Efficient Deep Learning Techniques for Multiphase Flow Simulation in Heterogeneous Porous Media

Yating Wang [*]   Guang Lin [†‡]

July 24, 2019

Abstract

We present efficient deep learning techniques for approximating flow and transport equations for both single phase and two-phase flow problems. The proposed methods take advantages of the sparsity structures in the underlying discrete systems and can be served as efficient alternatives to the system solvers at the full order. In particular, for the flow problem, we design a network with convolutional and locally connected layers to perform model reductions. Moreover, we employ a custom loss function to impose local mass conservation constraints. This helps to preserve the physical property of velocity solution which we are interested in learning. For the saturation problem, we propose a residual type of network to approximate the dynamics. Our main contribution here is the design of custom sparsely connected layers which take into account the inherent sparse interaction between the input and output. After training, the approximated feed-forward map can be applied iteratively to predict solutions in the long range. Our trained networks, especially in two-phase flow where the maps are nonlinear, show their great potential in accurately approximating the underlying physical system and improvement in computational efficiency. Some numerical experiments are performed and discussed to demonstrate the performance of our proposed techniques.

1   Introduction

Various physical phenomena in engineering applications are described by flow and transport problem in porous media, including reservoir engineering, climate dynamics, material science and so on. The underlying problems naturally exhibit heterogeneities covering from large physical scales down to micro-scales. Numerical simulations for these problems are challenging due to a rich class of length scales and potential uncertainties. In the past decades, numerous model reduction techniques and multiscale methods are proposed to design alternative models with great computational efficiency as well as desired accuracy. Local model reduction techniques typically involve building representations of underlying heterogeneity using some basis functions or effective coarse grid properties, and then constructing a form of the coarse level equations [3, 24, 5, 23, 19, 7, 12, 22, 2, 27, 28, 37, 36, 30, 18, 16, 14, 15, 17]. For large scale dynamical system, global reduced order models adopting Proper Orthogonal Decomposition method, Krylov subspace projection method, etc, are proposed to approximate the state-space problems. Though both local and global model reduction techniques have been extensively applied in many problems, reduced order models may have complicated forms in the linear case, not mentioning in nonlinear settings [25, 4, 8, 46].

Recently, deep learning has attracted growing attention in a rich class of applications. It has gained revolutionary results in image, speech, and text recognition [35, 31, 29]. The potential of deep neural networks lies in their great capacity in approximating high-dimensional nonlinear maps. There are extensive efforts devoted to learning the expressivity of deep neural nets theoretically, just to mention a few [21, 32, 20, 42, 38], where the universal approximation property are investigated and the approximation ability of deep networks to a rich classes of functions are shown. This motivates a number of works bringing deep neural

---

[*]Department of Mathematics, Purdue University, West Lafayette, IN 47907, USA (`wang4190@purdue.edu`)

[†]Department of Mathematics, Purdue University, West Lafayette, IN 47907, USA & Department of Mechanical Engineering, Purdue University, West Lafayette, 47907, USA (`guanglin@purdue.edu`)

[‡]Corresponding author.

networks to the field of scientific computing. Some recent works include numerically solving parametric partial differential equations [33], designing multigrid solvers [26], constructing efficient multiscale models [43], learning surrogate models by deep convolution networks [49, 45], incorporating reduced order models in conjunction with observation data [44], solving ODE systems driven by data [9, 39], combining deep learning concepts and some proper orthogonal decomposition (POD) model reduction methods [10, 48] and so on. In particular, the authors in [44] propose novel methods to learn the nonlinear feed-forward map considering the sparsity nature of the underlying problem, and authors in [26] also deals with nonlinear maps issued from underlying physical problems by designing multiscale neural networks.

The focus of this work is to apply sparse learning ideas on the coupled flow and transport systems in both single phase and two-phase settings. Our contribution is to design appropriate deep neural networks as computational efficient alternative models to the complicated physical problems. We start with the single-phase case, where the flow problem is described by Darcy's law and the transport dynamics is driven by a velocity field. Motivated by ideas from model reduction algorithms, we design a neural network architecture to learn the maps between some input (such as the source term or the permeability field) and output (velocity solutions) in the flow problem. In this flow approximation task, we will first transform the input to some coarse features by using convolutional layers or average pooling layers. It is worth mentioning that, although traditional multi-layer perceptron models with densely connected layers were successfully utilized in many applications, they suffer from the curse of dimensionality because of the full connectivity between layers. To handle this issue, it's natural to come up with some locally connected networks which can benefit from the fact that neighboring nodes have an inherent relationship. For instance, convolutional layers/ locally connected layers have gained remarkable success and performed superior to others. Hence, in our network design for flow approximation, some locally connected layers are adopted. They do not only take into account the spatial structure of data but also provide sufficient learning capacity to extract rich hidden features. We remark that the locally connected layer is similar to the convolutional layer, but a different set of filters is applied at each different patch of the input. After some coarse features are extracted from the input layer, we then consider working on the coarse level deeply to learn more hidden properties, which is inspired by the analogy to the multiscale method. In the end, a decoding step is realized by a single densely connected layer between the coarse features and the fine velocity solution. Another important aspect of our proposed network is that it preserves the physical property of the approximating quantity. The velocity field used in the training process is locally mass conservative. One expects the predicted solution produced from the trained neural network can also maintain the desired physical property. We tackle this issue by imposing constraints with physical meaning in the loss function while training the network. Our numerical results show that the proposed framework did help to improve the local mass conservative property among predictions. As for the transport equation, we are interested in learning the dynamics between saturation solutions at different time steps. To be specific, the solution state at the time $n+1$ depends on the solution at the time step n and other parameters, such as velocity fields and source terms. It is natural to employ a residual network structure in approximating the feed-forward map. In this approximation task, we still want to rely on the sparsity of the discrete system to define unknowns and design sparse custom network layers. More specifically, our contribution in this paper is to design the Sparse Velocity Layer, which treats the velocity input as multipliers of the weight matrices and also take advantages of the data sparsity structure of the discrete system to create sparse weight matrices. This will result in a notable reduction in the number of trainable parameters for the weight matrices and require less training data due to the simplification of the network. We present detailed network architecture as well as several numerical experiments. The numerical results illustrate the capability of the proposed network and implies both the accuracy and efficiency of our methods.

Solving two-phase system is challenging due to its nonlinearity. We apply our experiences in designing neural networks for single-phase flow and transport approximations, and appropriately generalize them to the two-phase problems. In classical numerical methods, the two-phase flow and saturation equations are solved sequentially. Since the flow velocity not only depends on the absolute permeability but also the relative permeability, which is a function of the phase saturation. Thus one needs to update the velocity field at every time step. Furthermore, the saturation equation is also nonlinear and some iterative methods are required at each time step to obtain the solution. With this regard, we expect to alleviate the computational burden using deep neural networks. The ideas of constructing approximation maps for velocity problem and saturation dynamics are based on the previously proposed architectures. For flow approximation, our

2

main contribution is to develop the input-output locally connected network between the total mobility (a function of saturation) and the velocity solution. For the nonlinear saturation approximation, one of the difficulties is that the velocity field varies in every saturation forward map. However, it's not trivial to utilize every velocity input as multipliers for the weight matrices, the same ways as we did in the single flow case. A workaround is that we compute the mean of the velocity fields in each training batch, and feed in the mean velocity field during the training process of the nonlinear saturation map. The sparse structure of the system is still realized when designing the weight matrices. After both networks for approximating flow and saturation equations are trained, we propose a sequential algorithm to recursively employ the trained networks to predict saturation solutions in the long range. Our numerical results show great potential both in accuracy and computational improvement for approximating the two-phase flow system.

The paper is organized as follows. In Section 2, we present deep neural network construction in the single-phase flow, as well as corresponding numerical experiments. In Section 3, the generalization, and adjustment of the proposed methods to two-phase flow problems are illustrated, numerical validations are followed after the methodologies. A conclusion is presented in Section 4.

2 Single phase flow

In the single phase flow, we consider solving the flow problem by the mixed finite element method (MFEM). One can take RT_0 (the lowest order Raviart-Thomas element) to approximate velocity, and P_0 (piecewise constant element) to approximate pressure. It is well known that MFEM is locally mass conservative. More precisely, we aim to solve the velocity u and pressure p from a mixed system

$$
\begin{aligned}
\kappa^{-1} u + \nabla p &= 0 & \text{in} \quad D \\
\operatorname{div}(u) &= f & \text{in} \quad D \\
u \cdot n &= 0 & \text{on} \quad \partial D
\end{aligned}
$$

where κ denotes the heterogeneous permeability field (see Figure 4 as an example), D represents the computational domain, and f is source term.

The fine grid of the problem reads

$$
\begin{aligned}
a(u, v) + b(v, p) &= 0 & \text{for all } v \in V_h \\
b(u, q) &= -(f, q) & \text{for all } q \in Q_h
\end{aligned}
$$

where $a(u, v) = \int_D \kappa^{-1} u \cdot v$, and $b(v, p) = -\int_D p \operatorname{div} v$. V_h and Q_h are the finite element space for velocity and pressure, respectively.

In addition to the flow equation, the transport equation of saturation S is given by

$$
\frac{\partial S}{\partial t} + u \cdot \nabla S = r
$$

where u is the velocity field obtained in the flow problem.

The transport of saturation can be solved by finite volume method on the fine grid. One can discretize the time derivative using the forward Euler scheme, where the time step size can be chosen from the CFL condition. To be specific, on a fine grid K_i, the value S_i at time t^{n+1} can be obtained by

$$
|K_i| \frac{S_i^{n+1} - S_i^n}{dt} + \sum_{e_j \in \partial K_i} F_{ij}(S^n) = |K_i| r_i \tag{1}
$$

where F_{ij} is the upwind flux, i.e.

$$
F_{ij}(S^n) = \begin{cases} \int_{e_j} (u_{ij} \cdot n) S_i^n & \text{if} \quad u_{ij} \cdot n \geq 0 \\ \int_{e_j} (u_{ij} \cdot n) S_j^n & \text{if} \quad u_{ij} \cdot n < 0 \end{cases}
$$

Here e_j denotes the edge shared by fine grids K_i and K_j, u_{ij} the velocity on the edge e_j.

3

The above flow and transport problem is solved sequentially. One first solves the flow equation to obtain the velocity, and the obtained velocity filed is used to drive the saturation. However, once the source term f or the permeability changed, the system needs to be solved again, which is computationally expensive. We are interested in developing efficient alternatives for the high fidelity model and obtain the desired accuracy with reduced computational effort.

2.1 Model reduction for flow problem using neural network

2.1.1 Discrete flow problem solver

We note that the discrete flow problem has the following matrix formulation on the fine grid $\mathcal{T}_h$

$$\begin{bmatrix} A_h(\kappa) & B_h^T \\ B_h & 0 \end{bmatrix} \begin{bmatrix} u_h \\ p_h \end{bmatrix} = \begin{bmatrix} 0 \\ -F \end{bmatrix} \tag{2}$$

In practice, one may need to solve the system with thousands of different κ (when there are uncertainties in the permeability field) or many different source terms f. Typically, the linear system (2) generated from MFEM on the fine grid can be solved using a direct solver, however, the computational expense grows heavily with the complexity of heterogeneity in the media and number of simulations.

To tackle this difficulty, one way is to solve the problem on a coarser grid $\mathcal{T}_H$. There are many mixed multiscale methods in literature, for example [13, 6, 1]. We assume the multiscale basis are computed for velocity in each local coarse region, and the piecewise constant on the coarse blocks are used for pressure. Let R_u be the matrix with size $N_u^H \times N_u^h$ consisting of multiscale velocity basis in each row, where N_u^H and N_u^h are the coarse and fine degree of freedom (DOF) for velocity, respectively. Let R_p be the matrix with size $N_p^H \times N_p^h$, which works as an averaging of fine-scale pressure unknowns over each coarse block. R_u and R_p together form a restriction matrix. And the coarse scale system has the form

$$\begin{bmatrix} A_H & B_H^T \\ B_H & 0 \end{bmatrix} \begin{bmatrix} u_H \\ p_H \end{bmatrix} = \begin{bmatrix} R_u & 0 \\ 0 & R_p \end{bmatrix} \begin{bmatrix} A_h(\kappa) & B_h^T \\ B_h & 0 \end{bmatrix} \begin{bmatrix} R_u^T & 0 \\ 0 & R_p^T \end{bmatrix} \begin{bmatrix} u_H \\ p_H \end{bmatrix} = \begin{bmatrix} 0 \\ -F_H \end{bmatrix} \tag{3}$$

One then can solve the coarse scale system (3), for instance, using the Arrow-Hurwicz iterative method

$$\begin{aligned} u_H^{k+1} &= u_H^k + \alpha(A_H u_H^k - B_H^T p_H^k) \\ p_H^{k+1} &= p_H^k + \omega(B_H u_H^{k+1} + F_H) \end{aligned} \tag{4}$$

where $\alpha > 0$, $\omega > 0$ are relaxation parameters.

Let the initial guesses p_H^0 and u_H^0 be zero, then after one iteration, we observe that p_H can be obtained by multiplying F_H by some matrix. Next, we obtain u_H by plugging in p_H in the first equation of (4). This indicates we can view p_H as some inherent unknowns and iteratively obtain the velocity solution on the coarse grid. Next, one can obtain the fine scale velocity by a downscaling step $u_h = R_u^T u_H$.

We remark that p_H will be some hidden coarse grid properties in our deep learning algorithm. In the end, we are interested in the velocity solution only.

2.1.2 Network architecture

The focus of this work is to utilize a deep neural network on the approximation of problems of interest. Before that, let's first briefly introduce some basics of deep learning.

Generally, in deep learning, let the function $\mathcal{N}$ be a network of d layers, the i-th layer is denoted by l_i ($i = 1, \cdots, m$). Let x be the input and y be the corresponding output. We write

$$\mathcal{N}(x; \theta) = \sigma(l_d \sigma(\cdots \sigma(l_2 \sigma(l_1(x))\cdots)))$$

where σ is the activation function. Denote by θ all the trainable parameters in the network. Suppose we are given a collection of sample pairs (x_j, y_j). The goal is to find θ^* by solving an optimization problem

$$\theta^* = \underset{\theta}{\text{argmin}} \frac{1}{N} \sum_{j=1}^N \mathcal{L}(y_j, \mathcal{N}(x_j; \theta)),$$

4

where N is the number of the samples. This implies that one aims to find parameter θ^* which can minimize the mean loss $\mathcal{L}(y_j, \mathcal{N}(x_j; \theta))$ among all training samples using stochastic gradient descent iteratively. Then, the trained network $\mathcal{N}(x; \theta^*)$ will be applied to make predictions on new inputs x_{new}.

To improve the efficiency of solving the flow problem, we are interested in learning the discrete velocity solution u_h (outputs) given different permeability fields κ or sources f (inputs). From now on, we omit the subscript h and just use the simple notation u instead. That is, we would like to use pre-computed samples pairs to train a neural network which approximates the map $u = \mathcal{N}(\kappa; \theta)$, or $u = \mathcal{N}(f; \theta)$. When given a test case, one can use the trained network to predict the quantity of interest effectively.

In the single phase flow case, we consider the map between f and u which is linear, a brute-force way is to use one dense layer to connect the input and output. However, as mentioned before, due to the multiple scales of the underlying permeability (see Figure 4 for illustration), one needs to use the sufficiently fine grid in order to resolve all scale property, which results in large dimensions of f and velocity solution on the fine grid. Then a direct dense connection between the input and output will not be affordable.

With this background in mind, we will propose a network architecture which is deeper and aims to perform the model reduction as described in the previous section. The proposed network will have a small number of trainable parameters compared with a fully connected neural network.

One of the core ingredients we will adopt in the network architecture is the locally connected layer. Let α be the number of filters, each filter has a size $k \times k$, s is the stride size, then the connection between two tensors Y^1 and Y^2 by a locally connected layer as follows

$$Y_{i;k}^1 = \sigma\left(\sum_{j=(i-1)s+1}^{(i-1)s+k^2} \sum_{l=1}^{\alpha} W_{i,j;k,l} Y_{j;l}^2 + b_{i;k} \right) \tag{5}$$

The locally connected layer shares some similarities with the convolutional layer, the difference lies in that a different set of filters is applied at each different patch of the input.

The proposed network consists of the following components. One first project the input tensor f on a coarser grid using an average pooling layer. For example, the input dimension of f is $\sqrt{N_p^h} \times \sqrt{N_p^h}$, where $1/\sqrt{N_p^h}$ is the size of fine grid in the computational domain. We then let the size of a coarse block to be $1/\sqrt{N_p^H}$. Then the average pooling layer are specified to have a pooling size of $\left(\sqrt{N_p^h}/\sqrt{N_p^H}\right) \times \left(\sqrt{N_p^h}/\sqrt{N_p^H}\right)$, which will transform the input from the fine grid level to the coarse grid level. Next, a flattened layer followed by a dense layer with N_p^H neurons is applied to the intermediate output from the previous step. In the dense layer, a square weight matrix is multiplied to its input, which results in some hidden coarse grid property with the same dimension $N_p^H \times 1$. Further, after reshaping the hidden coarse grid property to $\sqrt{N_p^H} \times \sqrt{N_p^H}$, we use a few locally connected layers to dig in the hidden features. Then, we flatten the resulting hidden features and connect them with the neurons with size $N_u^H \times 1$, where N_u^H is the number of DOF for velocity on the coarse grid. In the end, the coarse level features are downscaled/decoded to fine grid velocity output $N_u^h \times 1$ by a densely connected layer. An illustration of the network architecture is presented in Figure 1.
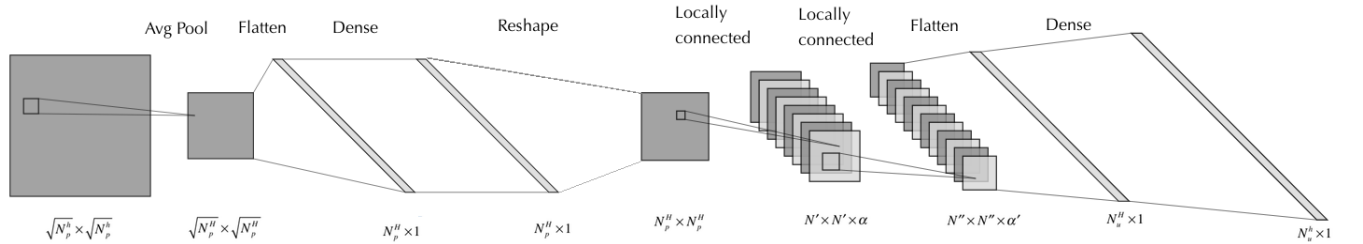

Figure 1: An illustration of the network architecture for flow approximation.

Furthermore, we construct the following constraint loss function in the deep learning process

$$\min_{\theta} \quad \frac{1}{N} \sum_{i=1}^{N} \left(\frac{||u_{\text{pred, i}} - u_{\text{true},i}||_2}{||u_{\text{true},i}||_2} + \beta ||B(u_{\text{pred},i} - u_{\text{true},i})||_2 \right) \tag{6}$$

where $u_{\text{pred}} = \mathcal{N}(\kappa; \theta)$ or $u_{\text{pred}} = \mathcal{N}(f; \theta)$ depends on the training samples, N is the number of samples, and B is the B_h matrix in (2), β is a regularization constant.

We remark that the second term in the loss function (6) helps to retain the local mass conservation of the predicted solution. The value of β strikes a balance between minimizing the relative L^2 error of the true and predicted values of the velocity and forcing the predicted velocity solutions satisfying the physical constraint. We note that if the relative L^2 error of the true and predicted values of the velocity is small enough, the physical constraint may be achieved automatically. When this part of the loss decrease to some extent, the physical based loss comes to play. In our work, we naively use the grid search to select β. We did several experiments on the training data when $\beta = \{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 10^0\} \times \frac{\text{std}(||u_{\text{true}}||)}{\text{std}(|M_{\text{true}}|)}$, where $\frac{\text{std}(||u_{\text{true}}||)}{\text{std}(|M_{\text{true}}|)}$ is the factor between the scales of physical based loss and the mean squared loss. Then we use $10^{-3} \times \frac{\text{std}(||u_{\text{true}}||)}{\text{std}(M_{\text{true}})}$ which gives the relatively better results in the experiments.

With this custom loss function, the training is then carried out using the Adam optimization algorithm to minimize the loss evaluated on batches of data from the training set.

2.2 Transport problem using sparse learning

2.2.1 Feed-forward map in transport problem

Next, we will move to the saturation equation. The saturation equation (1) can be discretized and written in the matrix equation as follows

$$S^{n+1} = S^n + \mathrm{d}t(F(u)S^n + R) \tag{7}$$

Suppose the computational domain is partitioned using squares as fine cells. Consider the fine cell on k-th row and h-th column, according to the labels in Figure 2, we can rewrite the saturation feed-forward map for this fine degree of freedom as follows

$$
\begin{aligned}
S_{k,h}^{n+1} = S_{k,h}^n &+ \frac{\mathrm{d}t}{|e|}(\mathrm{Relu}(u_{kh}^1 \cdot \boldsymbol{n}) - \mathrm{Relu}(u_{kh}^2 \cdot \boldsymbol{n}) + \mathrm{Relu}(u_{kh}^3 \cdot \boldsymbol{n}) - \mathrm{Relu}(u_{kh}^4) \cdot \boldsymbol{n})S_{k,h}^n \\
&+ \frac{\mathrm{d}t}{|e|}(\mathrm{Relu}(-u_{kh}^1 \cdot \boldsymbol{n})S_{k-1,h}^n + \mathrm{Relu}(-u_{kh}^2 \cdot \boldsymbol{n})S_{k+1,h}^n \\
&+ \mathrm{Relu}(-u_{kh}^3 \cdot \boldsymbol{n})S_{k,h-1}^n + \mathrm{Relu}(-u_{kh}^4 \cdot \boldsymbol{n})S_{k,h+1}^n) + \mathrm{d}t\, r_{kh}
\end{aligned}
\tag{8}
$$

where $\boldsymbol{n}$ is the unit outward normal vector, $|e|$ denotes the length of the edge.
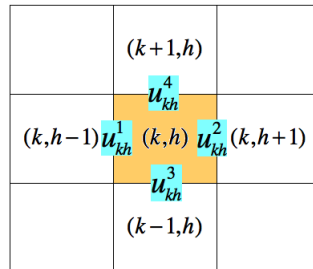

Figure 2: An illustration of (k, h) cell and its neighborhood.

2.2.2 Network structure

Given the flow velocity (either is trained from the deep network as described in the previous section, or obtained directly from MFEM solver), we can design a residual network to learn the dynamics of saturation equation. That is, we would like to approximate the feed-forward map from S^n (input) to S^{n+1} (output) using a deep neural network $\mathcal{M}$.

It is clear that the velocity field plays an important rule in the dynamics. However, if we directly let both velocity and saturation be input nodes to the network which are connected to the next layer neurons, the connections will be substantially large and may lead to an over-complicated neural network with redundancy. However, we can see from (7) that, the impact of velocity are stored in the matrix $F(u)$. Since the matrix $F(u)$ in (7) is sparse and the sparsity structure (a pentadiagonal matrix) is known, we will utilize the velocity values as well as the sparse structure of $F(u)$ to design some sparse weight matrices in the neural network. The sparse property will reduce the number of trainable parameters during the training.

To be specific, we first save the velocity solution obtained in Section 2.1 in four directions separately (corresponding to four edges of each fine cell), i.e. $U = [u^1, u^2, u^3, u^4]$, where each column vector $u^\cdot$ has dimension $N_s^h \times 1$, where N_s^h is the number of fine scale cells in the computational domain.

Next, we design a sparse custom layer, called *Sparse Velocity Layer.* Motivated by Equation (8), we introduce four pentadiagonal matrices W_i ($i = 1, 2, 3, 4$). Let I and J be the row and column indices of the pentadiagonal matrix, and V_i be vectors consisting some random values drawn from a normal distribution, for $i = 1, 2, 3, 4$. The size of I, J, V_i should be consistent. We initialize four sparse weight matrices as follows

$$W_i = \text{sparse}(I, J, V_i), \quad i = 1, 2, 3, 4. \tag{9}$$

where V_i are trainable parameters.

The Sparse Velocity Layer is defined by the following function

$$\phi_i(S^n, u^i) = \sigma((W_i \circ u^i)S^n) \tag{10}$$

where $\circ$ denotes the element-wise multiplication with broadcasting. To be specific, W_i is a sparse matrix whose corresponding dense shape is $N_s^h \times N_s^h$, u^i is an $N_s^h \times 1$ column vector, $W_i \circ u^i$ computes the element-wise product of each column of sparse matrix W_i and u^i. The latter multiplication of $(W_i \circ u^i)$ and S^n is the standard matrix-vector multiplication.

Then we can design the following network to model the dynamics of the transport equation

$$\mathcal{M}(S^n; u) = \sum_{i=1}^{4} \phi_i(S^n, u^i) + S^n \tag{11}$$

where the network contains an addition of residual part $\sum_{i=1}^{4} \phi_i(S^n, u^i)$ with the input S^n, as shown in Figure 3.

Once the network $\mathcal{M}$ is trained as an accurate surrogate model to approximate the map between saturation solutions at two consecutive time steps, we can use it iteratively several times to predict saturation solutions in the long range.

Remark 1: Remark 1: We remark that the equations of the problem are assumed to be known in this work, and we use both simulation data and the knowledge of the equations for the training of neural networks. Our approach doesn't require large data samples for the training, and can be used as an efficient surrogate solver for the investigated problems. When there are only simulation data without knowing physics, one may train surrogate models as a supervised learning task using methods like Gaussion process for small dataset, and some deep learning approaches for large data set. These data-driven methods usually relies on sufficient training data, which may be expensive to obtain for complex problems. If the underlying physics are also known, there are extensive work on physics informed neural networks (PINN) [40, 41]. This approach naturally encodes any underlying physical laws as prior information, and is successfully applied to solve PDE problems constrained to obey the law of physics that govern the data. Another development of physics-constrained surrogate modeling for stochastic PDEs without simulation data are proposed in [50]. This work incorporates the governing equations in the loss function to enforce physical constraints without solving PDEs to obtain labeled/output data for the training. These developments shed lights on
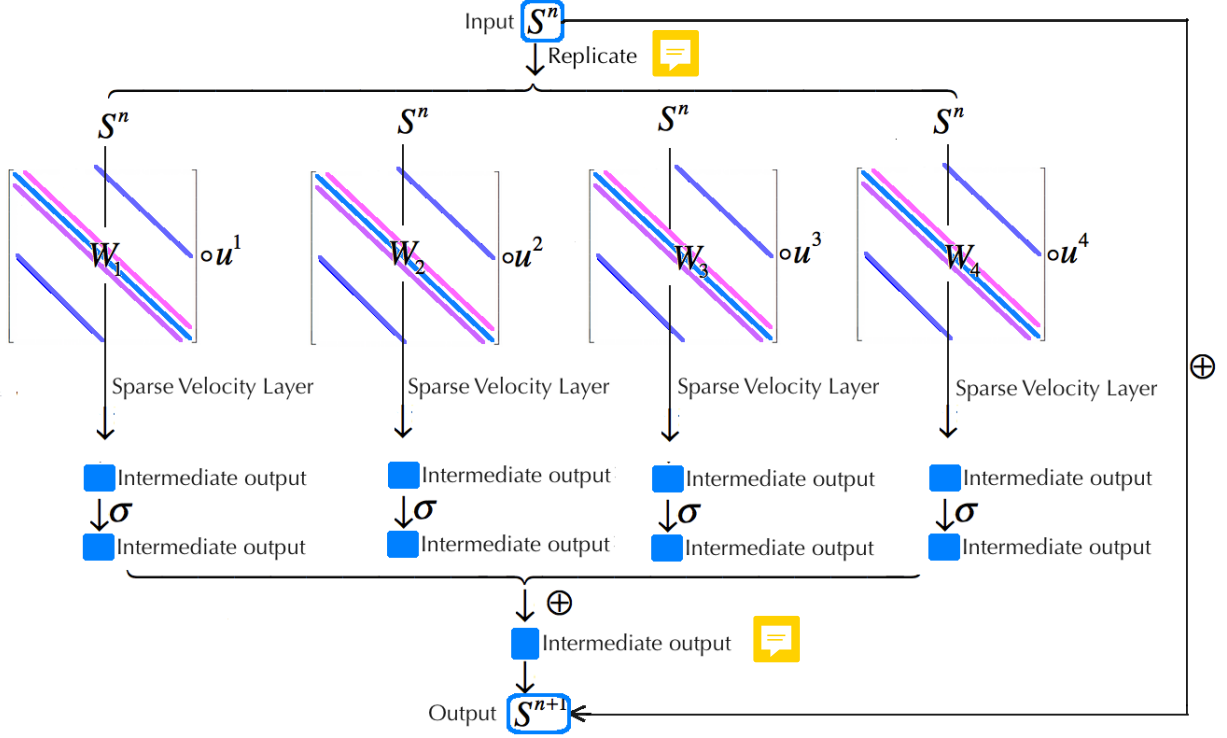
Figure 3: An illustration of the neural network architecture for learning saturation problem.

the combination of physics constraint surrogates and data driven surrogates for PDE systems. Our future work includes investigating the problems when there are uncertainty in the underlying physics or only part of the physics are known, and developing methods to combine physics and data based on current approach.

2.3 Numerical results for the single phase flow case

2.3.1 Numerical test on neural network approximation of the flow equation

In this example, we will validate our proposed network for single phase flow problem. We consider the five spot reservoir configuration of the source term. Four injection wells are placed at the corners of the computational domain, and one production well is located at the center of the domain. By randomly varying the values of the injection rates, we generate different sources terms. Using these samples, we apply the mixed fem to solve for the velocity solutions. The permeability field (shown in Figure 4) remains the same among samples. The corresponding source and velocity pairs form the samples in the training process. In this example, we have 1000 samples for training and 250 for validation. The learning rate is chosen to be 0.008. There are 100 samples in each batch during the training. For training the network which approximate the flow problem, the number of epochs is 250, and the total run time for the training is 29.2 seconds when there are 1000 samples. After training, we take new source input and predict the velocity. We present, in Tables 1 and 2, the average errors between the predicted and true velocity solutions among 250 test samples. We remark that all the network training in this work are performed using the Python deep learning API Keras [11] with TensorFlow framework.

We first compare the performance of (1) the proposed network as shown in Figure 1, denoted by locally connected network (LCN), where $N_p^H = 100$, $\alpha = 4$, $\alpha' = 8$, and the stride size is 4×4; (2) a convolutional neural network, denoted by CNN, where we replace the locally connected layers with convolutional layers and all the other hyperparameters stay the same; (3) a fully connected network, denoted by DNN, where
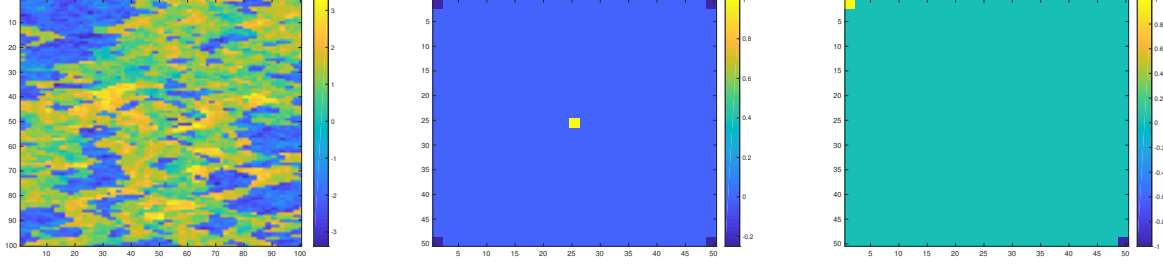
Figure 4: Left: Log scale of the permeability field (SPE10 model layer 53). Middle: Illustration of five-spot source in single phase flow problems. Right: Two-well source in two phase flow problems.

the network has the same number of layers as the LCN/CNN, and each hidden layer has 500 neurons. The mean errors among the same set of testing samples are shown in Table 1. One can observe that, the

	$\|u_{\mathrm{pred}} - u_{\mathrm{true}}\|_{L^2}$ (%)	$\|u_{\mathrm{pred}} - u_{\mathrm{true}}\|_{L_\kappa^2}$(%)	number of trainable parameters
LCN	0.6	0.7	2,536,772
CNN	0.8	1.0	2,525,708
DNN	2.1	2.2	4,346,220

Table 1: Comparison of the true velocity solution (obtained from standard solver) and predicted velocity solution (obtained from trained neural networks) for three different types of networks. Mean errors are computed among 250 testing samples. Second column: velocity solution relative error in standard L^2 norm; Third column: velocity solution relative error in weighted L^2 norm, where $\|u\|_{L_\kappa^2} = \int_\Omega \kappa^{-1}|u|^2$.

proposed network shows great potential to approximate the mapping from input source terms to output velocity solutions. Our proposed network architecture works better than the CNN (where we only change the locally connected layers in the designed network structure with standard convolutional layers), and also outperforms the DNN network. Compare the trainable parameters in the three network, LCN has slightly more parameters than that in the CNN, but has much smaller numbers than that in the DNN.

Remark 2: We compare the LCN and CNN and restrict them to the same designed architecture in Figure 1, and the numerical results show that under same architecture, replacing convolutional layers by locally connected layers can give us slightly better results. The only reason for the comparison is that, we think the locally connected layer is more suitable simply in our case. In this work, the neural network we constructed is motivated by ideas from multiscale model reduction algorithms. From this perspective, we chose locally connected layers since it has a different set of filters applied at each different patch of the input. This is the analogy to the multiscale method, where local features of the underlying heterogeneity can be extracted using multiple different basis function in different local coarse regions. The locally connected layers is a good choice and can provide sufficient learning capacity to extract rich hidden features. However, there are many well established neural networks for solving problems of the similar interest. CNNs with deeper layers were also investigated in [47] to automatically extract multiscale features from high dimensional input. Moreover, the deep convolutional encoder-decoder networks designed [49] shows very good results as surrogate models to solve stochastic PDEs. The method, on the other hand, doesn't require any explicit intermediate dimension reduction method and provide uncertainty estimates. These work provide alternative approaches and gain significant progress in surrogate modeling and uncertainty quantification.

Let the standard loss refer to the mean of relative weighted L^2 errors

$$\frac{1}{N}\sum_{i=1}^{N}\frac{\|u_{\mathrm{pred, \ i}} - u_{\mathrm{true},i}\|_2}{\|u_{\mathrm{true},i}\|_2} \tag{12}$$

where there is no additional constraint as in (6).

We next investigate LCN with the standard loss function (see (12)) and the proposed loss function (see (6)). We can observe from Table 2 that, using the proposed loss function with physical constraint, the

average relative L^2 errors are smaller than the ones using standard loss. For the last column in the table, $\overline{M_{\text{true}}^i - M_{\text{pred}}^i}$ measures the errors for the local mass, where we first compute the mean of the local mass difference between predicted solutions and true solutions among all fine cells, then the average among all testing samples. ==With the additional constraint in the loss function, the locally mass conservative property is enhanced.==

Remark 3: There are many efforts to enforce physical constraints in many deep learning tasks recently. For example, the deep fluids CNN architecture is proposed in [34] which can synthesize divergence-free fluid velocities by introducing a novel stream function based loss function. There are also physical constrained models, such as [40, 41] employ the automatic differentiation techniques to differentiate neural networks regarding the input coordinates to obey the law of physics.

	$\|u_{\text{pred}} - u_{\text{true}}\|_{L^2}$ (%)	$\|u_{\text{pred}} - u_{\text{true}}\|_{L^2_\kappa}$ (%)	$\overline{M_{\text{true}}^i - M_{\text{pred}}^i}$
Standard loss	0.4	1.0	1.64e-8
Loss with constraint	0.4	1.0	1.64e-9

Table 2: Comparison of the true velocity solution (obtained from standard solver) and predicted velocity solution (obtained from trained neural networks) using standard loss and loss with constraints. Mean errors among 250 testing samples. Second column: velocity solution relative error in standard L^2 norm; Third column: velocity solution relative error in weighted L^2 norm; Fourth column: local mass relative error in l^2 norm.

We also present the comparisons when we take a different number of nodes N_p^H as shown in the network 1. As shown in Table 3, when we use more neurons in the intermediate layers, (which corresponds to the larger number of coarse grid properties N_p^H, for example, $N_p^H = 25$ corresponds to 5×5 coarse features, and so on), the trained network produce more accurate predictions. However, as N_p^H become larger, the prediction errors decrease slower. This is due to the fact that the more complex of the network, the harder to train or the more training samples it requires. In our experiments, we use the same number of training sample in all three cases, so the results are acceptable.

N_p^H	$\|u_{\text{pred}} - u_{\text{true}}\|_{L^2}$ (%)	$\|u_{\text{pred}} - u_{\text{true}}\|_{L^2_\kappa}$ (%)	$\overline{(M_{\text{true}}^i - M_{\text{pred}}^i)}$
25	16.8	26.0	2.1e-9
100	0.4	1.0	1.64e-9
225	0.6	0.8	1.0e-9

Table 3: Comparison of the true velocity solution and predicted velocity solution using a different number (N_p^H) of neurons. Mean errors among 250 testing samples. All other hyperparameters in the network are the same. Second column: velocity solution relative error in standard L^2 norm; Third column: velocity solution relative error in weighted L^2 norm; Fourth column: local mass relative error in l^2 norm.

Remark 4: The last layer performs a downscaling step from coarse grid velocity to fine grid velocity solution. One can replace it with a sparse layer, where the weight matrix is sparse and takes into account the local effects. Designing the sparse layer requires the information about coarse-to-fine degrees of freedom map. If the dimensions of the fine/coarse scale space change, then one may need to redesign the last layer's sparse connection. We did both use dense layer and sparse layer in the last layer, and get similar results. For example, when $N_p^H = 100$ and sparse connections are used in the last layer, we got the mean error 0.7% (compared to 0.4% when dense connections are used in the last layer) in standard L^2 norm, and 0.3% (compared to 1.0% when dense connections are used in the last layer) in weighted L^2 norm.

In the end, we remark on the efficiency of the neural network. One fine solve of the problem (2) using the Matlab direct solver takes 0.05 seconds, and a prediction step using the trained network takes only 0.001 second. We remark in this work, all experiments in Matlab are done on Intel(R) Xeon(R) CPU E5-1650, and experiments in tensorflow are done on GeForce GTX 1080 Ti.

2.3.2 Numerical test on neural network approximation of the saturation equation

In this section, we show a numerical example using the proposed neural network to learn the saturation dynamics. We still use a specific five spot configuration for the source term, and first obtain the velocity solution as described in Section 2.1. Next, the transport equation for saturation is solved using the finite volume scheme (7) given initial condition S^0. We choose the time step size $\triangle t = 1$, and solve the problem 1200 time steps to obtain a series of saturation solution on the fine grids, i.e. $[S^1, S^2, \cdots, S^{1200}]$.

In order to train the dynamics between S^n and S^{n+1}, we choose the solution pairs (S^i, S^{i+1}), $i = 1, \cdots, N_{\text{train}}$ as training samples, and (S^i, S^{i+1}), $i = N_{\text{train}} + 1, \cdots, 1200$ for validation. Then we use $S^{N_{\text{train}}+1}$ as a test sample for prediction. As mentioned in the previous section, once the feedforward map is trained, we can use it multiple times to predict solutions at later times. Our goal in this example, is to apply the trained network $\mathcal{M}$ for $1200 - N_{\text{train}}$ times, and compare the prediction with the true solution S^{1200}. During the training, the number of epochs is 500, and the total run time for the training is 54 seconds when there are 1000 samples.

We will first take $N_{\text{train}} = 1000$, and test the performance of (a) the proposed network with the introduced Sparse Velocity Layer as in (10), (b) Sparse Connected Layer, where the weight matrices to be trained only have the same sparsity structure as in (9), but are not multiplied by velocity, (c) densely connected layers. The results are presented in Table 4. We remark that, the other hyperparameters, such as batch size, learning rate, training epochs, etc, in (a)-(c) are chosen as the same. The errors between the predicted solution S_{pred} and true solution S_{true} are measured by

$$E_s = \frac{\|S_{\text{pred}} - S_{\text{true}}\|_{L^2}}{\|S_{\text{true}}\|_{L^2}}.$$

	Sparse velocity layer	Sparsely connected layer	Densely connected layer
$E_s(1)$ (%)	0.008	0.02	2.67
$E_s(10)$ (%)	0.08	0.21	9.56
$E_s(100)$ (%)	0.91	1.85	blow up
$E_s(199)$ (%)	2.20	4.16	blow up
Trainable paras #	59,200	59,200	6,252,500

Table 4: $E_s(n)$ denotes the relative error after n time steps. Errors between true (obtained from standard solver) and predicted saturation (obtained from trained neural network), single phase flow. Second column: using proposed Sparse Velocity Layer; Third Column: using Sparsely Connected Layer; Fourth Column: using Densely Connected Layer.

From Table 4, one can see that the number of trainable parameters is relatively small compared with a densely connected network using our proposed network. Moreover, the relative error of the predicted solution (the second column in the table) is very small even after 199 time steps. However, using just sparsely connected layers without velocity information, the network prediction becomes a bit worse after some time (the third column in the table). With densely connected layers, the network works produce unreliable results (the fourth column in the table), this is due to a large number of the trainable parameters are hard to be trained effectively considering the limited number of training samples. What's more, this densely connected network didn't take into account the velocity field either. A comparison of the predicted solution using the proposed method and true solution in the long range is presented in Figure 5, where we note the accuracy of the predicted solution.

Furthermore, we also test the performance of the proposed network by taking a different number of samples in the training set. The results are shown in Table 5. It can be observed that our proposed method is quite robust. With more training samples, the network can be trained better. As we increase the predicted time steps, the errors grow reasonably.

In the end, we just want to mention that, in this single phase setting, using our proposed neural network 199 times to predict the final solution takes only 0.466 s. Solving the saturation equation using a direct solver in Matlab 199 times takes 1.120s.
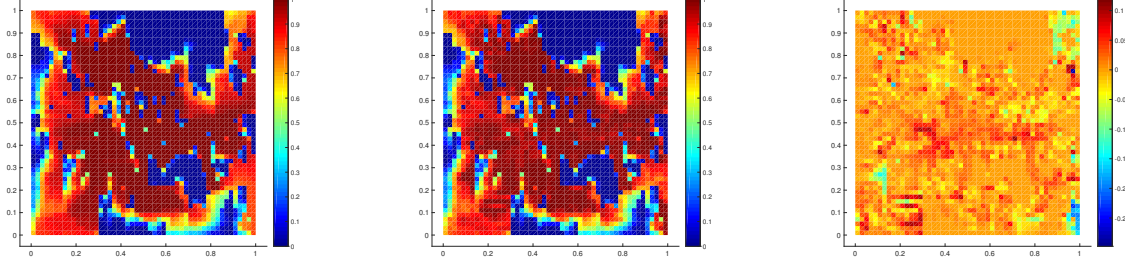
Figure 5: Given solution at time step 1001, use trained neural network to predict iteratively 199 times. Comparison of saturation: left: true solution at time step 1200, middle: predicted solution at time step 1200, right: difference between the true and predicted solutions.

samples # \ # of time steps	199	399	599
1000	2.20		
800	2.65	9.64	
600	5.11	17.7	68.4

Table 5: With different numbers of training samples and different predicted time steps, errors between predicted saturation (obtained from trained neural network) and true saturation (obtained from standard solver).

3 Two phase flow

Now, we consider incompressible two-phase flow. The gravity effects and capillary pressure will be neglected in the model. The flow equation reads

$$u = -\lambda(S)\kappa\nabla p \qquad \text{in} \quad D \tag{13}$$

$$\text{div}(u) = r \qquad \text{in} \quad D \tag{14}$$

$$u \cdot n = 0 \qquad \text{on} \quad \partial D \tag{15}$$

where κ is the absolute permeability and,

$$\lambda(S) = \frac{\kappa_{rw}(S)}{\mu_w} + \frac{\kappa_{ro}(S)}{\mu_o}$$

is the total mobility, which depends on the saturation states. κ_{rw}, κ_{ro} are the relative permeability, μ_w and μ_o are the viscosity, for water (denoted by subscript w) and oil (denoted by subscript o), respectively. Notice that, κ_{rw}, κ_{ro} are nonlinear functions of S.

The saturation equation of S_w for the water phase is given by

$$\frac{\partial S_w}{\partial t} + u \cdot \nabla f_w(S_w) = r$$

where $f(S_w) = \dfrac{\kappa_{rw}(S_w)/\mu_w}{\kappa_{rw}(S_w)/\mu_w + \kappa_{ro}(S_w)/\mu_o}$. We simplify the notation and use S for the saturation S_w from now on.

The transport equation can also be solved using the finite volume method, and the time derivative is discretized using backward Euler. On a fine grid K_i, the value S_i at time t^{n+1} can be obtained by

$$S_i^{n+1} = S_i^n + \frac{\text{d}t}{|K_i|}\Big[-\sum_{e_j \in \partial K_i} F_{ij}(S^{n+1}) + f_w(S^{n+1})r_i^- + r_i^+\Big] \tag{16}$$

12

where $r_i^- = \min(0, r_i)$, $r_i^+ = \max(0, r_i)$. And F_{ij} is the upstream flux, i.e.

$$F_{ij}(S^{n+1}) = \begin{cases} \int_{e_j} (u_{ij}^{n+1} \cdot n) f_w(S_i^{n+1}) & \text{if} \quad u_{ij}^{n+1} \cdot n \geq 0 \\ \int_{e_j} (u_{ij}^{n+1} \cdot n) f_w(S_j^{n+1}) & \text{if} \quad u_{ij}^{n+1} \cdot n < 0 \end{cases} \tag{17}$$

Here e_j denotes the edge between fine grid K_i and K_j, u_{ij} the velocity on the edge e_j.

For the two-phase problem, the flow and transport equations are solved sequentially follow the Algorithm 1.

Algorithm 1 Sequentially solve flow and transport problem

1: **procedure** TWO PHASE($S^0, \mathrm{d}t, T, t = 0$)   ▷ S^0: initial saturation, $\mathrm{d}t$: time step, T: total time
2:     $S \leftarrow S^0$
3:     **while** $t < T$ **do**
4:         $\lambda(S^0) \leftarrow \lambda(S)$
5:         Solve u from (13) using $\lambda(S^0)$
6:         Solve S from (16) using Newton-Ralphson method, S^0 and u
7:         $S^0 \leftarrow S$
8:         $t \leftarrow t + \mathrm{d}t$
9:     **return** S

One can observe from Algorithm 1 that, step 6 requires Newton-Ralphson method which converges with a number of iterations. Also, a flow problem and a nonlinear transport problem are both solved at each time step. These will result in a heavy computational burden in practical problems. Our goal in this section is to construct powerful deep neural networks based on the experience in single-phase flow, to effectively approximate and enhance the computational efficiency of the coupled flow and transport solver.

3.1 Flow velocity learning in two phase flow

In two-phase flow case, the source function r is chosen to be a piecewise constant function. The source functions take nonzero values only at two fine cells, and zero values elsewhere. One of the two nonzero blocks is the fine cell at the upper left corner of the computational domain, the other is at the lower right corner of the computational domain. Considering compatibility, we assume one block has value $-Q$ acting as an injection well, the other has value Q acting as a production well. The absolute permeability is highly heterogeneous and remains unchanged during the simulation, shown in Figure 4.

Given source terms and initial conditions, suppose we have solved the coupled flow and transport problem use Algorithm 1 for n times with time step $\triangle t$. Let $\{r_1, \cdots, r_{n_s}\}$ be a set of source functions, where n_s denotes the number of source samples. For every source function r_j $(j = 1, \cdots, n_s)$, we can obtain the velocity solutions $[u_j^1, \cdots u_j^n]$, and the saturation solutions are $[S_j^1, \cdots S_j^n]$. We will use the velocity and saturation solutions as samples to train neural networks. And we will omit the subscript j for the simplification of notations. The design of neural networks in two-phase flow are based on the proposed methods in the single-phase flow, but some necessary modifications are made to handle the differences.

3.1.1 Network Architecture

For the two-phase system, we are interested in the map between the total mobility $\{\lambda(S), r\}$ and velocity field u in the flow problem. As we can see from Algorithm 1, in the coupled flow and transport system, the velocity fields are updated at each time step. The velocity updates come from the mobility change, and the mobility $\lambda(S)$ depends on the saturation solution from the previous time step. We remark that the absolute permeability κ won't change during the simulation. However, the variation of S changes the relative permeability.

Let $\left(\{\lambda(S^i), r^i\}, u^i\right)$ $(i = 1, \cdots n_t)$ be the training samples, where $\{\lambda(S^i), r\}$ is the input, and u^i is the output, n_t is the number of samples. We will follow the similar idea as in Figure 1 to construct the neural network $\mathcal{N}_2$. However, there are some main differences we need to take care of. First, we have now both the mobility $\lambda(S)$ and f as inputs. Compared with the single phase flow case, where we input r as a single

channel image, now we will place $\lambda(S)$ and r in two channels of an image separately and use it as input. Second, we will add a few convolutional layers before the first average pooling layer. This is due to the complexity in $(\lambda(S^i), r^i)$ compared with the only source term r. The additional convolutional layers will extract important hidden features. After that, we will reduce the dimension by an average pooling layer. Then similar strategies are applied as described in the single-phase case 1.

3.1.2 Numerical example for flow map in two-phase flow

In the following, we present the results for two-phase flow velocity learning. Given a set of different source terms $\{r_1, \cdots, r_{n_s}\}$, the coupled flow and transport system are solved $n = 1000$ times with time step $\triangle t = 0.2$ for each source. Omitting the subscripts, we have $\left((\lambda(S^i), r^i), u^i\right)$ $(i = 1, \cdots 1000 \times n_s)$ as training samples. After training, given a new source term r_{new}, we will take $\lambda(S^i_{\text{new}}), r^i_{\text{new}})$ $(i = 1, \cdots, 1000)$ are used for validation/prediction.

In this example, the number of trainable parameters in this neural network is $2,625,236$. The number of epochs in the training is 1000, and the total run time is 295 seconds when there are 1000 samples. A prediction step of the trained neural network takes around 0.003 seconds. Similar as in the single phase, local mass conservation is an important feature of the numerical velocity solution obtained from MFEM. This property ensures its adequacy in the transport simulations. We would like the solutions predicted by the proposed neural network also preserve this property. The constraint loss function is employed. By comparing the performance of the network when we use the constraint loss function (6) and (12), we notice that with constraint loss function, not only the local mass conservative property of the predicted solution is enhanced, the accuracy of the predicted solution in both L^2 and weighted L^2 norm is also improved. The results are shown in Table 6.

Errors	$\|u_{\text{pred}} - u_{\text{true}}\|_{L^2}$ (%)	$\|u_{\text{pred}} - u_{\text{true}}\|_{L^2_\kappa}$ (%)	$\overline{(M^i_{\text{true}} - M^i_{\text{pred}})}$
Loss with constraint	0.7	1.3	1.2e-8
Standard loss	0.8	1.9	2.2e-8

Table 6: With different loss functions, mean errors between the true velocity solution and predicted velocity solution in two phase flow. Second column: velocity solution relative error in standard L^2 norm; Third column: velocity solution relative error in weighted L^2 norm; Fourth column: local mass relative error in l^2 norm.

3.2 Saturation dynamics learning in two phase flow

As we see from the saturation equation, (16) can be written in the following matrix equation

$$S^{n+1} = S^n + \mathrm{d}t(\tilde{F}(u^{n+1}, S^{n+1}) + R) \tag{18}$$

where $\tilde{F}$ is the upwind flux, nolinearly depends on the saturation as described in (17). Due to the nonlinearity, (18) needs to be solved using an iterative method. This makes it much more computationally expensive compared with the single flow problem.

To improve efficiency, a residual network $\mathcal{M}_2$ is expected to learn the dynamics of the saturation equation. Similar as before, the feed-forward map we are interested in is still from S^n to S^{n+1}. The main difference in the two-phase flow lies in that, the velocity field driving the convection is time-dependent. During the learning, the velocity fields are no longer universal. We need to adjust the previously proposed network in Section 2.2 to handle these issues.

3.2.1 Network structure

We recall that, in the single-phase learning, we introduce the Sparse Velocity Layer to take into account the velocity impact on the transport problem, where a steady velocity field can be multiplied to the weight tensor element wisely in the layer construction. Different from the single-phase case, the velocity field is now changing at each time step. Thus, we need to take velocity fields corresponding to the saturation solution at

each time step together as input to the network. However, it will introduce too many trainable parameters if we exactly use the velocity field corresponding to the saturation for each sample in the Sparse Velocity Layer. One workaround is to take the average velocity of each batch and use their mean in the Sparse Velocity Layer. Please see Figure 6 as an illustration. Moreover, since the map we approximate is nonlinear, before the final residual add layer, we also add a few dense layers with nonlinear activation functions to capture the nonlinearity in the map we approximate.
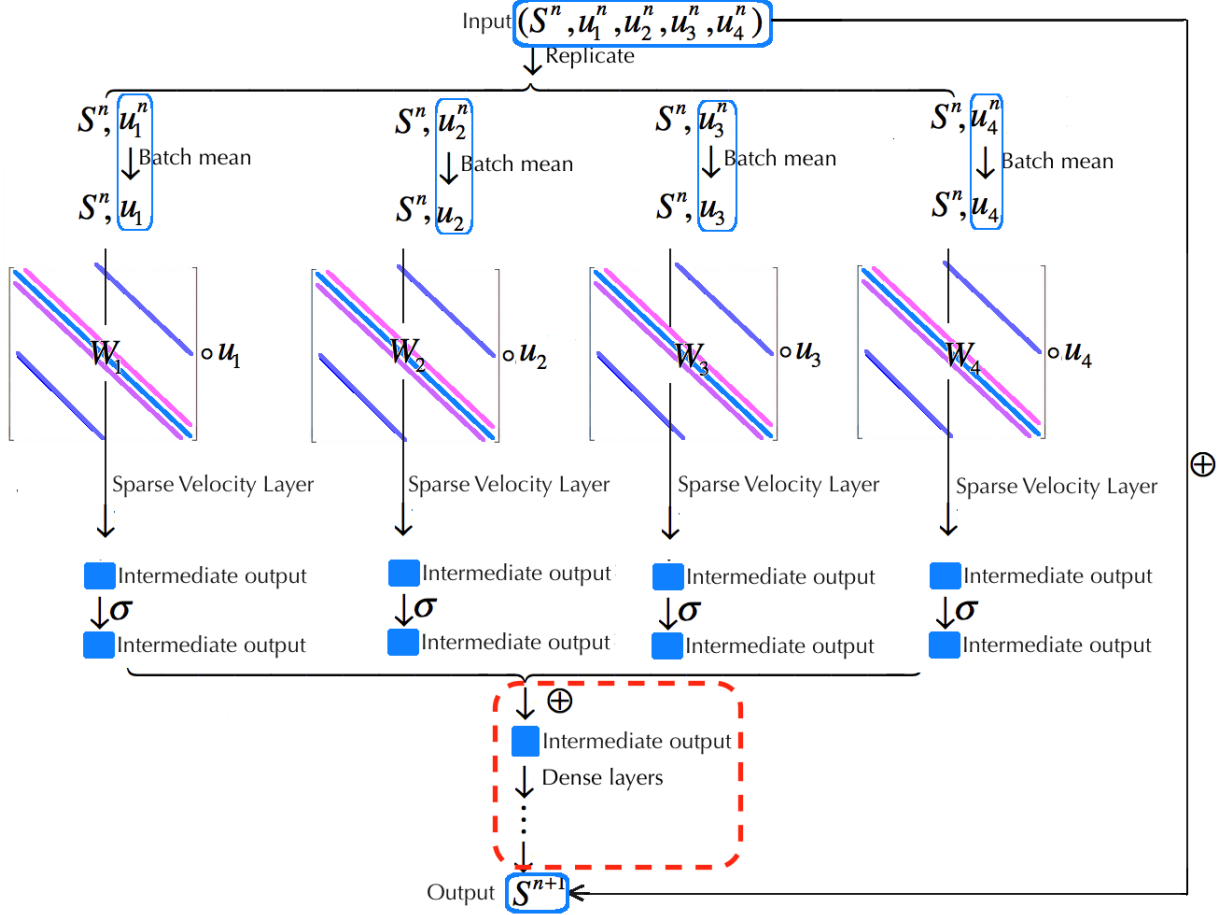
Figure 6: An illustration of the neural network architecture for learning saturation problem in two phase flow.

3.2.2 Numerical example of saturation learning in two-phase flow

Here, we apply the proposed neural network shown in Figure 6 to learn the saturation dynamics in two-phase flow.

For a given set of source terms $\{r_1, \cdots, r_{n_s}\}$, we take the velocity and saturation solutions obtained from the coupled flow and transport solver, i.e. $[u_j^1, \cdots u_j^{1000}]$, and $[S_j^1, \cdots, S_j^{1000}]$ ($j = 1, \cdots, n_s$). We then separate the velocity in four directions, $u^i \rightarrow (u_1^i, u_2^i, u_3^i, u_4^i)$. Abandon the subscript, we choose the states $(S^i, u_1^i, u_2^i, u_3^i, u_4^i, f^i)$ as training inputs, and S^{i+1} as training outputs, for $i = 1, \cdots, 1000 \times n_s$. In the end, the velocity and saturation solutions at all time steps associated with a new source term $r_{\text{new}} \notin \{r_1, \cdots, r_{n_s}\}$, are used for validation/prediction.

In this example, for training, we choose $n_s = 5$, and $r_j = j$ at the injection well, for $j = 1, \cdots n_s$. Here r_j is a constant for all time steps. As for validation, we generate a random source as follows. First, the values of the source term is changing at four random time instants between 1 and 1000. Moreover, in each

sub-interval, the values of the injection rate is also randomly chosen between 1 and 5. For example, we show a test source term in the left side of Figure 7. The corresponding water cuts for training source and testing source are shown in the right side of Figure 7. For training the network in the two phase flow, the number of epochs is 500, and the total run time for the training is around 588 seconds.
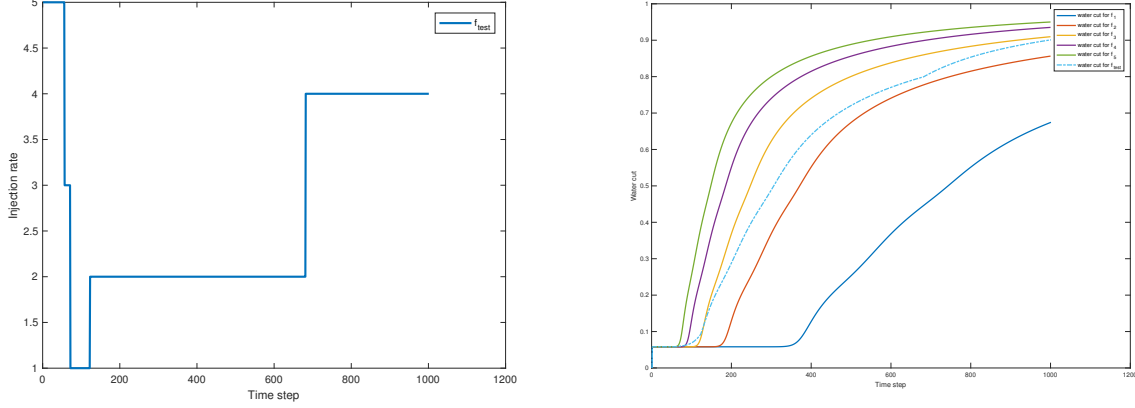

Figure 7: Left: a random source r_{test} for generating test samples. Right: the water cuts at all time steps, for training sources r_j $(j = 1, \cdots, n_s = 5)$, and r_{test}.

We remark that the batch size in this proposed network may affect the accuracy of the trained network since we take the mean of velocity solution among each batch in the Sparse Velocity Layer. We will test the performance of the proposed network (as shown in Figure 6) with different batch sizes.

Batch size	Mean error E_s over 1000 test cases (%)
10	0.036
100	0.039
200	0.064

Table 7: Two phase flow: For different batch size, mean errors between the true saturation solutions at time $n+1$ and predicted saturation solutions at time $n+1$ (predicted from saturation at time n). The test source term is shown in the left side of Figure 7.

Our numerical tests show that the errors are not increasing too much when the batch sizes become large. One reason for this may be the variations in the velocity field are not that dramatic. Although the mobility $\lambda(S)$ in each time step is different, the absolute permeability κ still have a strong impact on the velocity solution. But the absolute permeability stays the same throughout the simulation in our experiment.

3.3   Coupled flow and transport prediction using trained neural networks

We have obtained two surrogate models using deep learning, namely, $\mathcal{N}_2$ for the velocity, and $\mathcal{M}_2$ for the saturation equation in two-phase flow case. They have shown their efficiency and accuracy separately in the previous sections.

Similar as the idea in the single phase flow case, once the feed-forward map for saturation equation is trained, we can use it multiple times to predict solutions at later times. However, in the two phase flow case, we also need to feed in velocity field at each prediction step. That is not a problem for us, since the velocity field can be obtained using the network $\mathcal{N}_2$ we trained in Section 3.1. That is, we will use both of the trained network $\mathcal{N}_2$ and $\mathcal{M}_2$ iteratively to predict future solutions. The algorithm is as follows:

Algorithm 2 Deep learning for flow and transport problem
___

1: **procedure** Two Phase(S^n, $\mathcal{N}_2$, $\mathcal{M}_2$, m) ▷ S^n: saturation at time step n, N_u : network for velocity, N_s : network for saturation, m: number of time steps for prediction
2:      $S \leftarrow S^n$
3:      **while** $i < m$ **do**
4:          $\lambda(S^n) \leftarrow \lambda(S)$
5:          Predict u^n using $\mathcal{N}_2$ with input $\lambda(S^n)$
6:          Predict S^{n+1} using $\mathcal{M}_2$, with input S^n, u^n
7:          $S \leftarrow S^{n+1}$
8:          $i \leftarrow i + 1$
9:      **return** S
___

3.3.1 Numerical example for applying trained networks to predict coupled system in the long range

Our last numerical example shows the power of the trained networks in two-phase flow. For a new source term r_{test} (shown in left side of Figure 7)), we only need to take the solution S^1 as an input to the trained neural network. Then we will apply the networks multiple times to predict the saturation solution in the long range according to Algorithm 2. Our goal in this example is to apply Algorithm 2 with trained networks $\mathcal{N}_2$ and $\mathcal{M}_2$ with $m = 999$ as the number of predicted time steps, to predict the saturation S^{1000}. Then we will compare the predictions with true solutions obtained from Algorithm 1. From Table 8, we observe that, the predictions are very close to the true solution. We have tested over many different cases of r_{test} and got similar results, here we just take a few for illustration.

A visualization of the saturation solutions at some predicted time steps is presented in Figure 8, where the source term is the one shown in the left side of Figure 7. We observe that the predicted solutions show a good match towards the true solution. This shows the accuracy of our trained network. In the end, we remark that 999 steps prediction takes only 2.25s using the trained networks while solving the coupled system 999 times takes about 94s. In the end, for three different source terms (shown in left side of Figure 9), we repeated the prediction as before. Then we also compute their corresponding water cuts based on the true saturation and the predicted saturation. The comparison is presented in Figure 9. We observe very good match of the predicted and true results.

Numbers of predicted time steps	$\|S_{\text{pred}} - S_{\text{true}}\|_{L^2}$ (%)
200	5.37
400	4.45
600	4.55
800	5.42
1000	6.88

Table 8: Two phase flow: Given an initial state, apply trained neural networks iteratively. Errors between true and predicted saturation at different numbers of time steps.

One last thing to mention is about the computational time. We list the information in Table 9.

Using Algorithm 1	Using Algorithm 2
94.36 seconds	2.25 seconds

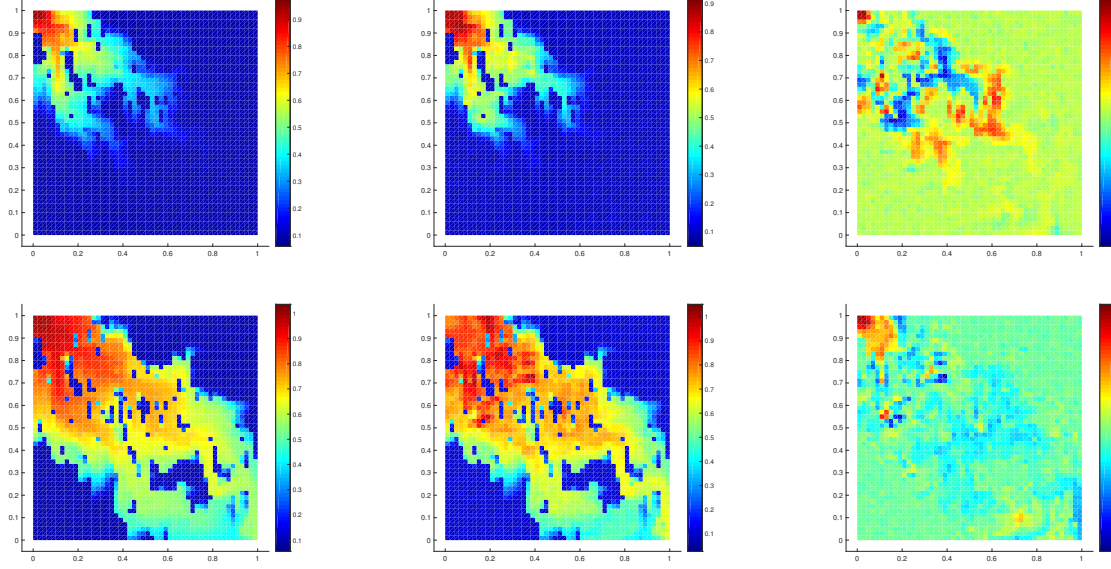Table 9: Comparison of computational time using Algorithm 2 and Algorithm 1.

Figure 8: Comparison of saturation using Algorithm 2. Given the initial solution, top row: after 50 prediction step, bottom row: after 950 predicted time steps. Left column: true solution, middle column: predicted solution, right column: difference between the true and predicted solutions. After 50 prediction step, the relative L^2 error between predicted and true solution is 7.0%. After 199 predicted time steps, the relative L^2 error between predicted and true solution is 6.5%

4 Conclusion

In this work, we propose efficient deep neural networks as surrogate models to approximate flow and transport equations. Both single phase and two-phase flow problems are considered. The designed deep neural networks honor the sparsity structures of the underlying discrete systems, thus having a significant reduction in the number of trainable parameters compared with fully connected neural networks. To be specific, for the flow problem, we apply deep learning to approximate the map from the source terms (in the single-phase flow case)/ relative permeability fields (in the two-phase flow case) to the velocity solution. The networks are constructed with convolutional and locally connected layers to perform model reductions and are equipped with a custom loss function to impose local mass conservation constraints. The custom loss function helps to maintain the physical property of the velocity solution. Once the velocity fields are obtained, they will be inputs to the saturation equation and drive the transport process. Incorporating the learned velocity fields, as well as the upstream scheme of discrete saturation equation, a residual type of network, is introduced to approximate the dynamics of the saturation. That is, we design custom sparsely connected layers which take into account the inherent sparse interaction between the input (saturation at a previous time instant, as well the velocity fields) and output (saturation at the next time instant). Once the feed-forward map between the solution at two consecutive time steps is trained, the approximated map can be used iteratively many times to predict solutions in the long future. An efficient algorithm is proposed to solve the coupled flow and saturation system using the trained neural networks. Considering the saturation map in two-phase flow is nonlinear, the constructed neural networks show great improvement in computational efficiency. The predicted solutions in our numerical examples also present excellent accuracy. Future work involves extending the method in conjunction with multiscale model reduction algorithms and developing more efficient deep learning tools to solve larger coupled flow and transport systems.
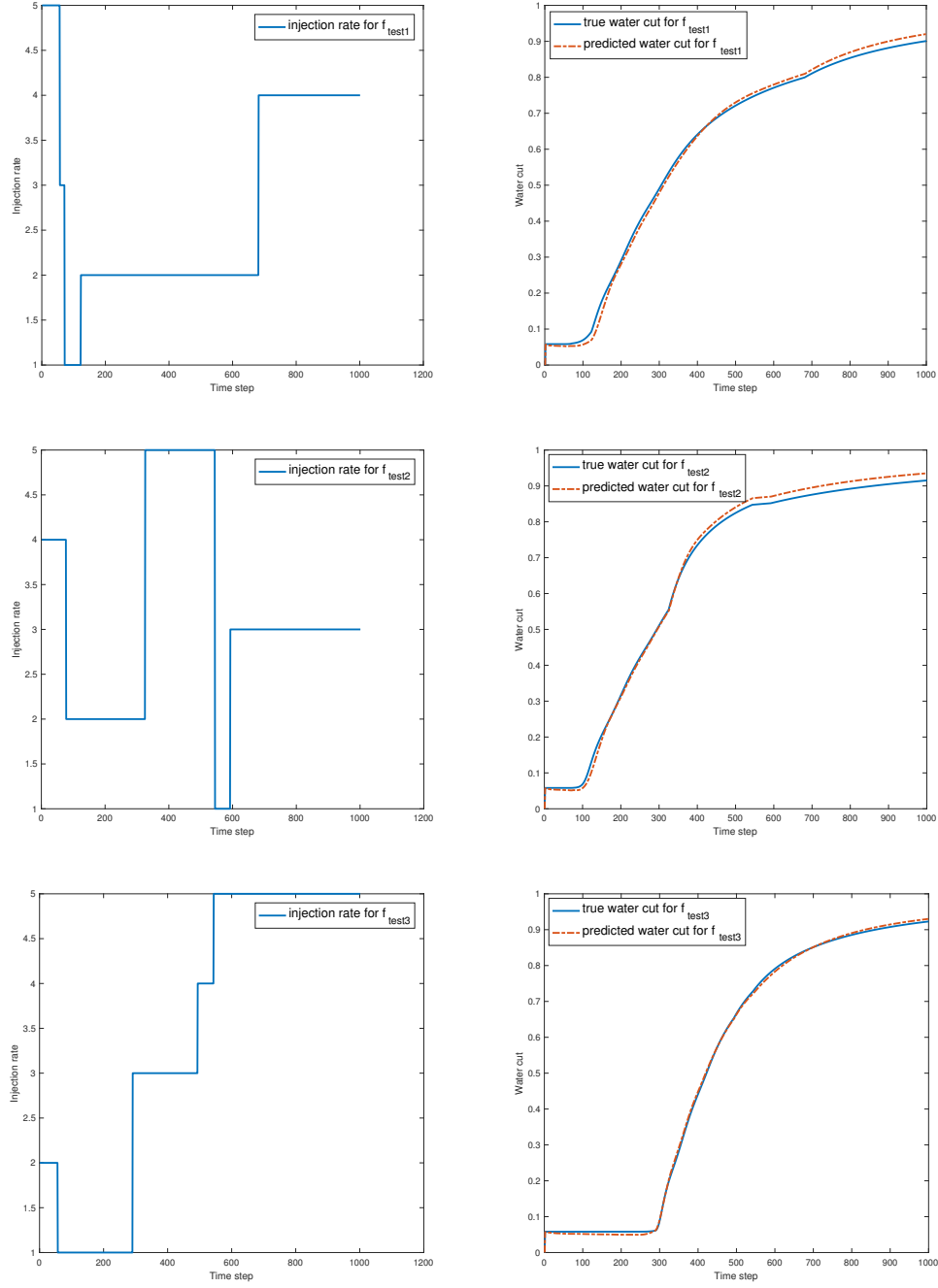
Figure 9: For three different source terms, the comparison of water cuts obtained from the true saturation and the predicted saturation.

Acknowledgements

References

[1] J. Aarnes and Y. Efendiev, *Mixed multiscale finite element for stochastic porous media flows*, SIAM J. Sci. Comput., 30 (5) (2008), pp. 2319–2339.

[2] A. Abdulle and Y. Bai, *Adaptive reduced basis finite element heterogeneous multiscale method*, Comput. Methods Appl. Mech. Engrg., 257 (2013), pp. 203–220.

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

[4] M. Alotaibi, V. M. Calo, Y. Efendiev, J. Galvis, and M. Ghommem, *Globallocal nonlinear model reduction for flows in heterogeneous porous media*, Computer Methods in Applied Mechanics and Engineering, 292 (2015), pp. 122–137.

[5] T. Arbogast, *Implementation of a locally conservative numerical subgrid upscaling scheme for two-phase Darcy flow*, Comput. Geosci, 6 (2002), pp. 453–481.

[6] T. Arbogast, *Homogenization-based mixed multiscale finite elements for problems with anisotropy*, Multiscale Model. Simul., 9 (2011), pp. 624–653.

[7] D. L. Brown and D. Peterseim, *A multiscale method for porous microstructures*, arXiv preprint arXiv:1411.1944, (2014).

[8] V. Calo, Y. Efendiev, J. Galvis, and M. Ghommem, *Multiscale empirical interpolation for solving nonlinear pdes using generalized multiscale finite element methods*. Submitted.

[9] R. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud, *Neural ordinary differential equations*, arXiv preprint arXiv:1806.07366, (2018).

[10] S. W. Cheung, E. T. Chung, Y. Efendiev, E. Gildin, and Y. Wang, *Deep global model reduction learning*, arXiv preprint arXiv:1807.09335.

[11] F. Chollet et al., *Keras.* https://keras.io, 2015.

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

[13] E. Chung, Y. Efendiev, and C. Lee, *Mixed generalized multiscale finite element methods and applications*, SIAM Multicale Model. Simul., 13 (2014), pp. 338–366.

[14] E. Chung, Y. Efendiev, and W. T. Leung, *Generalized multiscale finite element method for wave propagation in heterogeneous media*, SIAM Multicale Model. Simul., 12 (2014), pp. 1691–1721.

[15] 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 (2013), pp. 370–392.

[16] E. Chung, M. Vasilyeva, and Y. Wang, *A conservative local multiscale model reduction technique for stokes flows in heterogeneous perforated domains*, Journal of Computational and Applied Mathematics, 321 (2017), pp. 389–405.

[17] E. T. Chung, Efendiev, W. T. Leung, M. Vasilyeva, and Y. Wang, *Non-local multi-continua upscaling for flows in heterogeneous fractured media*, arXiv preprint arXiv:1708.08379, (2018).

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

[19] E. T. Chung, Y. Efendiev, and G. Li, *An adaptive GMsFEM for high contrast flow problems*, J. Comput. Phys., 273 (2014), pp. 54–76.

[20] B. C. Csáji, *Approximation with artificial neural networks*, Faculty of Sciences, Etvs Lornd University, 24 (2001).

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

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

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

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

[25] Y. Efendiev, T. Hou, and V. Ginting, *Multiscale finite element methods for nonlinear problems and their applications*, Comm. Math. Sci., 2 (2004), pp. 553–589.

[26] Y. Fan, L. Lin, L. Ying, and L. Zepeda-Nnez, *A multiscale neural network based on hierarchical matrices*, arXiv preprint arXiv:1807.01883, (2018).

[27] J. Fish and W. Chen, *Space–time multiscale model for wave propagation in heterogeneous media*, Computer Methods in applied mechanics and engineering, 193 (2004), pp. 4837–4856.

[28] J. Fish and R. Fan, *Mathematical homogenization of nonperiodic heterogeneous media subjected to large deformation transient loading*, International Journal for numerical methods in engineering, 76 (2008), pp. 1044–1064.

[29] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, Proceedings of the IEEE conference on computer vision and pattern recognition, (2016), pp. 770–778.

[30] P. Henning and M. Ohlberger, *The heterogeneous multiscale finite element method for elliptic homogenization problems in perforated domains*, Numerische Mathematik, 113 (2009), pp. 601–629.

[31] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. rahman Mohamed, N. Jaitly, and A. Senior, *Approximation capabilities of multilayer feedforward networks*, IEEE Signal Processing Magazine, 29 (2012), pp. 82–97.

[32] K. Hornik, *Approximation capabilities of multilayer feedforward networks*, Neural Networks, 4 (1991), p. 251257.

[33] Y. Khoo, J. Lu, and L. Ying, *Solving parametric pde problems with artificial neural networks*, arXiv:1707.03351, (2017).

[34] B. Kim, V. C. Azevedo, N. Thuerey, T. Kim, M. Gross, , and B. Solenthaler, *Deep fluids: A generative network for parameterized fluid simulations*, In Computer Graphics Forum, 38 (2019), pp. 59–70.

[35] A. Krizhevsky, I. Sutskever, and G. E. Hinton, *Imagenet classification with deep convolutional neural networks*, Advances in neural information processing systems, (2012), pp. 1097–1105.

[36] A.-M. Matache and C. Schwab, *Two-scale fem for homogenization problems*, ESAIM: Mathematical Modelling and Numerical Analysis, 36 (2002), pp. 537–572.

[37] H. Owhadi and L. Zhang, *Metric-based upscaling*, Comm. Pure. Appl. Math., 60 (2007), pp. 675–723.

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

[39] T. Qin, K. Wu, and D. Xiu, *Data driven governing equations approximation using deep neural networks*, arXiv preprint arXiv:1811.05537, (2018).

[40] M. Raissi, P. Perdikaris, and G. E. Karniadakis, *Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations*, arXiv preprint arXiv:1711.10561, (2017).

[41] ——, *Physics informed deep learning (part ii): Data-driven discovery of nonlinear partial differential equations*, arXiv preprint arXiv:1711.10566, (2017).

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

[43] M. Wang, S. W. Cheung, E. T. Chung, Y. Efendiev, W. T. Leung, and Y. Wang, *Prediction of discretization of gmsfem using deep learning*, arXiv preprint arXiv:1810.12245, (2018).

[44] Y. Wang, S. W. Cheung, E. T. Chung, Y. Efendiev, and M. Wang, *Deep multiscale model learning*, 2018.

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

[46] Y. Yang, M. Ghasemi, E. Gildin, Y. Efendiev, and V. Calo., *Fast multiscale reservoir simulations with pod-deim model reduction*, SPE Journal, 21 (2016), pp. 2–141.

[47] M. D. Zeiler and R. Fergus, *Visualizing and understanding convolutional networks*, In European conference on computer visio, (2014), pp. 818–833.

[48] J. Zhang, S. W. Cheung, Y. Efendiev, E. Gildin, and E. T. Chung, *Deep model reduction-model learning for reservoir simulation*, In SPE Reservoir Simulation Conference. Society of Petroleum Engineers, (2019).

[49] Y. Zhu and N. Zabaras, *Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification*, Journal of Computational Physics, 366 (2018), pp. 415–447.

[50] Y. Zhu, N. Zabaras, P.-S. Koutsourelakis, and P. Perdikaris, *Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data*, Journal of Computational Physics, 394 (2019), pp. 56–81.