of the computational complexity involved in multiscale methods for time-dependent nonlinear problems. Nonlinear time-dependent PDEs will be treated as multi-layer networks. More precisely, the solution at the time instant $n+1$ depends on the solution at the time instant n and input parameters, such as permeability fields and source terms. One can regard the solution as the output of a multi-layer network. We will rely on rigorous multiscale concepts, for example from [17], to define unknowns and regions of influence (oversampling neighborhood structure). In each layer, our reduced-order models will provide an initialization of the architecture for the surrogate forward map, which will then be modified using available data. It is critical to use reduced-order models for this purpose, which will identify the regions of influence and the appropriate number of variables.

Because of the lack of available data in porous media applications, the training will be supplemented with coarse simulation data as needed, which will result in data based modified multiscale models. There are various sources for observed data, for example, they can be selected from different permeability fields (or can be taken as different multi-phase models), or they can be obtained from the solutions of the flow problem on sufficient fine grid. We will investigate the observed fine data-rich and data-deficient models. Multi-scale hierarchical structure of porous media will be employed to construct neural networks that can approximate the forward map in the governing non-linear equations.

In our numerical example, we will consider a model problem, a diffusion equation, and measure the solution at different time steps. The neural network is constructed using information from an upscaled model based on the non-local multi-continuum approach [17]. That is, we select apriori numbers of coarse-grid variables in our simulations (based on the possible number of fractures) and impose a constraint on the connection between different layers of neurons to indicate the region of influence. Because of coarseness of the model, the predictions are more computationally inexpensive. We have observed that the network identifies multiscale features of the solution.

During the deep learning process, we train the network using observed fine data and coarse simulation data. We compute the mean errors across different samples and observe that if only the coarse simulation data is used in the training, the error can be larger compared to if we combined fine observed data in the training. Our approach indicates that incorporating some observed fine data in the training can improve the network which approximate the coarse grid model. The resulting deep neural network provides a modified forward map, which can be treated as a new coarse-grid model that is closer to the fine model or fits the fine observed data better.

We also remark that using plenty of coarse simulation data together with the existing observed fine data in the training can leverage the burden of expensive data acquisition. Deep learning algorithms have also been tested for training elements of the stiffness matrix and multiscale basis functions for channelized systems [48]. Our initial numerical results show that one can achieve a high accuracy using multi-layer networks in predicting the discrete coarse-grid systems.

The paper is organized as follows. In the next section, Section 2, we present general multiscale concepts. Section 3 is dedicated to neural network construction. In Section 4, we present numerical results.

2 Preliminaries

In general, we study

$$u_t = F(x, t, u, \nabla u, I) \tag{1}$$

where I denotes the input, which can include the media properties, such as permeability field, source terms (well rates), or initial conditions. F can have a multiscale dependence with respect to space and time. The coarse-grid equation for (1) can have a complicated form for many problems (cf. [17, 18]). This involves multiple coarse-grid variables in each computational coarse grid, non-local connections between the coarse-grid variables, and complex nonlinear local problems with constraints. In a formal way, the coarse-grid equations in the time interval $[t_n, t_{n+1}]$ can be written for $u_i^{j,n}$, where i is the coarse-grid block, j is a continuum representing the coarse-grid variables, and n is the time step.

For each coarse-grid block K_i, one may need several coarse-grid variables, which will be denoted by indices j.Let $\overline{u}_i^{j,n} = \int_{K_i^j} u^n$ be the average of fine scale solution in the j-th continuum of coarse region K_i.

The equation for $\overline{u}_i^{j,n}$, in general, has a form

$$\frac{\overline{u}_i^{j,n+1}-\overline{u}_i^{j,n}}{\triangle t}=\sum_{i,j}\overline{F}_{i,j}(x,t,\overline{u}_i^{j,n+1},\nabla\overline{u}_i^{j,n+1},I), \tag{2}$$

where $\overline{u}_i^{j,n}$ is the coarse grid variables defined by prescribing averages of the solution at time t^n, for the j_{th} continuum within i_{th} coarse block, and the sum is taken over some neighborhood cells and corresponding connectivity continuum. The computation of $\overline{F}$ can be expensive and involve local nonlinear problems with constraints. In many cases, researchers use general concepts from upscaling, for example, the number of continua, the dependence of $\overline{F}$, non-locality, to construct multiscale models. We propose to use the overall concept of the complex upscaled models in conjunction with deep learning strategies to design novel data-aware coarse-grid models.

Denote coarse-grid solutions by $\{u_i^{j,n}\}$, which satisfy the upscaled model. Let $\mathcal{G}$ be a nonlinear map such that the downscaled solution u_h^n can be thought as a nonlinear interpolation of $u_i^{j,n}$, i.e.

$$u_h^n=\mathcal{G}(\{u_i^{j,n}\}).$$

Once $\mathcal{G}$ is computed by solving nonlinear local problems with constraint, then one can seek all coarse-grid variables $\{u_i^{j,n}\}$ such that the downscaled solution u_h^n satisfies the global variational problem.

Due to the choices of observed fine data and coarse simulation data in this paper, we use $u_s=\{u_i^{j,n}\}$, and $u_o=\{\overline{u}_i^{j,n}\}$ later in the paper.

Next, let's take a look at a specific equation. In the paper, we consider a special case of (1), the diffusion equation in fractured media

$$\frac{\partial u}{\partial t}-\text{div}(\kappa(x)\lambda(t,x)\nabla u)=g(t),\quad \text{in}\quad D, \tag{3}$$

subject to some boundary conditions. Our numerical examples consider the zero Neumann boundary condition $\nabla u\cdot n=0$. Here, D is the computational domain, u is the pressure of flow, $g(t)$ is a time dependent source term, and $\kappa(x)$ is a fixed heterogeneous fractured permeability field. The $\lambda(t,x)$ is some given mobility which is time dependent and represent the nonlinearities in two-phase flow. Our approach can be applied to nonlinear equations. As the input parameter I, we will consider source terms $g(t,I)$, which correspond to well rates. In general, we can also consider permeability fields as well as initial conditions as the input parameter.

2.1 Multiscale model: Non-local multi-continuum approach

In this section, we describe in more details nonlocal multi-continuum approach following [17]. In our work, we consider the diffusion problem in fractured media, and divide the domain D into the matrix region and the fractures, where the matrix has low conductivity and the fractures are low dimensional objects with high conductivity. That is

$$D=D_m\bigoplus_i d_iD_{f,i} \tag{4}$$

where m and f corresponds to matrix and fracture respectively, and d_i is the aperture of fracture $D_{f,i}$. Denote by $\kappa(x)=\kappa_m$ the permeability in the matrix, and $\kappa(x)=\kappa_i$ the permeability in the i-th fracture. The permeabilities of matrix and fractures can differ by orders of magnitude.

The fine-scale solution of (3) on the fine mesh $\mathcal{T}^h$ can be obtained using the standard finite element scheme, with backward Euler method for time discretization,i.e. we seek $u_f^{n+1}\in V_h$ such that:

$$\left(\frac{u_f^{n+1}-u_f^n}{\Delta t},v\right)+(\kappa\lambda^{n+1}\nabla u_f^{n+1},\nabla v)=(g^{n+1},v). \tag{5}$$

Here,$(\cdot,\cdot)$ denotes the L^2 inner product, V_h is the standard finite element space over $\mathcal{T}^h$ while $v\in V_h$ is a test function. In the matrix form, we have

$$M_fu_f^{n+1}+\Delta tA_fu_f^{n+1}=\Delta tb_f+M_fu_f^n, \tag{6}$$

where M_f and A_f are fine scale mass and stiffness matrix respectively, b_f is the right hand side vector.

For the coarse scale approximation, we assume $\mathcal{T}^H$ is a coarse-grid partition of the domain D with mesh size H (see Figure 1) for an illustration of the fine and coarse mesh, where coarse elements are blue rectangles and fine elements are unstructured black triangles. 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. For each K_i, we define the oversampled region K_i^+ to be an oversampling of K_i with a few layers of neighboring coarse blocks, see Figure 1 for the illustration of K_i and K_i^+. We will use the non-local multi-continuum approach (NLMC) [17].

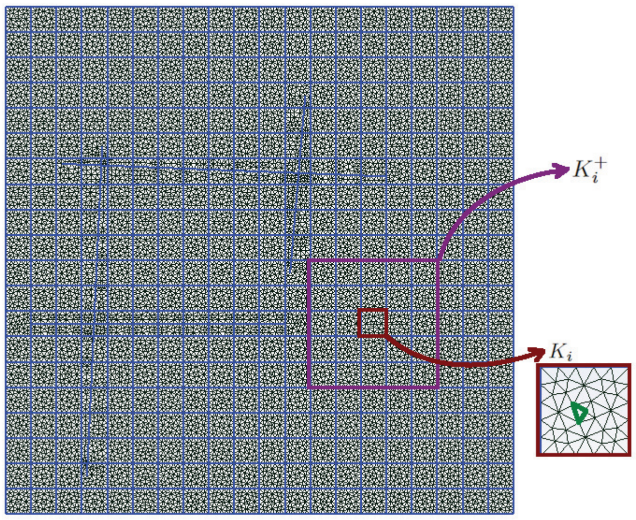


Figure 1: Illustration of coarse and fine meshes.

In the NLMC approach, the multiscale basis functions are selected such that the degrees of freedom have physical meanings and correspond to average solutions. This method derives its foundation from Constraint Energy Minimizing Generalized Multiscale Finite Element Method (CEM-GMsFEM) [16], and starts with the definition of the auxiliary space. The idea here is to use a constant as auxiliary basis for the matrix in each coarse block, and constants for each separate fracture network within each coarse block. The simplified auxiliary space uses minimal degrees of freedom in each continua, thus one can obtain an upscaled equation with a minimal size and the degrees of freedom represent the averages of the solution over each continua. To construct the multiscale basis function for NLMC, we consider an oversampling region K_i^+ of coarse block K_i, the basis $\psi_m^{(i)}$ solves the following local constraint minimizing problem on the fine grid

$$\begin{aligned} &a(\psi_m^{(i)}, v) + \sum_{K_j \subset K_i^+} \left(\mu_0^{(j)} \int_{K_j} v + \sum_{1 \le m \le L_j} \mu_m^{(j)} \int_{f_m^{(j)}} v \right) = 0, \quad \forall v \in V_0(K_i^+), \\ &\int_{K_j} \psi_m^{(i)} = \delta_{ij}\delta_{0m}, \quad \forall K_j \subset K_i^+, \\ &\int_{f_n^{(j)}} \psi_m^{(i)} = \delta_{ij}\delta_{nm}, \quad \forall f_n^{(j)} \in F^{(j)}, \, \forall K_j \subset K_i^+. \end{aligned} \tag{7}$$

where $a(u, v) = \int_{D_m} \kappa_m \lambda \nabla u \cdot \nabla v + \sum_i \int_{D_{f,i}} \kappa_i \lambda \nabla_f u \cdot \nabla_f v$, $\mu_0^{(j)}, \mu_m^{(j)}$ are Lagrange multipliers. By this way of construction, the average of the basis $\phi_0^{(i)}$ equals 1 in the matrix part of coarse element K_i, and equals 0 in other coarse blocks $K_j \subset K_i^+$ as well as any fracture inside K_i^+. As for $\phi_l^{(i)}$, it has average 1 on the l-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.

We then define the transmissibility matrix T by

$$T_{mn}^{(i,j)} = a(\psi_m^{(i)}, \psi_n^{(j)}). \tag{8}$$

We note that m, n denotes different continua, and i, j are the indices for coarse blocks. Since the multiscale basis are constructed in oversampled regions, the support of multiscale basis for different coarse degrees of freedom will overlap, and this results in non-local transfer and effective properties for multi-continuum. The mass transfer between continua m in coarse block i and continua n in coarse block j is $T_{mn}^{(i,j)}([u_T]_n^{(j)} - [u_T]_m^{(i)})$, where $[u_T]$ is the coarse scale solution.

With a simple index, we can write T (transmissibilities) in the following form

$$\begin{bmatrix} t_{11} & t_{12} & \dots & t_{1n} \\ t_{21} & t_{22} & \dots & t_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ t_{n1} & t_{n2} & \dots & t_{nn} \end{bmatrix} \tag{9}$$

where $n = \sum_{i=1}^{N}(1 + L_i)$, and $1 + L_i$ means the one matrix continua plus the number of discrete fractures in coarse block K_i, and N is the number of coarse blocks.

The upscaled model for the diffusion problem (3) will be as follows

$$M_T u^{n+1} + \Delta t A_T u^{n+1} = \Delta t b_T + M_T u^n, \tag{10}$$

where A_T is the NLMC coarse scale transmissibility matrix, i.e.

$$\begin{pmatrix} -\sum_j t_{1j} & t_{12} & \dots & t_{1n} \\ t_{21} & -\sum_j t_{2j} & \dots & t_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ t_{n1} & t_{n2} & \dots & -\sum_j t_{nj} \end{pmatrix}$$

and M_T is an approximation of coarse scale mass matrix. We note that both A_T and M_T are non-local and defined for each continua.

To this point, we obtain an upscaled model from the NLMC method. We remark that the results in [17] indicate that the upscaled equation in our modified method can use small local regions.

3 Deep Multiscale Model Learning (DMML)

3.1 Main Idea

We will utilize a rigorous NLMC model as stated in previous section to solve the coarse scale problems and use the resulting solutions in deep learning framework to approximate $\overline{F}$ in (2). The advantages of NLMC approach lie in that, one can not only get accurate approximations compared to the reference fine grid solutions, but the coarse grid solutions also have important physical meanings. That is, the coarse grid quantities are the average pressure in the corresponding matrix or fracture in a coarse block. Usually $\overline{F}$ is expensive to compute and conditioned to data. In fact, we need to recompute $\overline{F}$ whenever multiscale coefficient changes. The idea of this work is to use the coarse grid information and available observed fine data in combination with deep learning techniques to overcome this difficulty.

It's clear that the solution at the time instant $n+1$ depends on the solution at the time instant n and input parameters, such as permeability/geometry of the fractured media and source terms. Here, we would like to learn the relationship of the solutions between two consecutive time instants by a multi-layer network. If we simply take only coarse simulation data in the training process, the neural network will provide a forward map to approximate our reduced-order models.

To be specific, let m be the number of samples in the training set. Suppose for a given set of various input parameters, we use NLMC method to solve the problem and obtained the coarse grid solutions

$$\{u_1^1, \cdots, u_1^{n+1}, u_2^1, \cdots, u_2^{n+1}, \cdots, \cdots, u_m^1, \cdots, u_m^{n+1}\}$$

at all time steps for these m samples. Our goal is to use deep learning to train the coarse grid solutions and find a network $\mathcal{N}$ to describe the pushforward map between u^n and u^{n+1} for any training sample.

$$u^{n+1} \sim \mathcal{N}(u^n, I^n), \tag{11}$$

where I^n is some input parameter which can also change with respect to time, and $\mathcal{N}$ is a multi-layer network to be trained.

Remark: The proposed framework also includes nonlinear elliptic PDEs, where the map $\mathcal{N}$ corresponds to the linearised discrete system.

In deep network, we call u^n and I^n the input, and u^{n+1} the output. One can take the coarse solutions from time step 1 (initial time instant) to time step n as input, and from time 2 to $n+1$ (final time instant) as corresponding output in the training process. In this case, a universal neural net $\mathcal{N}$ can be obtained. With that being said, the solution at time 1 can be forwarded all the way to time $n+1$ by repeatedly applying the universal network n times, that is

$$u^{n+1} \sim \mathcal{N}(\mathcal{N} \cdots \mathcal{N}(u^1, I^1) \cdots, I^{n-1}), I^n). \tag{12}$$

Then in the future testing/predicting procedure, given a new coarse scale solution at initial time u^1_{new}, we can also easily obtain the solution at final time step by the deep neural network

$$u^{n+1}_{\text{new}} \sim \mathcal{N}(\mathcal{N} \cdots \mathcal{N}(u^1_{\text{new}}, I^1) \cdots, I^{n-1}), I^n). \tag{13}$$

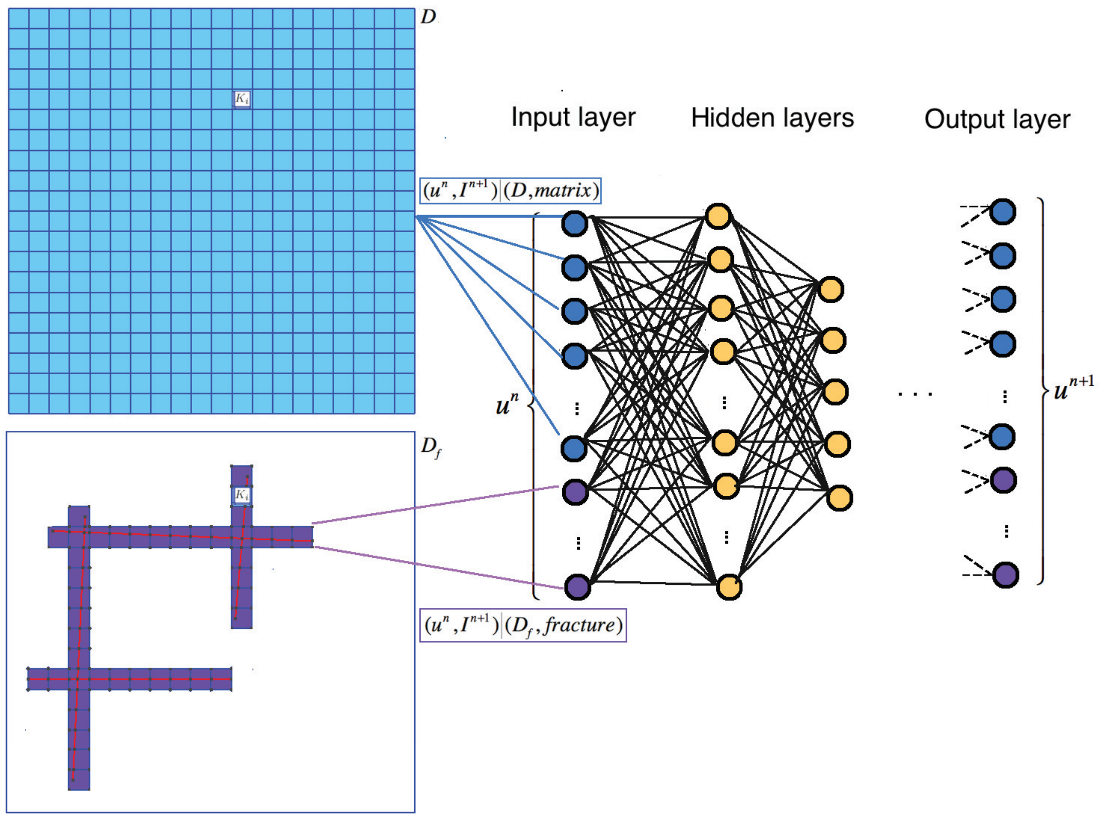
Actually, one may train each forward map for any two consecutive time instants if the problem is highly nonlinear as needed. That is, we will have $u^{j+1} \sim \mathcal{N}_j(u^j, I^j)$, for $j = 1, \cdots, n$. In this case, to predict the final time solution u^{n+1}_{new} given the solution at initial time ${u_{\text{new}}}^1$, we use n different networks $\mathcal{N}_1, \cdots, \mathcal{N}_n$

$$u^{n+1}_{\text{new}} \sim \mathcal{N}_n(\mathcal{N}_{n-1} \cdots \mathcal{N}_1(u^1_{\text{new}}, I^1) \cdots, I^{n-1}), I^n).$$

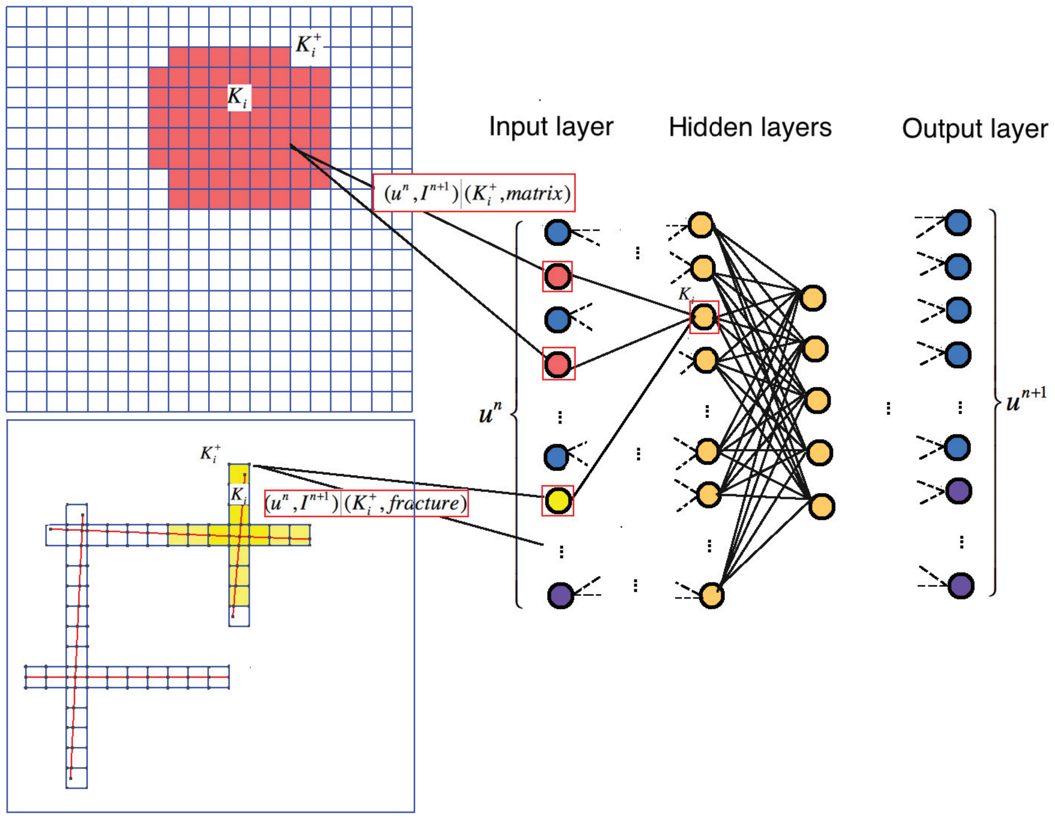
We would like to remark that, besides the previous time step solutions, the other input parameters I^n such as permeability or source terms can be different when entering the network at different time steps. In our numerical example, a universal network will give us reasonably good results, so we didn't investigate this further more.

As mentioned previously, we can also design the connections in the neural network using the idea of "regions of influence". We remark that it is important to use reduced-order model, since it will identify the regions of influence and appropriate numbers of variables. In NLMC approach, we construct a non-local multi-continuum transmissibility matrix, which provides us some information about the connections between coarse parameters. It has been shown in [17] that, due to the exponential decay of the global basis function away from the target coarse region, one can use the constructed local basis functions which are supported in a small oversampling region (compared with the whole domain) to solve the problem, and get good accuracy in solutions compared with those using global basis functions. Moreover, the transmissibility matrix formed by these local basis functions also indicates local connections (in a slightly larger oversampling region) between a target coarse degree of freedom (dofs) with others. Taking the advantage of the underlying NLMC model, we can simplify the problem when designing neural networks. Typically, for specific coarse degrees of freedom (corresponding to a coarse block or a fracture in the coarse block) of the solution at time instant $n+1$, we can only activate the connections between this coarse degree of freedom and the coarse scale degrees of freedom in some oversampling neighborhood at time instant n. The advantage of defining regions of the influence is to reduce the complexity of the deep network. An illustration of the comparison between deep neural nets with full connections, or with local connections indicated by region of influence is shown in Figure 2. In Figure 2 (a), we illustrate the deep network using full connections, where the input layer and the first hidden layer of the network is fully connected. This means, for example, the matrix continua parameter in any coarse block K_i is always connected with all the dofs in the matrix continua (blue shaded coarse blocks/neurons) and all the dofs in the fracture continua (purple shaded coarse blocks/neurons). On the other hand, in Figure 2 (b), the network applies some local connections in the first layer. For the coarse block K_i, it only connects with some matrix continua dofs (pink shaded areas/neurons) and some fracture continua dofs (yellow shaded areas/neurons).

Furthermore, the map we are interested in is the relation from u^n to u^{n+1} in the field rather than the multiscale model only. If the observed fine data is sufficient enough, there's not much need to reconstruct



(a) Deep network using full connections


K_i^+
K_i
Input layer
Hidden layers
Output layer
$(u^n, I^{n+1})|(K_i^+, matrix)$
u^n
u^{n+1}
$(u^n, I^{n+1})|(K_i^+, fracture)$

(b) Deep network using local connections indicated by the region of influence.

Figure 2: Comparison of deep nets with full connections or local connections indicated by with region of influence.

the relationship through various methods as the data itself is a one to one mapping. But this is not the case, the observed fine data available to us is not enough. This limitation actually makes it impossible to reconstruct the true map. As a compensation, we will use an approximation of fine grid model, i.e. NLMC, to work as a raw but computation-efficient model, and correct it with observed fine data. We modified and improved the NLMC model in the sense that we are able to adjust it to fit observed fine data which is hard for forward construction of NLMC. More specifically, we can see later from the numerical example that the corrected model can approximate the fine scale model better.

To be specific, in this work, we will incorporate available observed fine data in the neural net, which will modify the reduced order model and improve the performance of the model such that the new model will take into account observed fine data effects. In other words, we utilize the neural network to interpolated two models: the fine scale model and the NLMC model. First, we introduce some notations.

denote the coarse simulation data by

$$\{u_s^1, \cdots, u_s^{n+1}\}$$

denote the observed fine data by

$$\{u_o^1, \cdots, u_o^{n+1}\}$$

at all time steps for these m samples. To get the observed data, we can (1) perturb the simulation data, (2) perturb the permeability or geometry of the fractured media, run a new simulation and use the results as observed data, (3) use available observed fine data. In this work, we will investigate the effects of taking into account observed fine data to train deep neural nets.

As a comparison, there are three networks we will consider:

- Network A: Use all observed fine data in the training,

$$u_o^{n+1} \sim \mathcal{N}_o(u_o^n, I^n) \tag{14}$$

- Network B: Use a mixture of observed fine data and coarse simulation data in the training,

$$u_{\text{mixed}}^{n+1} \sim \mathcal{N}_m(u_{\text{mixed}}^n, I^n) \tag{15}$$

- Network C: Use all coarse simulation data (no observed fine data) in the training,

$$u_s^{n+1} \sim \mathcal{N}_s(u_s^n, I^n) \tag{16}$$

where u_{mixed} is a mixture of coarse simulation data and observed fine data.

In Network A, we assume the observed fine data is sufficient, and train the observed fine data at time $n+1$ as a function of the observed fine data at time n. In this case, the map fits the data in a very good manner but will ignore the coarse model if the data are obtained without using underlying simulation model in any sense. This is usually not the case in reality, since the observed fine data are expensive to get and deep learning requires a large amount of data to make the training effective. In Network C, we simply take all coarse simulation data in the training process. For this network, one will get a network describes the coarse simulation model (in our example, the NLMC model) as best as it can but ignore the observed fine data effects. This network can serve as an emulator (simplified forward map, which avoids deriving/solving coarse-grid models) to do a fast simulation. We will utilize Network A and C results as references, and investigate more about Network B. Network B is the one where we take a combination of coarse simulation data and observed fine data to train. It will not only take into account the information provided by the NLMC model but also use the observed fine data to modify the model.

We expect that the proposed algorithm will provide new upscaled model that can honor the data while it follows our general multiscale concepts.

3.2 Network structures

Generally, in deep learning, let the function $\mathcal{N}$ be a network of L layers, x be the input and y be the corresponding output. We write

$$\mathcal{N}(x;\theta) = \sigma(W_L\sigma(\cdots\sigma(W_2\sigma(W_1x+b_1)+b_2)\cdots)+b_L)$$

where $\theta := (W_1,\cdots,W_L,b_1,\cdots,b_L)$, W's are the weight matrices and b's are the bias vectors, and σ is the activation function. Suppose we are given a collection of example pairs (x_j, y_j). The goal is then to find θ^* by solving an optimization problem

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \frac{1}{N}\sum_{j=1}^{N} ||y_j - \mathcal{N}(x_j;\theta)||_2^2,$$

where N is the number of the samples. We note that the function $\frac{1}{N}\sum_{j=1}^{N} ||y_j - \mathcal{N}(x_j;\theta)||_2^2$ to be optimized is called the loss function. Other important aspects in tuning the deep neural network are to choose suitable number of layers, number of neurons in each layer, the activation function, the loss function and the optimizers for the network.

In our example, without loss of generality, we suppose that there are uncertainties in the injection rates g, i.e., the value or the position of the sources can vary among samples. Suppose we have a set of different realizations of the source $\{g_1, g_2, \cdots, g_m\}$, where m is a sufficiently large number, we need to run simulation based on NLMC model and take the solutions as data for deep learning. To obtain the observed fine data, we will solve the problem with the same set of source realizations using the fine scale model.

As discussed in the previous section, we consider three different networks, namely $\mathcal{N}_o$, $\mathcal{N}_m$ and $\mathcal{N}_s$. For each of these networks, we take the vector $x = (u_\alpha^n, g^n)$ ($\alpha = o, m, s$) containing the coarse scale solution vectors and the source term in a particular time step as the input. As discussed before, we can take the input coarse scale quantities in the whole domain D or in the region of influence K^+. Based on the availability of the observed fine data in the example pairs, we will define an appropriate network among (14), (15) and (16) accordingly. The output $y = {u_\alpha}^{n+1}$ is taken as coarse scale solution in the next time step, where $\alpha = o, m, s$ corresponds to three networks.

Assume with enough samples of the source terms $\{g_1, \cdots, g_m\}$, there exist some coarse simulation data u_s and some observed fine data u_o, and we can train deep neural networks to well approximate the function $\overline{F}$ in (1) with respect to the loss functions. Then for some new source term g_{m+1}, given the coarse scale solution at time instant n, we expect our networks output $\mathcal{N}(u_\alpha^n, g_{m+1}^n; \theta^*)$ which is close to the output data u_o^{n+1}.

Here, we briefly summarize the architecture of networks $\mathcal{N}_\alpha$ ($\alpha = o, m, s$) we defined in (14), (15) and (16) respectively.

- Input: $x = (u_\alpha^n, g^n)$ is the vector containing the coarse scale solution vectors and the source term in a particular time step.
- Output: $y = {u_\alpha}^{n+1}$ is the coarse scale solution in the next time step.
- Sample pairs: $N = mn$ example pairs of (x_j, y_j) are collected, where m is the number of samples of flow dynamics and n is the number of time steps.
- Loss function: This cost function which is typical for regression problems is the mean squared error function: $\frac{1}{N}\sum_{j=1}^{N} \|y_j - \mathcal{N}_\alpha(x_j;\theta)\|_2^2$. In our numerical examples, our output is the coarse grid solution vector, so we would like to consider the relative L^2 error as the loss function, i.e. $\frac{1}{N}\sum_{j=1}^{N} \dfrac{||y_j - \mathcal{N}_\alpha(x_j;\theta)||_{L^2}}{||y_j||_{L^2}}$
- Weighted loss function: In building a network in $\mathcal{N}_m$ by using a mixture of N_1 pairs of observed fine data $\{(x_j, y_j)\}_{j=1}^{N_1}$ and N_2 pairs of coarse simulation data $\{(x_j, y_j)\}_{j=N_1+1}^{N}$, where $N_1 + N_2 = N$, we may consider using weighted loss function, i.e,

$$w_1 \sum_{j=1}^{N_1} \frac{||y_j - \mathcal{N}_\alpha(x_j;\theta)||_{L^2}}{||y_j||_{L^2}} + w_2 \sum_{j=N_1+1}^{N} \frac{||y_j - \mathcal{N}_\alpha(x_j;\theta)||_{L^2}}{||y_j||_{L^2}},$$

where $w_1 > w_2$ are user-defined weights. In the scenarios of limited observed fine data, the training process is supplemented with coarse simulation data, resulting in an intermediate approximation of the two classes of data. However, the imbalanced data, i.e. high ratio of coarse simulation data, will result in a biased model and reduce the ability to predict rare points. To this end, a weighted loss function is employed to undersample the coarse simulation data and oversample the observed data.

- Activation function: The popular ReLU function (the rectified linear unit activation function) is a common choice for activation function in training deep neural network architectures [28]. In our numerical we use ReLU activation in the hidden layers, and use linear activation in the output layer.
- DNN structure: The number of layers and the number of neurons in each layer are specified in the following numerical examples. For the connection between the input neurons and first layer neurons, we will use dense connections or our self-designed local connections, and compare their performance in the numerical examples.
- Training Optimizer: We use AdaMax [36], a stochastic gradient descent (SGD) type algorithm well-suited for high-dimensional parameter space, in minimizing the loss function.

We remark that, the optimization problems are in most cases non-convex.The universal algorithm used to optimize such problems is gradient descent. The advantage of such an algorithm is computational efficiency while it can only guarantee a local minimum instead of a global one. However, in practice, we do not bother to find the global minimum due to the overfitting concerns. A local minimum with acceptable training error would be ideal in our case.

4 Numerical examples

In this section, we present some representative numerical results. We consider the fractured media as shown in the Figure 3, where the red lines denotes the fractures. The permeability of the matrix is $\kappa_m = 1$, and the permeability of the fractures are $\kappa_f = 10^3$. We assume that the observed fine data samples are obtained from the fine scale model solution for different source terms. As for the corresponding coarse simulation data, we compute the NLMC solution on a 20 by 20 coarse grid as shown in Figure 3, using the same sets of source terms and permeability field. Thus, the observed fine data and coarse simulation data are generated from different underlying models. As discussed in [17], the local basis are constructed in oversampled regions of each coarse block. In our numerical examples, we choose 2 layers of oversampling. With this small oversampling size, the NLMC simulation is fast, but as a trade-off, it sacrifice a little accuracy.

As for the similarity/difference between the observed fine data and coarse simulation data, we perform hypothesis tests using empirical MMD estimates as the test statistic, the results show that the observed fine data and coarse simulation data are from the same distribution. This indicates that we can use coarse simulation data can be computational inexpensive surrogate data as observed fine data for the training of the neural network. On the other hand, there are some differences between the observed fine data and coarse simulation data. Generally, in practice, the relative upscaling errors for the NLMC solution and averaged fine scale solution are $5\% - 15\%$ in L^2 norm.

We will train the networks N_s, N_o and N_m using the coarse simulation data samples (from NLMC model), fine observed data samples (from fine scale model), and the mixture of them, respectively.

All the network training are performed using the Python deep learning API Keras [9] with TensorFlow framework.

4.1 Example 1

In our first example, we use a time independent, constant mobility coefficient $\lambda = 1$. For the source term, we use a piece-wise constant function. Namely, in one of the coarse blocks, the value of g is a positive number c, in another coarse block, the value of $g = -c$, and $g = 0$ elsewhere. This represents a two-well source, one of the block is injection well, the other is production well. Randomly selecting the location of the two wells, with the constraint that the two wells are well separated in the domain, we get source samples $g_1, \cdots, g_{1000}$.

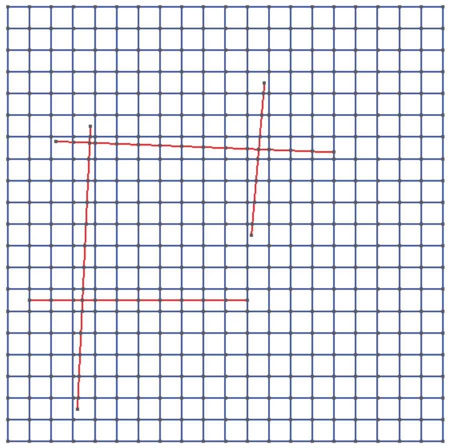


Figure 3: Geometry (permeability) for obtaining both coarse simulation and fine observed data.

In this example, we set the values of the source to be time independent. The equation (3) is solved using both the fine scale and NLMC model, where we set $T = 0.1$, and divide T into 10 time steps.

For the 1000 source terms, we use the solutions corresponds to the last 100 for validation, i.e, they won't be seen in the training process. The comparison of training and test data can be seen in different perspective. First, we can take a look at the source terms which we vary when generating samples. Here, we have 1000 different sources. Every source term differs from its well location. We test the maximum mean discrepancies (MMD) on the source terms corresponding to the training and testing sets. Using Gaussian radial basis kernel function with $\sigma = 3.59$, the hypothesis tests using empirical MMD estimates indicate that the two sets of sources are drawn from the same distribution. Second, we can also take a look at the statistics of the input (which are the flow problem solutions for these source terms). The inputs are high dimensional, and the hypothesis tests using empirical MMD estimates again indicate that the two sets of sources are drawn from the same distribution.

We will use 300, 500 and 900 out of the first 900 solutions to train the network separately. We remark that, we make sure the random source terms are distinct when generating samples, thus the input samples are unique.

As discussed before in (11), we would like to find a universal deep network to describe the map between two time steps. We use the solution at time step n as input, and solution at time step $n+1$ as corresponding output, where $n = 1, 2, 3, 4.....9$. That is, each sample pair is in the form of $(u^n, u^{n+1}), n = 1, 2, ...9$. For example, the solutions corresponding to 300 different training source terms result in $300 * 9 = 2700$ samples, and the solutions corresponding to 100 testing source terms result in $100 * 9 = 900$ testing samples, where the multiplier 9 stands for 9 time steps (time steps 1 to 9, or time steps 2 to 10).

We will test the performance of the three networks (14), (15), and (16). For N_s, we take all training samples to be the NLMC solutions. In this case, there is no observed fine data in training, and the network will only approximate the NLMC model. For N_o, we use all the fine scale solutions as training samples, thus the network aim to approximate the fine grid model and will be used as reference. As for N_m, we take half training samples from u_s and the other half from u_o. Specifically, we assume the observed fine data are given for some well configurations while for other configurations observed fine data can only be replaced by simulation data due to their rarity. This is the case when the network is trained partially with observed fine data. We expect the trained network N_m to produce an improved model compared with N_s. In the training process, we also consider both the full connections and local connections indicated by the region of influence input (see Figure 2), where we use multiscale concepts to reduce the region of influence (connections) between the nodes.

We remark that, a solve of the NLMC is around 0.031s, an inference step is around 0.0012s.

4.1.1 Full connection

The three networks are firstly constructed adopting the structure of DNN with densely connected layers. The input and output both have dimension 445, which is the degree of freedom in the NLMC model associated with the 20 by 20 coarse grid and the fracture configuration as shown Figure 3. For this example, since the mobility coefficient is constant which makes the map linear, we only take 1 layer with 445 neurons. The activation function is chosen to be linear at the output layer. The training was performed over 50 epochs, and the batch size is chosen to be 100. We use the Adam algorithm as the optimizer, and the learning rate is 0.002. The number of trainable parameters in this network is $198,470$. The training and validation losses for N_o are plotted in the left of Figure 4, and they have similar behavior for the other two networks. We remark that, in order to compare the performance for different size of training data set (namely $2700, 4500, 8100$), in the figure, the training losses (vs epochs) are depicted at the end of learning all samples in the whole data set instead of after every batch. We observe that, for each data set, the validation loss is very close to the training loss, which indicates the network performs and generalizes well.

Next, in the testing procedure, we input an identical input data from the 900 testing samples as described before to three networks defined in (14), (15), (16). To compare the outputs of three networks with observed fine data, we compute the mean of relative L^2 errors between the observed fine data(samples computed from fine grid model) and predicted data (obtained from the output of neural networks) , i.e

$$\frac{1}{N}\sum_{j=1}^{N}\frac{||u_o^{n+1}-N_\alpha(u_\alpha^n, I^{n+1})||_{L^2}}{||u_o^{n+1}||_{L^2}}, \quad \alpha = o, s, m.$$

The results are shown in Table 1. We can see that, with a mixture of coarse simulation data and observed fine data (the second column in the table), we can get a better model, since the mean error of $\left\|\mathcal{N}_m(u_\alpha^n, I^n) - u_o^{n+1}\right\|$ among testing samples is closer to that of $\left\|\mathcal{N}_o(u_\alpha^n, I^n) - u_o^{n+1}\right\|$, compared to the mean error of $\left\|\mathcal{N}_s(u_\alpha^n, I^n) - u_o^{n+1}\right\|$ among testing samples.

Number of Training Samples	Using $\mathcal{N}_o$	Using $\mathcal{N}_m$	Using $\mathcal{N}_s$
2700	6.5	6.7	7.1
4500	4.8	5.1	5.7
8100	3.7	4.0	4.7

Table 1: Example 1, fully connected network. Network parameters: $198,470$. Mean errors (%) between prediction and true solutions for two consecutive time steps, 900 samples are tested for three different networks.

4.1.2 Sparse connection: Region of influence

Though we have obtained promising results using fully connected networks, the number of trainable parameters is quite large. In this section, we would like to design a locally-connected layer in the neural network by taking advantage of the region of influence in the underlying model. This can help to reduce the trainable parameters in the network and further reduce the required training resources considering observed fine data are expensive to acquire.

The idea is to reduce the full connections among the neurons between two layers. Thanks to the NLMC model, with which we derive the transmissibility matrix T that describes the nonlocal connections of coarse degrees of freedom, not only spatially but also across continua. This inspires us to define the connections between the input neuron to the first layer. That is, we design a layer with the number of trainable weight parameters equal to the nonzero entries in the matrix T, this can reduce the number of parameters due to the sparsity of T. The defined sparse weight will only activate the connections between the input and nodes in the next layer as indicated in the transmissibility matrix. The following hidden layers will still be fully connected if they exist.

Remark 2: If the underlying permeability field changes, then the transmissibility matrix will change. Then one needs to redesign the network.

Remark 3: We remark that a direct application of CNN is not trivial for our problem. Since the samples we used in the network training are the coarse scale solutions for the multi-continuum model, which contains the degrees of freedom from both matrix and fracture continuum. In our example, the degrees of freedom for the matrix continuum lies in the 20 by 20 coarse grid (which is 400), but the additional fracture degrees of freedom only lies in the coarse blocks which contain the fractures (which is 45 in the geometry shown in Figure 3). The multi-continuum solutions thus can not be directly represented by a square image which is needed for CNN. Furthermore, according to the NLMC model, the transmissibility connections exist not only among coarse blocks but also among different continua, and the effects of the fracture continua should not be ignored. We then tried to extend the fracture continua solution to the 20 by 20 grid using zero padding, and input matrix and fracture continua solutions as two-channel images to train CNN network. But the zero padding procedure increases the number of training parameters (instead of decreasing as expected when using CNN), since we enlarge the input dimension from 445 to 800. And the training accuracy is also affected in a bad sense because the network has the additional burden to learn the zeros in the second channel. Thus, a self defined locally connected layer is needed.

Remark 4: We comment that in our case, we also tried out LSTM netwokrs to predict the solutions at the last time step. However, we do not see prominent advantage using such structure for the following reasons: 1) RNN is difficult for us to implement because of computational cost given our computational resources and our problem sizes. The training of an RNN could be fairly hardware-demanding. 2) LSTM is good at dealing with time series where the prediction is dependent on a series of previous time events. But mainly, we would like to emphasize that in our case, we are interested in the map connecting solutions at two consecutive time steps. Remembering previous events is unnecessary here. 3) The number of trainable coefficients will significantly increase with RNN. So, a locally-connected feed-forward structure would perfectly fit in our need.

We would like to compare the predicted results from the neural networks trained with different types of coarse parameters. Specifically, two types of networks are trained with the coarse parameters in the whole domain and parameters only in the region of influence, respectively. Comparing Table 1 and Table 2, we can see that, using the region of influence idea can result in similar results for all three networks $\mathcal{N}_o$, $\mathcal{N}_m$ and $\mathcal{N}_s$ when we use similar network parameters such as the number of layers, number of neurons (445) in each layer, training epochs (50), learning rate (0.002), loss functions (relative l^2 error) and activation functions (linear). The training/validation losses are plotted in the right of Figure 4 for Locally Connected Networks (LCN). We observe that the losses of LCN decay faster compared with those in DNN. As for the number of trainable parameters, it is $198,470$ for the fully connected network, but is only $28,107$ for the sparsely connected network. This suggests that, the data in the region of influence of the underlying model is of dominant importance in deciding the outputs and thus enables reduction in training effort.

Number of Training Samples	Using $\mathcal{N}_o$	Using $\mathcal{N}_m$	Using $\mathcal{N}_s$
2700	6.2	6.4	6.7
4500	5.1	5.3	5.8
8100	4.3	4.6	5.1

Table 2: Example 1, locally connected network. Network parameters: $28,107$. Mean errors (%) between prediction and true solutions for two consecutive time steps, 900 samples are tested for three different networks.

4.2 Example 2

In our second example, we use heterogeneous time-dependent mobility and source term. Here, we fix the location of the source term and vary the value of the source. The distributions of the mobility in some time steps are shown in Figure 5, which is from two-phase flow mobility. The source term in the right hand side is defined as follows. At $0 \leq x \leq 0.1, 0 \leq y \leq 0.1$, we have $g = 10[(\sin(\alpha x))^2 + (\sin(\beta y))^2]$ denotes an injection well, and at $0.9 \leq x \leq 1.0, 0.9 \leq y \leq 1.0$ we have $g = -10[(\sin(\alpha x))^2 + (\sin(\beta y))^2]$ denotes an production well, where the parameters α and β are randomly chosen in each time step, and are different among samples (which are obtained using these different source terms g). So for each sample, we have the different values

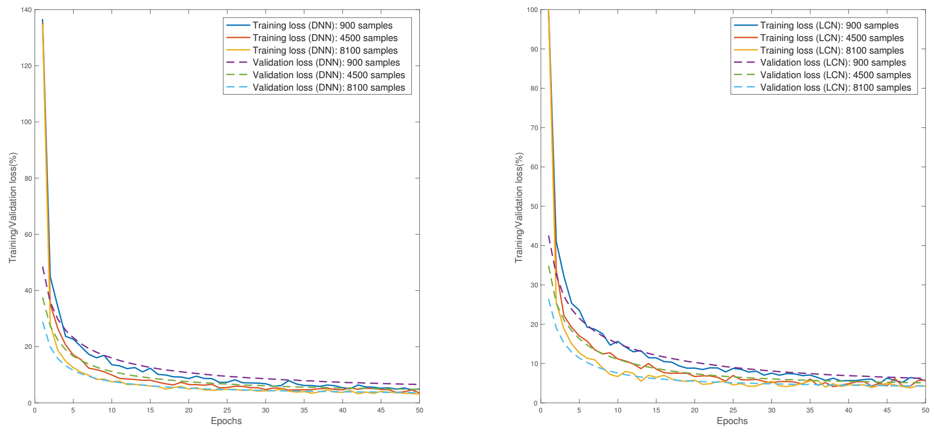
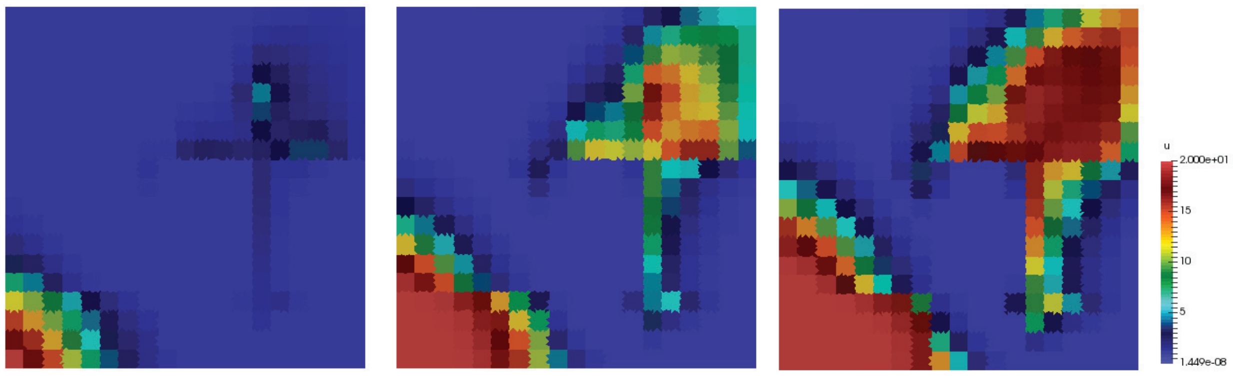


Figure 4: DNN(left) and Locally-connected Network(right) training/validation losses over epochs for N_o, with different number of samples.

of the source term, and, in each sample, the source term is time dependent.



(a) Mobility at time $t = 0.01$ (b) Mobility at time $t = 0.05$ (c) Mobility at time $t = 0.1$

Figure 5: Illustration of mobility $\lambda(x, t)$.

For the coarse simulation data u_s, we use the solution obtained from NLMC model for n random source terms. We note that, $n = 200, 600$ and 1000 in this example, and we take solutions associated with $100, 500, 900$ sources out of them for training correspondingly, and the other 100 for validation/testing.

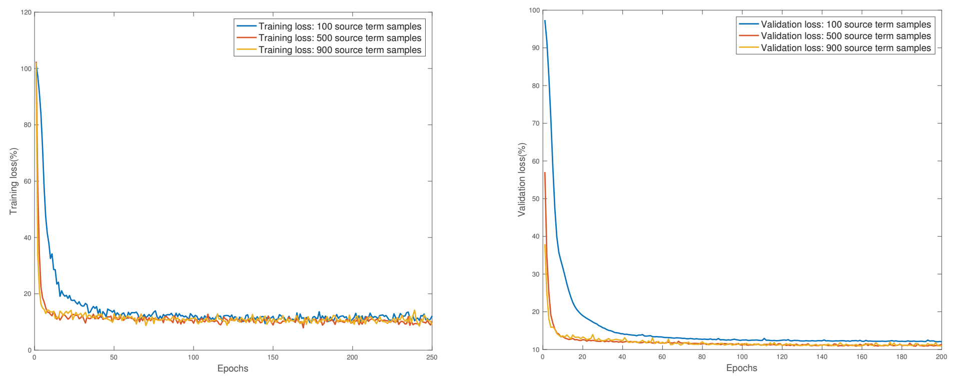
For the observed fine data u_o, we first solve the system from the fine grid model using the same set of source terms, and use the coarse degree of freedom fine solution average as the observed fine population. As for the mixture u_m of coarse simulation and observed fine data, we take the samples relating to half of the sources from u_s, and the samples relating to another half of the sources from u_o. In practice, to explain the mixture data u_m, we can assume we have the observed fine data in the whole domain given some well configurations, but for some other well configurations, we only have simulation results.

4.2.1 Full connection

We first build the three deep neural networks with densely connected layers. Due to the nonliearity of the underlying problem in this example, we will take 4 hidden layers with ReLU activation function, and use linear activation at the output layer. The input vector has dimension 445 as before, which is the degree of freedom in the NLMC model. The first hidden layer has also 445 neurons, and they are fully connected with the input. We take 50 neurons in the other three hidden layers. And the output has the dimension 445.

We solve the equation (3) with $T = 0.1$. Similarly, we use the solutions at time step 1 to time step 9 as input data, and from at time step 2 to time step 10 as output data. In this example, we have $900, 4500, 8100$ training sample pairs, respectively. The validation set then has 900 samples in each case.

In the training process, we take the batch size to be 100. For the loss function, we use the relative L^2 error between the samples (computed from fine grid model/ NLMC model) and predicted data (the output of the neural networks), same as in Example 1. We remark that (u^n, I^{n+1}) is taken to be $(u^n + \triangle t \cdot g^{n+1})$ in this examples, where g is the time dependent source term. The training and validation loss for the network N_o are shown in Figure 6, the loss history for N_s and N_m are similar.



(a) Training loss (b) Validation loss

Figure 6: DNN training/validation loss vs epochs for N_o, with different number of samples.

As we discussed before, we can use (12) or (13) to forward the solution from the initial time step to the final time step using the "universal" deep neural nets. Assume we have 10 time steps in total, for a given solution u_1 at the initial time, we will apply $\mathcal{N}_\alpha$ for $\alpha = o, m, s$ for 9 times to obtain the the final time predictions. That is,

$$u^{10}_{pred,\alpha} = \underbrace{\mathcal{N}_\alpha \circ \mathcal{N}_\alpha \circ \cdots \circ \mathcal{N}_\alpha}_{9 \text{ times}}(u^1)$$

for $\alpha = o, m, s$

Finally, we compare the final time predictions $u^{10}_{pred,\alpha}$ (for $\alpha = o, m, s$) with the observed fine data at the final time step given u^1_s. The results are shown in Table 3. There are 100 samples to test in total. As we increasing the number of training samples, it is clear that $\left\|u^{10}_{pred,o} - u^{10}_o\right\|$ is decreasing. For $\left\|u^{10}_{pred,m} - u^{10}_o\right\|$, with the increasing of training sample size, it will give better and better prediction results. When we take 900 samples, the mean of $\left\|u^{10}_{pred,m} - u^{10}_o\right\|$ over testing samples is 16.8%, which is very close to the reference case where $\left\|u^{10}_{pred,o} - u^{10}_o\right\|$ has mean error 13.5%. This indicates that using a mixture of coarse simulation data and observed fine data can enhance the performance of NLMC model induced by deep learning, as we compared the predictions to the observed fine data (which is the fine grid solution). We also show the comparison between one of the samples in Figure 7, where the solution produced by the network N_m has almost the same accuracy as the solution produced by N_o, and is much more accurate than that of N_s.

4.2.2 Sparse connection: Region of influence

In this section, we examine the region of influence input for this example. Different from Example 1, the DNN network here has more layers. We will just replace the first fully connected layer with a locally connected layer. In this section, we only show the results for the case that the training samples are generated corresponding to 500 different source terms. The other two sample sets perform similarly. The training and validation loss over some epochs are shown in Figure 8. Compared the fully connected network and locally

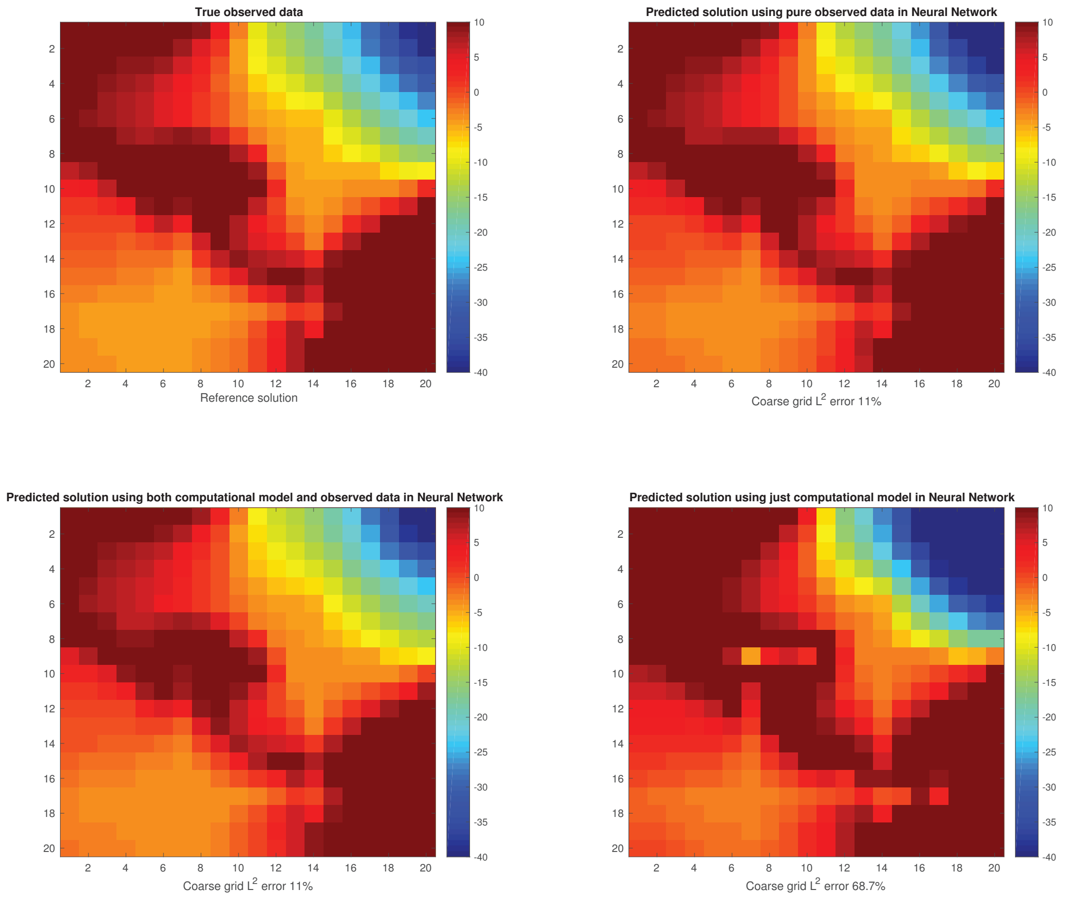
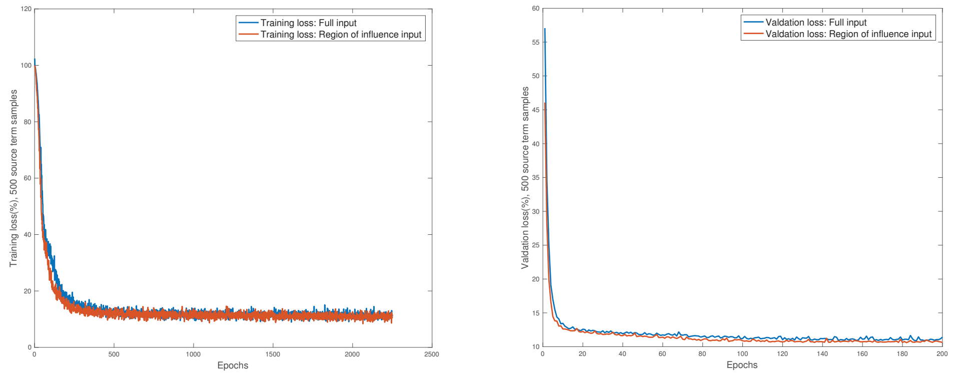


Figure 7: Example 2, fully connected network. DNN predicted solutions' comparison for one of the samples.

	Errors (%)		
Number of Training Samples	$\|u^{10}_{pred,o} - u^{10}_o\|$	$\|u^{10}_{pred,m} - u^{10}_o\|$	$\|u^{10}_{pred,s} - u^{10}_o\|$
900	20.2	30.6	31.1
4500	14.6	17.8	37.1
8100	13.5	16.8	45.9

Table 3: Example 2, fully connected network. Mean error between final time step prediction and true solutions over 100 testing samples for three different networks.

connected network, we can see that the training loss decays faster for locally connected network, and the validation loss have very similar behavior for both networks. We remark that, the number of trainable parameters in fully connected network is $318,315$, and that number in locally connected network is $154,422$, where we only replace the first layer by a self defined locally connected layer. All the other hyperparameters in both networks are chosen to be the same. The mean errors for the same testing samples as in the previous section are presented in Table 4. Compared with Table 3, we observe slightly better results are obtained by locally connected network.



(a) Training loss (b) Validation loss

Figure 8: Comparison of Fully connected network and locally connected network. Training/validation loss vs epochs for N_o, number of source samples is 500.

	Errors (%)		
Number of Training Samples	$\|u^{10}_{pred,o} - u^{10}_o\|$	$\|u^{10}_{pred,m} - u^{10}_o\|$	$\|u^{10}_{pred,s} - u^{10}_o\|$
4500	13.7	17.9	33.5

Table 4: Example 2, locally connected network. Mean error between final time step prediction and true solutions over 100 testing samples for three different networks.

5 Conclusions

The paper uses deep learning techniques to derive and modify upscaled models for nonlinear PDEs. In particular, we combine multiscale model reduction (non-local multi-continuum upscaling) and deep learning techniques in obtaining better approximations of the underlying models, which takes into account observed data. Multi-layer networks provide a nonlinear mapping between the time steps, where the mapping has a certain structure. The multiscale concepts, used in multi-layer networks, provide appropriate coarse-grid variables, their connectivity information, and some information about the mapping. However, constructing complete and accurate nonlinear push-forward map is expensive, if not impossible, in general multiscale simulations. Moreover, these models will not honor the available observed fine data. Deep Multiscale Model Reduction Learning (DMML). We present numerical results, where we test our main concepts. We show that the regions of influence derived from upscaling concepts can lighten the computations. Our approach indicates that incorporating some observed fine data in the training can improve the coarse grid model. Similarly, incorporating some coarse simulation data to the observed fine data can improve the predictions, when there is not sufficient observed data. The use of coarse-degrees of freedom is another main advantage of our method. Finally, we use observed data and show that DMML can obtain accurate solutions, which can honor the observed data. In conclusion, we believe DMML can be used as a new coarse-grid model for complex nonlinear problems with observed data, where upscaling of the computational model is expensive and may not accurately represent the true observed model.

Acknowledgment

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 NSF 1620318, the U.S. Department of Energy Office of Science, Office of Advanced Scientific Computing Research, Applied Mathematics program under Award Number DE-FG02-13ER26165 and National Priorities Research Program grant NPRP grant 7-1482-1278 from the Qatar National Research Fund. YE would also like to acknowledge the support of Mega-grant of the Russian Federation Government (N 14.Y26.31.0013).

References

[1] Assyr Abdulle and Yun Bai. Adaptive reduced basis finite element heterogeneous multiscale method. *Computer Methods in Applied Mechanics and Engineering*, 257:203–220, 2013.

[2] G. Allaire and R. Brizzi. A multiscale finite element method for numerical homogenization. *SIAM Multiscale Modeling & Simulation*, 4(3):790–812, 2005.

[3] Manal Alotaibi, Victor M. Calo, Yalchin Efendiev, Juan Galvis, and Mehdi Ghommem. Global–local nonlinear model reduction for flows in heterogeneous porous media. *Computer Methods in Applied Mechanics and Engineering*, 292:122–137, 2015.

[4] T. Arbogast. Implementation of a locally conservative numerical subgrid upscaling scheme for two-phase Darcy flow. *Computational Geosciences*, 6:453–481, 2002.

[5] I. Bilionis and N. Zabaras. Solution of inverse problems with limited forward solver evaluations: a bayesian perspective. *Inverse Problems*, 30(015004), 2013.

[6] Ilias Bilionis, Nicholas Zabaras, Bledar A. Konomi, and Guang Lin. Multi-output separable gaussian process: Towards an efficient, fully bayesian paradigm for uncertainty quantification. *Journal of Computational Physics*, 241:212–239, 2013.

[7] Donald L Brown and Daniel Peterseim. A multiscale method for porous microstructures. *SIAM Multiscale Modeling & Simulation*, 14(3), 2016.

[8] V. Calo, Y. Efendiev, J. Galvis, and M. Ghommem. Multiscale empirical interpolation for solving nonlinear pdes using generalized multiscale finite element methods. *Journal of Computational Physics*, 278:204–22, 2014.

[9] François Chollet et al. Keras. `https://keras.io`, 2015.

[10] E. Chung, Y. Efendiev, and S. Fu. Generalized multiscale finite element method for elasticity equations. *International Journal on Geomathematics*, 5(2):225–254, 2014.

[11] E. Chung, Y. Efendiev, and W. T. Leung. Generalized multiscale finite element method for wave propagation in heterogeneous media. *SIAM Multiscale Modeling & Simulation*, 12:1691–1721, 2014.

[12] E. Chung and W. T. Leung. A sub-grid structure enhanced discontinuous galerkin method for multiscale diffusion and convection-diffusion problems. *Communications in Computational Physics*, 14:370–392, 2013.

[13] E. T. Chung, Y. Efendiev, W.T. Leung, M. Vasilyeva, and Y. Wang. Online adaptive local multiscale model reduction for heterogeneous problems in perforated domains. *Applicable Analysis*, 96(12):2002–2031, 2017.

[14] E. T. Chung, Y. Efendiev, and G. Li. An adaptive GMsFEM for high contrast flow problems. *Journal Computational Physics*, 273:54–76, 2014.

[15] Eric Chung, Maria Vasilyeva, and Yating Wang. A conservative local multiscale model reduction technique for stokes flows in heterogeneous perforated domains. *Journal of Computational and Applied Mathematics*, 321:389–405, 2017.

[16] Eric T Chung, Yalchin Efendiev, and Wing Tat Leung. Constraint energy minimizing generalized multiscale finite element method. *Computer Methods in Applied Mechanics and Engineering*, 339:298–3194, 2018.

[17] Eric T Chung, Yalchin Efendiev, Wing Tat Leung, Maria Vasilyeva, and Yating Wang. Non-local multi-continua upscaling for flows in heterogeneous fractured media. *Journal of Computational Physics*, 327:22–34, 2018.

[18] Eric T Chung, Yalchin Efendiev, Wing Tat Leung, and Mary Wheeler. Nonlinear nonlocal multicontinua upscaling framework and its applications. *International Journal for Multiscale Computational Engineering*, 16(5), 2018.

[19] Balázs Csanád Csáji. Approximation with artificial neural networks. *Faculty of Sciences, Etvs Lornd University*, 24(48), 2001.

[20] G. Cybenko. Approximations by superpositions of sigmoidal functions. *Mathematics of Control, Signals, and Systems*, 2(4):303–314, 1989.

[21] Martin Drohmann, Bernard Haasdonk, and Mario Ohlberger. Reduced basis approximation for nonlinear parametrized evolution equations based on empirical operator interpolation. *SIAM Journal on Scientific Computing*, 34(2):A937–A969, 2012.

[22] W. E and B. Engquist. Heterogeneous multiscale methods. *Comm. Math. Sci.*, 1(1):87–132, 2003.

[23] Y. Efendiev, J. Galvis, and T. Y. Hou. Generalized multiscale finite element methods (gmsfem). *Journal of Computational Physics*, 251:116–135, 2013.

[24] Y. Efendiev, J. Galvis, and X.H. Wu. Multiscale finite element methods for high-contrast problems using local spectral basis functions. *Journal of Computational Physics*, 230:937–955, 2011.

[25] Y. Efendiev, T. Hou, and V. Ginting. Multiscale finite element methods for nonlinear problems and their applications. *Communications in Mathematical Sciences*, 2:553–589, 2004.

[26] Jacob Fish and Wen Chen. Space–time multiscale model for wave propagation in heterogeneous media. *Computer Methods in applied mechanics and engineering*, 193(45):4837–4856, 2004.

[27] Jacob Fish and Rong Fan. Mathematical homogenization of nonperiodic heterogeneous media subjected to large deformation transient loading. *International Journal for numerical methods in engineering*, 76(7):1044–1064, 2008.

[28] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 315–323. PMLR, 2011.

[29] Boris Hanin. Universal function approximation by deep neural nets with bounded width and relu activations. *arXiv:1708.02691*, 2017.

[30] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[31] Patrick Henning and Mario Ohlberger. The heterogeneous multiscale finite element method for elliptic homogenization problems in perforated domains. *Numerische Mathematik*, 113(4):601–629, 2009.

[32] Geoffrey Hinton, Li Deng, Dong Yu, George E. Dahl, Abdel rahman Mohamed, Navdeep Jaitly, and Andrew Senior. Approximation capabilities of multilayer feedforward networks. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[33] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991.

[34] T.J.R. Hughes, G.R. Feijóo, L. Mazzei, and J.-B. Quincy. The variational multiscale method - a paradigm for computational mechanics. *Computer methods in applied mechanics and engineering*, 127:3–24, 1998.

[35] Yuehaw Khoo, Jianfeng Lu, and Lexing Ying. Solving parametric pde problems with artificial neural networks. *arXiv:1707.03351*, 2017.

[36] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[37] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, pages 1097–1105, 2012.

[38] Zhen Li and Zuoqiang Shi. Deep residual learning and pdes on manifold. *arXiv:1708.05115.*, 2017.

[39] I. Lunati and P. Jenny. The multiscale finite volume method: A flexible tool to model physically complex flow in porous media. In *10th European Conference on the Mathematics of Oil Recovery*, Amsterdam, The Netherlands, 2006.

[40] X. Ma, M. Al-Harbi, A. Datta-Gupta, and Y. Efendiev. A multistage sampling approach to quantifying uncertainty during history matching geological models. *SPE Journal*, 13(10):77–87, 2008.

[41] Ana-Maria Matache and Christoph Schwab. Two-scale fem for homogenization problems. *ESAIM: Mathematical Modelling and Numerical Analysis*, 36(04):537–572, 2002.

[42] A. Mondal, Y. Efendiev, B. Mallick, and A. Datta-Gupta. Bayesian uncertainty quantification for flows in heterogeneous porous media using reversible jump Markov Chain Monte-Carlo methods. *Advances in Water Resources*, 33(3):241–256, 2010.

[43] H. Owhadi and L. Zhang. Metric-based upscaling. *Communications on Pure and Applied Mathematics: A Journal Issued by the Courant Institute of Mathematical Sciences*, 60:675–723, 2007.

[44] H. Mhaskar Q. Liao and T. Poggio. Learning functions: when is deep better than shallow. *arXiv:1603.00988v4*, 2016.

[45] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations. *arXiv preprint arXiv:1711.10561*, 2017.

[46] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics informed deep learning (part ii): Data-driven discovery of nonlinear partial differential equations. *arXiv preprint arXiv:1711.10566*, 2017.

[47] M. Telgrasky. Benefits of depth in neural nets. *JMLR: Workshop and Conference Proceedings*, 49(123), 2016.

[48] Min Wang, Siu Wun Cheung, Eric T Chung, Yalchin Efendiev, Wing Tat Leung, and Yating Wang. Prediction of discretization of gmsfem using deep learning. *Mathematics*, 7(5):412.

[49] E. Weinan and Bing Yu. The deep ritz method: A deep learning-based numerical algorithm for solving variational problems. *Communications in Mathematics and Statistics*, 6(1):1–12, 2018.

[50] Yinhao Zhu and Nicholas Zabaras. Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification. *Journal of Computational Physics*, 366:415–447, 2018.

Declaration of interests

☒ 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.

☐The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: