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

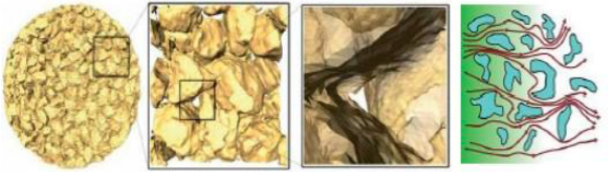
Yating Wang, Guang Lin

Department of Mathematics, Purdue University

Outline

Motivation and Objective

- Multiscale phenomena extensively exist in flow problems in porous media



- Recover information in all scales requires very fine mesh, resulting in large number of dofs in the discrete system.
- Large scale system of nonlinear equations need to be solved at many time steps.

Objective: Design Deep Neural Networks as surrogates to reproduce solutions of flow problems over multiple inputs at reduced computational cost.

Single phase problem

Single phase flow equation

Flow equation:

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

κ: heterogeneous permeability field.
Fine scale problem, solve for $(u_h, p_h) \in V_h \times Q_h$ using mixed finite element method:

$$\begin{aligned} a(u_h, v) + b(v, p_h) &= 0 && \forall v \in V_h \\ b(u_h, q) &= -(f, q) && \forall q \in Q_h \end{aligned}$$

where $a(u, v) = \int_\Omega \kappa^{-1} u \cdot v$, and $b(v, p) = -\int_\Omega p \, \mathrm{div} v$.
Velocity: RT_0 element. Pressure: P_0 element.

Matrix formulation:

$$\begin{bmatrix} A_h(\kappa) & B_h^T \\ B_h & 0 \end{bmatrix} \begin{bmatrix} u_h \\ p_h \end{bmatrix} = \begin{bmatrix} 0 \\ -F \end{bmatrix} \tag{1}$$

<u>Difficulties</u>: $A_h(\kappa)$ is large, and depends on κ.
Model reduction: Find $(u_H, p_H) \in V_H \times Q_H$ satisfying

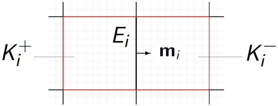
$$\begin{aligned} a(u_H, v) + b(v, p) &= 0 & \forall v \in V_H \\ b(u_H, q) &= -(f, q) & \forall q \in Q_H \end{aligned}$$

where

$$V_H \subseteq V_h \subseteq H_0(\mathrm{div}, \Omega), \quad Q_H \subseteq Q_h \subseteq L_0^2(\Omega)$$

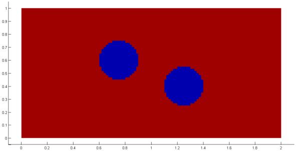
Mixed Multiscale Finite Element Method: Chen, Hou

In $\omega_i = K_i^+ \cup K_i^-$ associated with an coarse edge E_i, solve for one multiscale basis w.r.t a coarse edge:

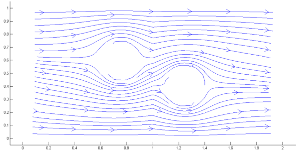


$$
\begin{aligned}
\kappa^{-1}\phi_i + \nabla p_i &= 0 \quad &\text{in} \quad \omega_i\\
\operatorname{div}(\phi_i) &= \alpha_i \quad &\text{in} \quad \omega_i\\
\phi_i \cdot n &= 0 \quad &\text{on} \quad \partial\omega_i\\
\phi_i \cdot m_i &= 1/|E_i| \quad &\text{on} \quad E_i
\end{aligned}
$$

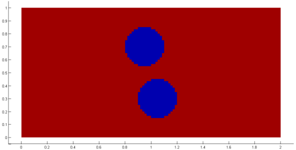
κ_1^{local}



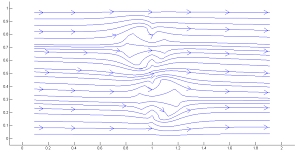
ϕ on κ_1^{local}



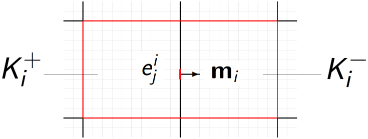
κ_2^{local}



ϕ on κ_2^{local}



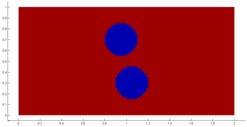
Generalized Mixed Multiscale Finite Element Method: Chung, Efendiev, Lee.



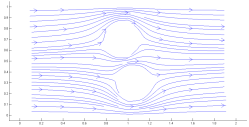
$$\begin{aligned} \kappa^{-1}\phi_i^j + \nabla p_i &= 0 \quad \text{in} \quad \omega_i \\ \operatorname{div}(\phi_i^j) &= \alpha_i \quad \text{in} \quad \omega_i \\ \phi_i^j \cdot n &= 0 \quad \text{on} \quad \partial\omega_i \text{ and } E_i \backslash e_j^i \\ \phi_i^j \cdot m_i &= 1/|e_j^i| \quad \text{on} \quad e_j^i \end{aligned}$$

$V_{\text{snap}}^i = \{\phi_i^j,\ \ j = 1, \cdots, J_i\}$. Then perform spectral decomposition in V_{snap}^i to get dominant modes and construct V_{off}^i. Resulting in several bases w.r.t a coarse edge.

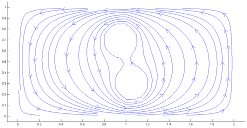
κ_2^{local}



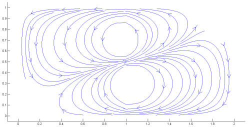
ϕ_1 on κ_2^{local}



ϕ_2 on κ_2^{local}



ϕ_3 on κ_2^{local}



Single phase flow equation: multiscale model reduction

Multiscale finite element space:

$$V_H = \oplus_{i=1}^{N_E} V_{\text{off}}^i, \qquad \dim V_H = N_u^H,$$
$$P_H = \{q_H \in L_0^2(\Omega) \mid q_H|_{K_i} = P_0(K_i)\}, \qquad \dim P_H = N_p^H$$

The matrix formulation of the reduced order model problem:

$$\begin{bmatrix} A_H & B_H^T \\ B_H & 0 \end{bmatrix} \begin{bmatrix} u_H \\ p_H \end{bmatrix} = \begin{bmatrix} R_u & 0 \\ 0 & R_p \end{bmatrix} \begin{bmatrix} A_h & B_h^T \\ B_h & 0 \end{bmatrix} \begin{bmatrix} R_u^T & 0 \\ 0 & R_p^T \end{bmatrix} \begin{bmatrix} u_H \\ p_H \end{bmatrix} = \begin{bmatrix} 0 \\ -F_H \end{bmatrix} \tag{2}$$

R_u: size $N_u^H \times N_u^h$, consists of a multiscale velocity basis per row.
R_p: size $N_p^H \times N_p^h$, consists of a coarse scale pressure basis per row.
$\begin{bmatrix} R_u^T & 0 \\ 0 & R_p^T \end{bmatrix}$: prolongation, $\begin{bmatrix} R_u & 0 \\ 0 & R_p \end{bmatrix}$: restriction.

$$u_H = -A_H^{-1} B_H p_H.$$

Difficulties: multiscale velocity bases usually need to be recompute when κ changes.

Single phase flow equation: Model reduction using neural network

Our goal: For flow equation, design deep neural networks to approximate the maps

$$u \approx \mathcal{N}(f;\theta), \quad \text{or} \quad u \approx \mathcal{N}(\kappa;\theta).$$

and perform model reduction more efficiently.
Generally in deep learning, given data pairs (x, y). Define

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

d: number of layers, σ: activation function.
θ: the trainable parameters in the network.
Aim to find θ^* by solving an optimization problem

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

Some ingredients in DNN

Fully connected, convolution, and locally connected layers:

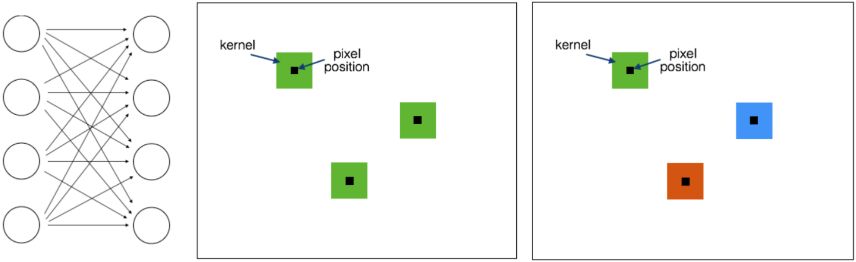


Figure 1: Left: fully connected, middle: convolution, right: locally connected.[1]

[1]Pictures from internet.

Single phase: velocity. Neural network architecture

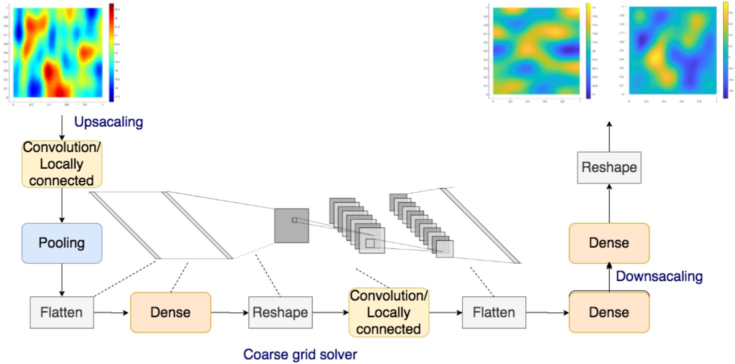


Figure 2: An illustration of the network architecture for flow approximation.

Single phase: velocity. Neural network architecture

We design a constraint loss function:

$$\mathcal{L}(u_{\text{pred}}, u_{\text{true}}; \theta) = \frac{1}{N} \sum_{i=1}^{N} \left(\frac{||u_{\text{pred, i}} - u_{\text{true},i}||_2}{||u_{\text{true},i}||_2} + \lambda ||B_h(u_{\text{pred},i} - u_{\text{true},i})||_2 \right) \tag{3}$$

where $u_{\text{pred}} = \mathcal{N}(\kappa; \theta)$ or $u_{\text{pred}} = \mathcal{N}(f; \theta)$
λ: a regularization constant.
This penalty in the loss function: help to enforce local mass conservation property in the predicted velocity solution.
The optimization problem is solved using ADAM algorighm.

Numerical example

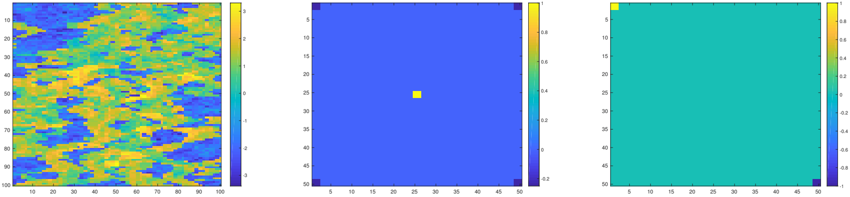


Figure 3: From left to right. Log scale of the absolute permeability field (SPE10 model layer 53). Five-spot source. Two-well source.

Numerical example for single phase flow equation approximation

Comparison of our proposed method when using different coarse grid.

N_p^H	$\lVert u_{\text{pred}} - u_{\text{true}} \rVert_{L^2}$ (%)	$\lVert u_{\text{pred}} - u_{\text{true}} \rVert_{L^2_\kappa}$ (%)	$\overline{M^i_{\text{true}} - M^i_{\text{pred}}}$
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 1: 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.

$$\lVert u_{\text{pred}} - u_{\text{true}} \rVert_{L^2_\kappa}^2 = \frac{\int_\Omega \kappa^{-1} |u_{\text{pred}} - u_{\text{true}}|^2 \mathrm{d}x}{\int_\Omega \kappa^{-1} |u_{\text{true}}|^2 \mathrm{d}x}$$

$$\overline{M^i_{\text{true}} - M^i_{\text{pred}}} = \frac{1}{N_k} \sum_{i=1}^{N_k} \int_{\partial k_i} (u_{\text{pred}} - u_{\text{true}}) \cdot n \mathrm{d}s$$

Numerical example for single phase flow equation approximation

Comparison of our proposed network LCN with standard CNN and DNN.

	$\left\|u_{\text{pred}} - u_{\text{true}}\right\|_{L^2}$	$\left\|u_{\text{pred}} - u_{\text{true}}\right\|_{L^2_\kappa}$	# of trainable paras
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 2: 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.

Numerical example for single phase flow equation approximation

Comparison of our proposed loss function with standard mse loss.

	Using Standard loss	Using our Loss funciton
$\left\|\left\|u_{\text{pred}} - u_{\text{true}}\right\|\right\|_{L^2}$ (%)	0.6	0.4
$\left\|\left\|u_{\text{pred}} - u_{\text{true}}\right\|\right\|_{L^2_\kappa}$ (%)	1.3	1.0
$\overline{M^i_{\text{true}} - M^i_{\text{pred}}}$	1.58e-8	1.64e-9

Table 3: 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.

Single phase saturation equation

Saturation equation:

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

where u is the velocity field obtained in the flow problem.
The fine grid discretization using finite volume method:

$$|K_i| \frac{S_i^{n+1} - S_i^n}{\mathrm{d}t} + \sum_{e_j \in \partial K_i} F_{ij}(S^n) = |K_i| r_i \tag{4}$$

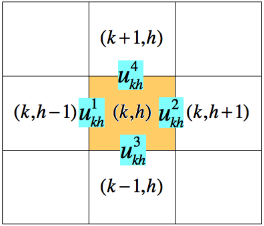
where F_{ij} is the upwind flux, i.e.

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

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

Consider the fine cell on k-th row and h-th column, the saturation feed-forward map become:

$$
\begin{aligned}
S_{k,h}^{n+1} = \\
&\frac{\mathrm{d}t}{|e|}\Big[\mathrm{Relu}(u_{kh}^{1}\cdot \boldsymbol{n}) - \mathrm{Relu}(u_{kh}^{2}\cdot \boldsymbol{n}) + \mathrm{Relu}(u_{kh}^{3}\cdot \boldsymbol{n}) - \mathrm{Relu}(u_{kh}^{4}\cdot \boldsymbol{n})\Big] S_{k,h}^{n} \\
&+ \frac{\mathrm{d}t}{|e|}\Big[\mathrm{Relu}(-u_{kh}^{1}\cdot \boldsymbol{n}) S_{k-1,h}^{n} + \mathrm{Relu}(-u_{kh}^{2}\cdot \boldsymbol{n}) S_{k+1,h}^{n} \\
&+ \mathrm{Relu}(-u_{kh}^{3}\cdot \boldsymbol{n}) S_{k,h-1}^{n} + \mathrm{Relu}(-u_{kh}^{4}\cdot \boldsymbol{n}) S_{k,h+1}^{n}\Big] + \mathrm{d}t\, r_{kh} + S_{k,h}^{n}
\end{aligned}
\tag{5}
$$



Single phase saturation equation: Neural network approximation

Goal: approximate the feed-forward map using DNN $\mathcal{M}$:

$$S^{n+1} \approx \mathcal{M}(S^n; f, u, \kappa)$$

(1) Write velocity on four edges of a fine grid: $U = [u^1, u^2, u^3, u^4]$.
(2) Initialize four pentadiagonal matrices W_i ($i = 1, 2, 3, 4$).

$$W_i = \mathrm{sparse}(I, J, V_i), \quad i = 1, 2, 3, 4. \tag{6}$$

I, J: the row and column indices vector, V_i: trainable parameters.
(3) Define the Sparse Velocity Layer

$$\phi_i(S^n, u^i) = \sigma((W_i \circ u^i)S^n + b_i) \tag{7}$$

$\circ$: the element-wise multiplication with broadcasting.
(4) The network to model the dynamics of the transport equation

$$\mathcal{M}(S^n; u) = \sum_{i=1}^{4} \phi_i(S^n, u^i) + S^n \tag{8}$$

Single phase saturation equation: Neural network approximation

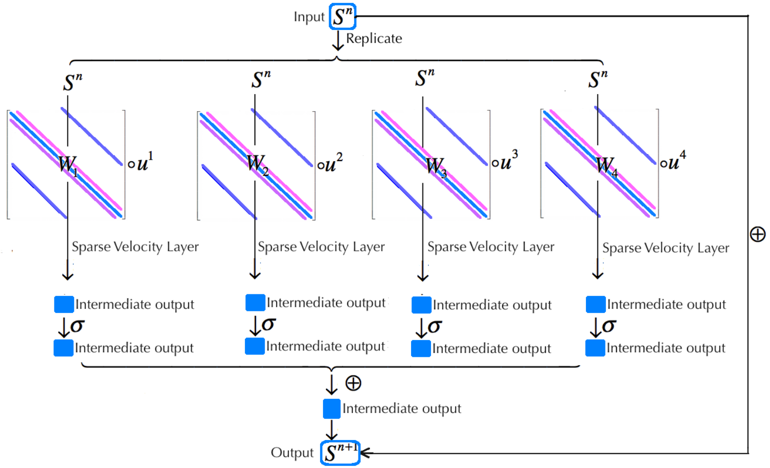


Figure 4: An illustration of the neural network architecture for learning saturation problem.

Numerical example: Single phase saturation equation

Comparison of our proposed network using Sparse Velocity layers, with simply sparse connected layers and densely connected layers.

	Sparse velocity	Sparse connect	Dense connect
$E_s(1)$ (%)	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).

Numerical example: Single phase saturation equation

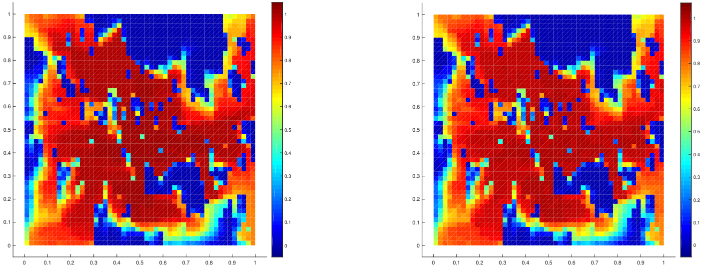


Figure 5: Comparison of saturation. Left: true solution at time step 1200, right: predicted solution at time step 1200 (given solution at time step 1001, use trained neural network to predict iteratively 199 times)

Two phase problem

Two phase flow equation

Flow equation for total velocity:

$$u = -\lambda(S)\kappa\nabla p \qquad \text{in} \quad D \tag{9}$$

$$\mathrm{div}(u) = f \qquad \text{in} \quad D \tag{10}$$

$$u \cdot n = 0 \qquad \text{on} \quad \partial D \tag{11}$$

and saturation equation for the water phase:

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

Total mobility:

$$\lambda(S) = \frac{\kappa_{rw}(S)}{\mu_w} + \frac{\kappa_{ro}(S)}{\mu_o}, \quad \mu_w, \mu_o \text{ are viscosity constants.}$$

The relative permeability: $\kappa_{rw}(S) = S^2, \quad \kappa_{ro}(S) = (1-S)^2$.
The flux function:

$$f_w(S) = \frac{\kappa_{rw}(S)/\mu_w}{\kappa_{rw}(S)/\mu_w + \kappa_{ro}(S)/\mu_o}.$$

Two phase flow equation

Finite volume with upwind scheme. On a fine grid K_i, the value S_i at time t^{n+1} can be obtained by

$$S_i^{n+1} = S_i^n + \frac{\mathrm{d}t}{|K_i|}[-\sum_{e_j \in \partial K_i} F_{ij}(S^t) + f_w(S^t) r_i^- + r_i^+] \qquad (12)$$

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

$$F_{ij}(S^t) = \begin{cases} \int_{e_j} (u_{ij}^t \cdot n) f_w(S_i^t) & \text{if} \quad u_{ij}^t \cdot n \geq 0 \\ \int_{e_j} (u_{ij}^t \cdot n) f_w(S_j^t) & \text{if} \quad u_{ij}^t \cdot n < 0 \end{cases} \qquad (13)$$

where $t = n$ or $t = n + 1$.

Two phase flow equation: sequential algorithm

Algorithm 1 Two phase flow and transport problem

```
Given initial conditions S^0, time step dt, final time T
Initialize S^0
Compute u^0 from (9) using λ(S^0)
for n = 1 : T/dt do
    S^{n,0} ← S^{n-1}, u^{n,0} ← u^{n-1}, j ← 1
    while Convergence criteria are not satisfied do
        Solve u^{n,j}, S^{n,j} from (9)-(12) in the coupled system using
        Newton-Ralphson method
        j = j + 1
    end while
    S^n ← S^{n,j}, u^n ← u^{n,j}
    n ← n + 1
end for
```

Two phase: Neural network approximation

Denote by n the current time step. Aim to train two surrogate models using DNN.

- Two phase flow equation: input $\{\lambda(S^n), f\}$, output u^n

$$u^n \approx \mathcal{N}_2(S^n, f)$$

- Two phase saturation equation: input $\{S^n, u^n, f\}$, output S^{n+1},

$$S^{n+1} \approx \mathcal{M}_2(S^n, u^n, f)$$

We adjust the networks designed before to two phase case.
$\mathcal{N}_2$ is a direct extension of the network $\mathcal{N}$ as described before.

Two phase saturation equation: DNN architecture

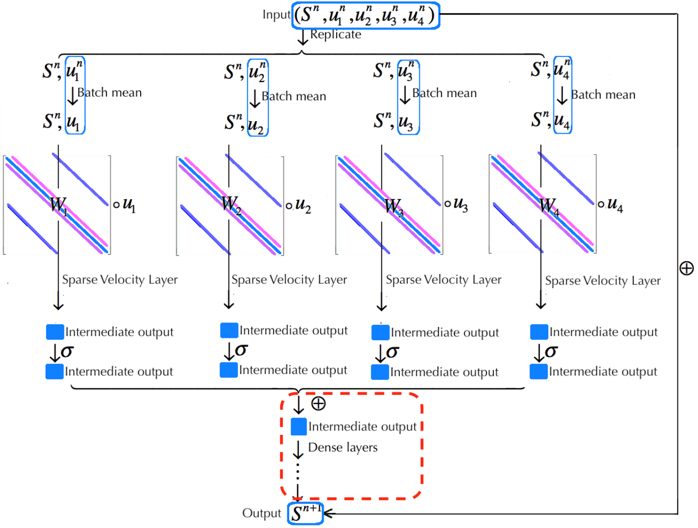


Figure 6: An illustration of the neural network architecture $\mathcal{M}_2$ for learning

Two phase flow equation: Neural network algorithm

Algorithm 2 Deep learning for flow and transport problem

Given saturation at time step n, network for velocity solver $\mathcal{N}_2$, network for saturation solver $\mathcal{M}_2$, m: number of time steps for prediction

$S \leftarrow S^n$

while $i < m$ **do**

  $\lambda(S^n) \leftarrow \lambda(S)$

  Predict u^n using $\mathcal{N}_2$ with input $\lambda(S^n), f$

  Predict S^{n+1} using $\mathcal{M}_2$, with input S^n, u^n, f

  $S \leftarrow S^{n+1}$

  $i \leftarrow i + 1$

end while

return S

Two phase flow equation: Numerical example

Training samples are generated using $f_j = j$, for $j = 1, 2, 3, 4, 5$. Testing samples are generated using random f in different time intervals.

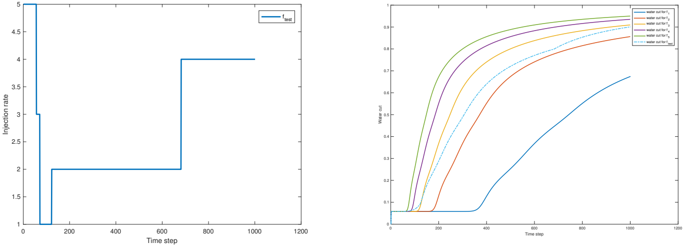


Figure 7: Left: a random source f_{test} for generating test samples. Right: the water cuts at all time steps, for training sources f_j ($j = 1, \cdots, n_s = 5$), and f_{test}.

Two phase flow equation: Numerical example

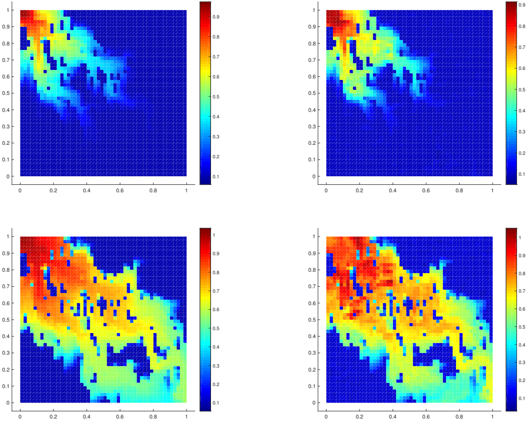


Figure 8: Comparison of saturation using Algorithm 2. Given the initial solution, top row: after 50 prediction step (relative L^2 error: 7.0%), bottom row: after 950 predicted time steps (relative L^2 error: 6.5%).

Two phase flow equation: Numerical example

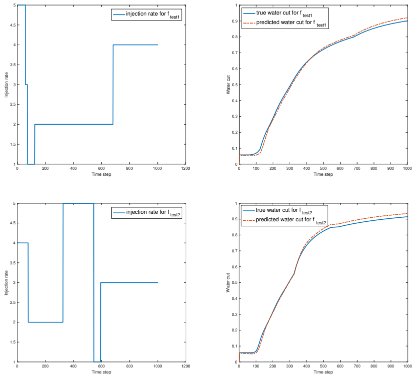


Figure 9: For two different source terms, the comparison of water cuts obtained from the true saturation and the predicted saturation.

Computational time for 1000 time steps: Algorithm 1 takes 94.36 seconds, Algorithm 2 takes 2.25 seconds.

Summary

- An efficient network architecture is proposed to approximate the maps in flow simulation.
- The designed layers in the network are in analogy to downscaling and upscaling procedures in multiscale model reduction methods.
- The customized loss function for the flow problem help to preserve the local mass conservative property of the predicted velocity.
- A residual type of neural network is proposed to approximate the dynamics in the saturation equation. Custom sparsely connected layers are introduced to reduce the complexity of the network.
- The trained feedforward map for the saturation problem can be used to iteratively many times to predict the solution in the long run.

- Thank you!

Single phase: velocity. Neural network architecture

Motivated by the standard multiscale method, we design the following network architechture.

(1) Input tensor f or κ with size $\sqrt{N_p^h} \times \sqrt{N_p^h}$

(2) If input κ, a few locally connected layers are needed to extract the hidden features.

(3) Average pooling with size $\left(\sqrt{N_p^h}/\sqrt{N_p^H}\right) \times \left(\sqrt{N_p^h}/\sqrt{N_p^H}\right)$ (Coarse scale features)

(4) Flatten, and fully connected layers to obtain hidden output with dim $N_p^H \times 1$(dim of coarse scale pressure)

(5) Reshape back to $\sqrt{N_p^H} \times \sqrt{N_p^H}$

(6) Locally connected layers

(7) Flatten and fully connected to $N_u^H \times 1$ (dim of coarse scale velocity)

(8) Fully connected layer as decoding/downscaling, to the dim $N_u^h \times 1$ (Fine scale velocity)