Machine Learning in OPM

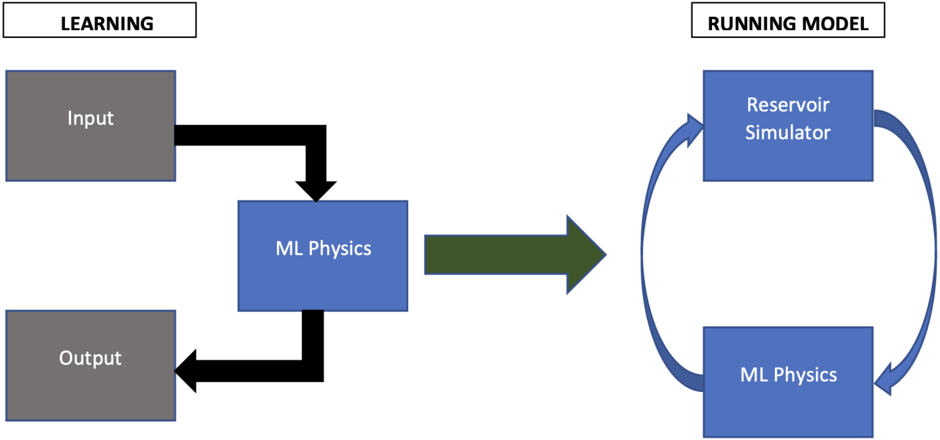
Birane Kane*, Tor H. Sandve

NORCE

April 2024

Motivation

New framework where we embed a neural network toolbox to our existing multiphase flow simulator in Dune or OPM (dune-project.org, opm-project.org).



Example: a two phase problem

Domain $\Omega \in \mathbb{R}^d$, $d \in \{2, 3\}$. Incompressible, immiscible phases $\alpha = \{w, n\}$. Unknown variables are p_w and s_n.

$$
\begin{aligned}
-\nabla \cdot \Big((\lambda_w + \lambda_n)\mathbb{K}\nabla p_w + \lambda_n p_c' \mathbb{K}\nabla s_n - (\rho_w\lambda_w + \rho_n\lambda_n)\mathbb{K}\mathbf{g} \Big) &= q_w + q_n, \\
\phi\frac{\partial s_n}{\partial t} - \nabla \cdot \Big(\lambda_n \mathbb{K}(\nabla p_w - \rho_n \mathbf{g}) \Big) - \nabla \cdot \Big(\lambda_n p_c' \mathbb{K}\nabla s_n \Big) &= q_n.
\end{aligned}
$$

$\lambda_\alpha := \lambda_\alpha(s_\alpha)$	phase mobility	$p_c := p_c(s_n)$	capillary-pressure
$\mathbf{g}$	gravity	$\mathbb{K}$	permeability tensor
$\phi > 0$	porosity	ρ_α	phase density
		q_α	source/sink term

Example: a two phase problem

Phases mobilities $\lambda_w = \lambda_w(s_n) = \frac{k_{rw}(s_n)}{\mu_w}$, $\qquad \lambda_n = \lambda_n(s_n) = \frac{k_{rn}(s_n)}{\mu_n}$

where μ_α is the viscosity and $k_{r\alpha}$ is the relative permeability of phase $\alpha = \{w, n\}$.

$$k_{rw}(s_{e_w}) = s_{e_w}^{\frac{2+3\theta}{\theta}}, \quad k_{rn}(s_{e_n}) = (s_{e_n})^2(1-(1-s_{e_n})^{\frac{2+\theta}{\theta}}),$$

where the effective saturation s_{e_α} is

$$s_{e_\alpha} = \frac{s_\alpha - s_{r\alpha}}{1 - s_{rw} - s_{rn}}, \quad \forall\, \alpha \in \{w, n\}.$$

Here $s_{r\alpha}$, $\alpha \in \{w, n\}$ are the phases residual saturations, $\theta \in [0.2, 3.0]$ is the inhomogeneity.

OPM-NN framework

- Model training
- Model Parsing
- Layer Conversion
- Code Generation
- Integration and Customization

OPM-NN framework

- Model training

Develop and train the neural network model in Python using a deep learning library such as Keras.

```
# design the neural network model
model = Sequential()
model.add(Dense(3, input_dim=1, activation='relu'))
model.add(Dense(10, activation='relu'))
model.add(Dense(1))
# define the loss function and optimization algorithm
model.compile(loss='mse', optimizer='adam')
# ft the model on the training dataset
model.fit(x, y, epochs=1000, batch_size=100, verbose=0)
# make predictions for the input data
yhat = model.predict(x)
# inverse transforms

#save model
from kerasify import export_model
export_model(model, 'example.modelBCkrn')
```

OPM-NN framework

- Model Parsing

First the Keras model file generated after the training of the model is read and interpreted. The structure and layers of the model are analyzed.

```
if layer_type == 'Dense':
  weights = layer.get_weights()[0]
  biases = layer.get_weights()[1]
  activation = layer.get_config()['activation']

  f.write(struct.pack('I', LAYER_DENSE))
  f.write(struct.pack('I', weights.shape[0]))
  f.write(struct.pack('I', weights.shape[1]))
  f.write(struct.pack('I', biases.shape[0]))

  weights = weights.flatten()
  biases = biases.flatten()

  write_floats(f, weights)
  write_floats(f, biases)

  write_activation(activation)
```

OPM-NN framework

- Layer Conversion

Each layer of the Keras model is then converted into its equivalent C++ representation. The code maps Keras layers to the corresponding C++ implementations.

```
template<class Evaluation>
class KerasLayerDense : public KerasLayer<Evaluation> {
  public:
    KerasLayerDense() {}

    virtual ~KerasLayerDense() {}

    virtual bool LoadLayer(std::ifstream* file);

    virtual bool Apply(Tensor<Evaluation>* in, Tensor<Evaluation>* out);

  private:
    Tensor<float> weights_;
    Tensor<float> biases_;

    KerasLayerActivation<Evaluation> activation_;
};
```

OPM-NN framework

- Code Generation

Based on the parsed model and optimized settings, C++ code is generated. This code is tailored to run efficiently on the target platform, taking advantage of available hardware acceleration, parallelization, automatic differentiation.

Integration to OPM code

- Integration

The generated C++ code can then be incorporated into OPM Flow scripts by accessing it as a usual function

```
KerasModel <Value > model;          % Declare the NN model
model.LoadModel ( path );           % Load a saved model
Tensor <Value > in {1};             % Declare the input tensor
in.data = {{S}};                            % Initialize the input tensor
Tensor <Value > out;                % Declare the output tensor
model.Apply (&in , &out);           % Run the model
```

From

```
template <class Evaluation>
static Evaluation twoPhaseSatKrw(const Params& params,
            const Evaluation& Sw)
{
    assert(0.0 <= Sw && Sw <= 1.0);

    return pow(Sw, 2.0/params.theta() + 3.0);
}
```

to

```
template <class Evaluation>
static Evaluation twoPhaseSatKrw(const Params& params,
            const Evaluation& Sw)
{
    assert(0.0 <= Sw && Sw <= 1.0);

    Tensor <Evaluation > inw {1};
    inw.data = {{Sw}};
    Tensor <Evaluation > outw;
    modelkrw.Apply (&inw , &outw);
    f_theta_krw = outw.data_[0];
    return f_theta_krw;
}
```

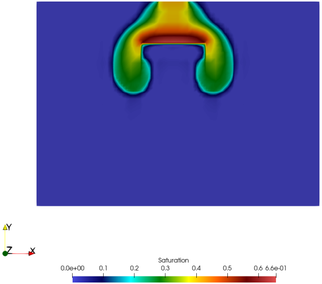
From

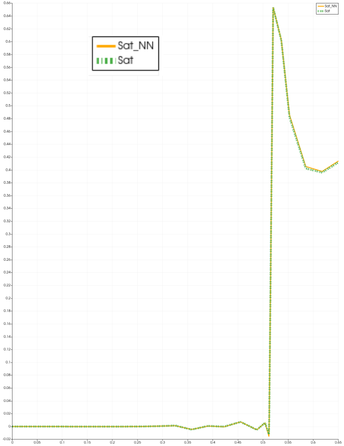
```
template <class Evaluation>
static Evaluation twoPhaseSatKrn(const Params& params,
            const Evaluation& Sw)
{
    assert(0.0 <= Sw && Sw <= 1.0);
    Scalar exponent = 2.0/params.theta() + 1.0;
    const Evaluation Sn = 1.0 - Sw;
    return Sn*Sn*(1. - pow(Sw, exponent));
}
```

to

```
template <class Evaluation>
static Evaluation twoPhaseSatKrn(const Params& params,
            const Evaluation& Sw)
{
    assert(0.0 <= Sw && Sw <= 1.0);
    const Evaluation Sn = 1.0 - Sw;
    Tensor <Evaluation > in {1};
    in.data = {{Sn}};
    Tensor <Evaluation> outn;
    modelkrn.Apply (&in , &outn);
    f_theta_krn = outn.data_[0];
    return f_theta_krn;
}
```

DNAPL pooling over lens

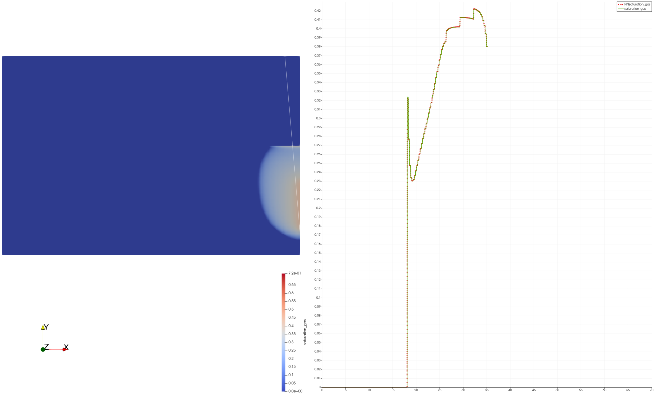




Left:Saturation distribution after 2000 s. Right: profile along the line x=0.45 m

CO_2 injection

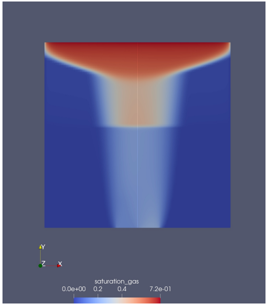
Problem where CO_2 is injected under a low permeable layer at a depth of 2700m.

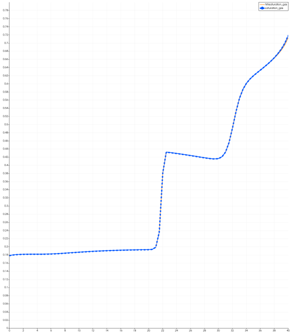


Left:Saturation distribution at final time. Right: profile along specified line

Water air problem

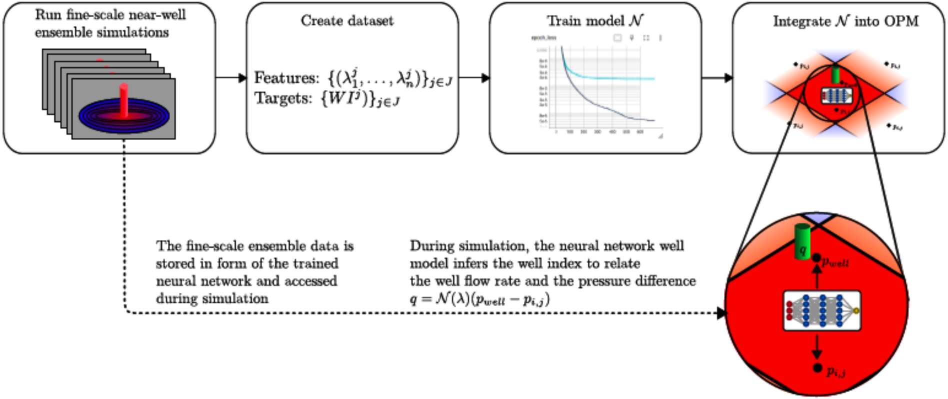
During buoyancy driven upward migration, the gas passes a rectangular high temperature area. This decreases the temperature of the high-temperature area and accelerates gas infiltration due to the lower viscosity of the gas.





Left:Saturation at final time. Right: profile along the line x=20 m

ML near-well model in OPM Flow



A machine-learned near-well model in OPM Flow

Peter Moritz von Schultzendorff*, Tor Harald Sandve, Birane Kane, David Landa-Marban, Jakub Wiktor Both, Jan Martin Nordbotten

Summary

- Provided a proof of concept / prototype
- Simple integration to reservoir simulator
- Wide range of possible applications