基于深度学习的动态地下流动问题的代理模型

王炎中

摘要

剩余油分布预测技术是一种在油田开采过程中对地下含油饱和度场的分布进行预测的技术。目前大多数剩余油分布预测技术都是基于油藏数值模拟法，具有建模周期长、计算代价大、泛化能力差等难题，而且预测时没有考虑历史饱和度场信息。剩余油分布变化是一个连续的物理过程，文中提出一种基于卷积神经网络和循环神经网络的代理模型[1]。它能基于前面时刻的信息预测后续时刻的饱和度场和压力场的状态。与传统的数值模拟相比，它的计算复杂度以及建模时间都得到大幅的缩减，同时它的预测精度也极具竞争力。多组不同的实验表明，该方法在预测饱和度时具有代替油藏数值模拟正向物理过程的潜力。

关键词：深度学习；饱和度预测；压力预测；代理模型；时间序列

1 引言

剩余油分布规律一直是石油工程师长期关心的问题，掌握地下剩余油分布就意味着为各种调整措施指明了方向。前人经过多年研究，形成了油藏数值模拟法、岩心实验分析法、试井和测井法等各种寻找剩余油的方法，为提高油田采收率起到巨大作用。这些研究成果对剩余油分布的研究提供了有力的支持。近年来，随着机器学习技术的逐渐成熟、计算机算力的不断发展以及海量数据的可用，机器学习技术在石油领域取得了大量研究成果。

2 相关工作

Zhu 和 Nicholas[2]首次在石油领域使用基于卷积神经网络的 Encoder-Decoder 的全卷积网络实现了从渗透率场到剩余油饱和度场到压力场的映射，为卷积网络在石油领域中的发展做出了突出性的贡献。至此，深度学习在石油以及地下水领域得到广泛的关注，各种论文相继问世，极大的推动了石油等领域的发展。

3 本文方法

3.1 本文方法概述

以卷积网路和 LSTM 为模型的基本架构，构建了一个 Encoder-Decoder 的全卷积网络模型。模型的输入只需要一个渗透率场（将平面地层进行网格划分，每个网格上填上相应的值，每个网格代表一个像素，可以理解成一张伪图像），模型的输出是含水饱和度场（对于一个油水两相的系统，含水饱和度加上含油饱和度的为 1，因此知道其中一种就能反推出另一种）和压力场的时间序列，也即含水饱和度场和压力视频。模型的大致结构如图 1 所示，它由编码器和解码器组成，其中 Encoder 网络中提取的局部特征与 Decoder 网络中的上采样特征连接起来预测生成状态图。Encoding Net 捕捉地质特征，提取初级特征，并逐渐缩小图片的大小（具体表现为图像的高和宽）；ConvLSTM Net 用来捕获特

征图的时间信息；Decoding Net 将包含时间信息的特征图恢复到原始图片大小，并逐渐将高级特征转化成目标特征图

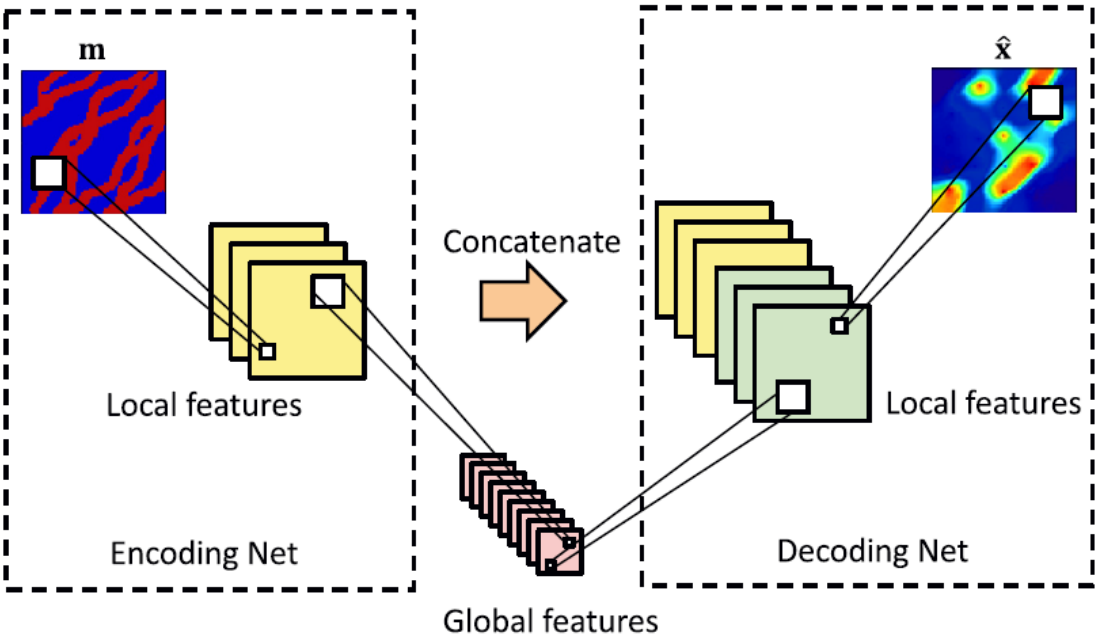


图 1: 模型的整体结构

3.2 特征提取模块

特征提取模块主要由 Encoder 网络构成，它由一系列卷积块和残差块组成，如图 2 所示。

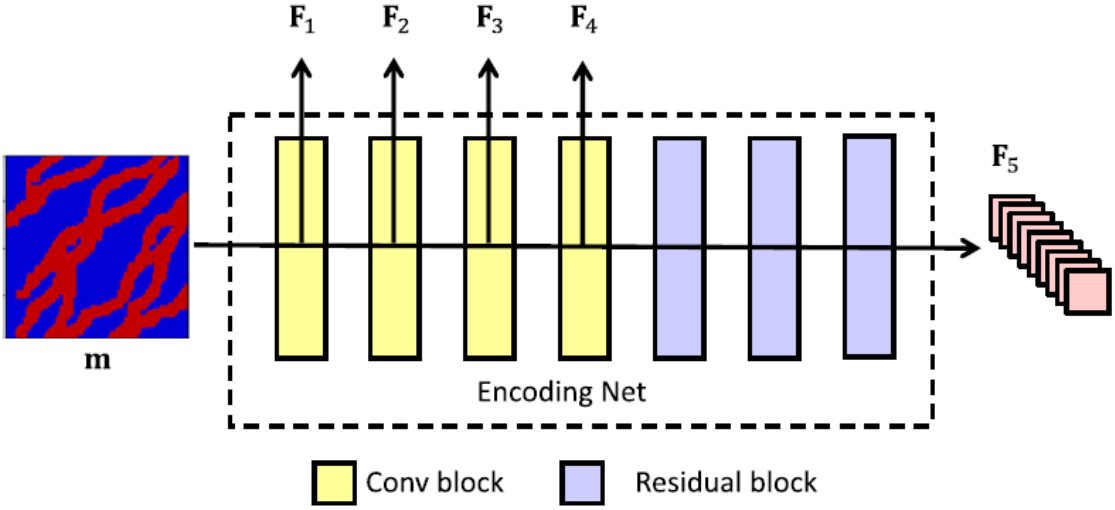


图 2: Encoding 网络

Encoder 网络提取出原始图像的特征以及将图像的长宽变小之后，Decoder 网络利用这些提取到特征恢复到原始图片大小并生成目标特征图，它由一系列转置卷积块和残差块组成，如图 3 所示。

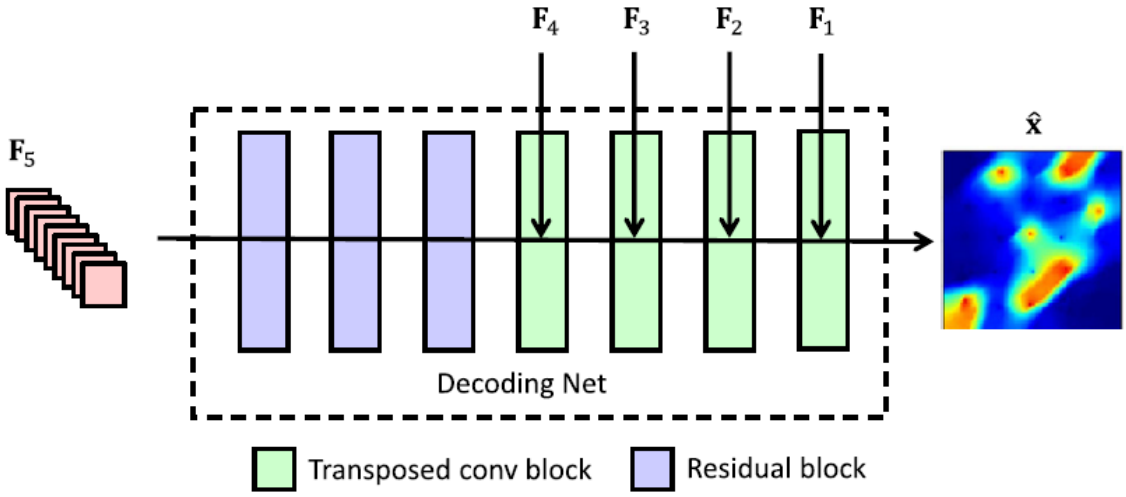
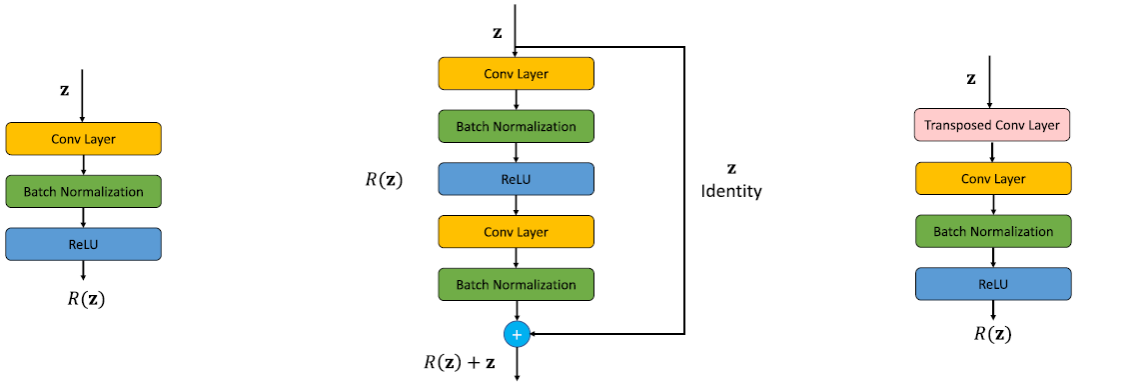


图 3: Decoding 网络

卷积块、残差块以及转置卷积块的具体结构如图 4 所示。



(a) Conv block (b) Residual block (c) Transposed conv block

图 4: 卷积块、残差块以及转置卷积块的示意图

整个模型（Recurrent R-U-Net）的具体结构如图 5 所示。

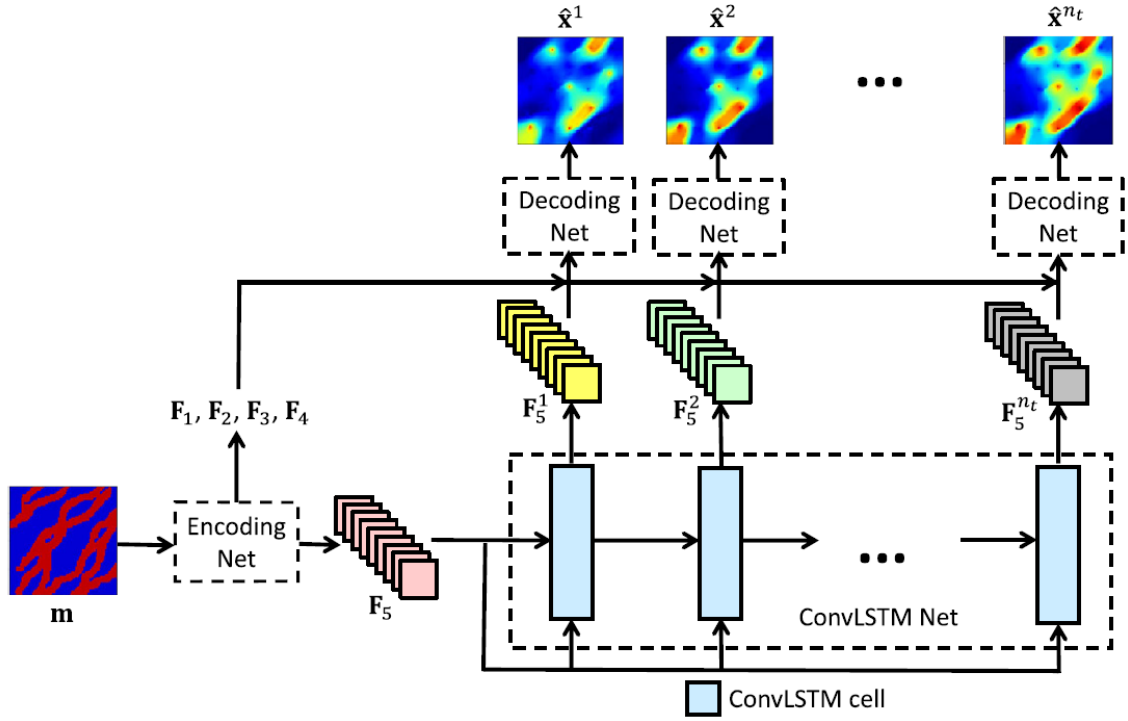


图 5: Recurrent R-U-Net 模型

整个网络的详细结构，每个小模块、网络层的具体定义如图 6 所示。

Net	Layer	Output size
Encoder	Input	$(n_x, n_y, 1)$
	conv, 16 filters of size 3×3, stride 2	$(n_x/2, n_y/2, 16)$
	conv, 32 filters of size 3×3, stride 1	$(n_x/2, n_y/2, 32)$
	conv, 64 filters of size 3×3, stride 2	$(n_x/4, n_y/4, 64)$
	conv, 128 filters of size 3×3, stride 1	$(n_x/4, n_y/4, 128)$
	residual block, 128 filters of size 3×3, stride 1	$(n_x/4, n_y/4, 128)$
	residual block, 128 filters of size 3×3, stride 1	$(n_x/4, n_y/4, 128)$
	residual block, 128 filters of size 3×3, stride 1	$(n_x/4, n_y/4, 128)$
ConvLSTM	convLSTM2D block, 128 filters of size 3×3, stride 1	$(n_x/4, n_y/4, 128, n_t)$
Decoder	residual block, 128 filters of size 3×3, stride 1	$(n_x/4, n_y/4, 128, n_t)$
	residual block, 128 filters of size 3×3, stride 1	$(n_x/4, n_y/4, 128, n_t)$
	residual block, 128 filters of size 3×3, stride 1	$(n_x/4, n_y/4, 128, n_t)$
	transposed conv, 128 filters of size 3×3, stride 1	$(n_x/4, n_y/4, 128, n_t)$
	transposed conv, 64 filters of size 3×3, stride 2	$(n_x/2, n_y/2, 64, n_t)$
	transposed conv, 32 filters of size 3×3, stride 1	$(n_x/2, n_y/2, 32, n_t)$
	transposed conv, 16 filters of size 3×3, stride 2	$(n_x, n_y, 16, n_t)$
	conv layer, 1 filter of size 3×3, stride 1	$(n_x, n_y, 1, n_t)$

图 6: Recurrent R-U-Net 模型具体细节

3.3 损失函数定义

Recurrent R-U-Net 模型的损失函数为普通的 L1 损失。

4 复现细节

4.1 与已有开源代码对比

此文章没有开源代码。同时，文章的数据也是作者自己制作的，没有开源，因此模型的复现效果会有所差异，具体的代码如图 7，是基于 Tensorflow 中的 Kears 框架编写的。

```
# -------------Encoder--------------
  self.add(Input(shape=input_shape, name='image'))
self.conv1 = Conv_block(output_features=16, strides=(2, 2), data_format=data_format)
self.conv2 = Conv_block(output_features=32, strides=(1, 1), data_format=data_format)
self.conv3 = Conv_block(output_features=64, strides=(2, 2), data_format=data_format)
self.conv4 = Conv_block(output_features=128, strides=(1, 1), data_format=data_format)

# all the Residual block are the same
self.res_conv1 = Residual_block(data_format=data_format)
self.res_conv2 = Residual_block(data_format=data_format)
self.res_conv3 = Residual_block(data_format=data_format)

self.repeat_conv = Repeat_conv(self.time_step)
self.ConvLSTM = ConvLSTM2D(filters=128, kernel_size=(3, 3), strides=(1, 1), padding='same',
                   data_format=data_format, activation='tanh', recurrent_activation='hard_sigmoid',
                   use_bias=True, return_sequences=True, dropout=False, recurrent_dropout=False)

# ----------------Decoder---------------
self.time_res_conv1 = Time_Residual_block(data_format=data_format)
self.time_res_conv2 = Time_Residual_block(data_format=data_format)
self.time_res_conv3 = Time_Residual_block(data_format=data_format)

self.time_t_conv1 = Time_Transposed_conv_block(output_features=128, kernel_size=(3, 3), strides=(1, 1), data_format=data_format)
self.time_t_conv2 = Time_Transposed_conv_block(output_features=64, kernel_size=(3, 3), strides=(2, 2), data_format=data_format)
self.time_t_conv3 = Time_Transposed_conv_block(output_features=32, kernel_size=(3, 3), strides=(1, 1), data_format=data_format)
self.time_t_conv4 = Time_Transposed_conv_block(output_features=16, kernel_size=(3, 3), strides=(2, 2), data_format=data_format)
self.time_t_conv5 = Time_Conv_block(output_features=1, data_format=data_format)
```

图 7: Recurrent R-U-Net 模型代码

```python
def call(self, inputs, training=None, mask=None):
    enc1 = self.conv1(inputs)  # (40, 40, 16)
    time_enc1 = self.repeat_conv(enc1)  # (time_step, 40, 40, 16)
    enc2 = self.conv2(enc1)  # (40, 40, 32)
    time_enc2 = self.repeat_conv(enc2)  # (time_step, 40, 40, 32)
    enc3 = self.conv3(enc2)  # (20, 20, 64)
    time_enc3 = self.repeat_conv(enc3)  # (time_step, 20, 20, 64)
    enc4 = self.conv4(enc3)  # (20, 20, 128)
    time_enc4 = self.repeat_conv(enc4)  # (time_step, 20, 20, 128)

    x = self.res_conv1(enc4)  # (20, 20, 128)
    x = self.res_conv2(x)  # (20, 20, 128)
    x = self.res_conv3(x)  # (20, 20, 128)

    x = Repeat_conv(self.time_step)(enc4)  # (time_step, 20, 20, 128)
    x = self.ConvLSTM(x)  # (time_step, 20, 20, 128)

    x = self.time_res_conv1(x)  # (time_step, 20, 20, 128)
    x = self.time_res_conv2(x)  # (time_step, 20, 20, 128)
    dec4 = self.time_res_conv3(x)  # (time_step, 20, 20, 128)

    merge4 = tf.concat([time_enc4, dec4], -1)  # (time_step, 20, 20, 128+128)
    dec3 = self.time_t_conv1(merge4)  # (time_step, 20, 20, 128)
    merge3 = tf.concat([time_enc3, dec3], -1)  # (time_step, 20, 20, 64+128)
    dec2 = self.time_t_conv2(merge3)  # (time_step, 40, 40, 64)
    merge2 = tf.concat([time_enc2, dec2], -1)  # (time_step, 40, 40, 32+64)
    dec1 = self.time_t_conv3(merge2)  # (time_step, 40, 40, 32)
    merge1 = tf.concat([time_enc1, dec1], -1)  # (time_step, 40, 40, 16+32)
    dec0 = self.time_t_conv4(merge1)  # (time_step, 80, 80, 16)
    output = self.time_t_conv5(dec0)  # (time_step, 80, 80, 1)

    return output
```

```
Model: "recurrent_r_u__net_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv_block_24 (Conv_block)   (None, 40, 40, 16)        512
_________________________________________________________________
conv_block_25 (Conv_block)   (None, 40, 40, 32)        4768
_________________________________________________________________
conv_block_26 (Conv_block)   (None, 20, 20, 64)        18752
_________________________________________________________________
conv_block_27 (Conv_block)   (None, 20, 20, 128)       74368
_________________________________________________________________
residual_block_18 (Residual_ (None, 20, 20, 128)       296192
_________________________________________________________________
residual_block_19 (Residual_ (None, 20, 20, 128)       296192
_________________________________________________________________
residual_block_20 (Residual_ (None, 20, 20, 128)       296192
_________________________________________________________________
repeat_conv_6 (Repeat_conv)  multiple                  0
_________________________________________________________________
conv_lst_m2d_6 (ConvLSTM2D)  multiple                  1180160
_________________________________________________________________
time__residual_block_18 (Tim (None, 10, 20, 20, 128)   296192
_________________________________________________________________
time__residual_block_19 (Tim (None, 10, 20, 20, 128)   296192
_________________________________________________________________
time__residual_block_20 (Tim (None, 10, 20, 20, 128)   296192
_________________________________________________________________
time__transposed_conv_block_ (None, 10, 20, 20, 128)   443136
_________________________________________________________________
time__transposed_conv_block_ (None, 10, 40, 40, 64)    147840
_________________________________________________________________
time__transposed_conv_block_ (None, 10, 40, 40, 32)    37056
_________________________________________________________________
time__transposed_conv_block_ (None, 10, 80, 80, 16)    9312
_________________________________________________________________
time__conv_block_6 (Time_Con (None, 10, 80, 80, 1)     149
=================================================================
Total params: 3,693,205
Trainable params: 3,689,171
Non-trainable params: 4,034
_________________________________________________________________
```

图 8: Recurrent R-U-Net 模型代码（续）

4.2 实验环境搭建

Tensorflow 的版本为 2.4.1，Keras 不用单独下载，已经包含在里面。

4.3 创新点

因数据集不同，根据原文复现的代码预测的结果受到比较严重的影响，预测效果总是达不到文章的精度，为了缓解这个问题，将直接学习标签的训练策略更改成残差学习。以含水饱和度 S_w^t 为例，具体表现为：论文直接学习的是 S_w^1，S_w^2，S_w^3，...，S_w^{15} 这样的时间序列，改进的方法是，学习两个连续时间步的饱和度差值 S_w^1-S_w^0，S_w^2-S_w^1，S_w^3-S_w^2，...，S_w^{15}-S_w^{14}。从流体力学中的基本渗流微分方程来说，它是具备一定的物理意义的。简单来说，可以从渗流微分方程中推导出模型的输出（饱和度差值）正比于模型的输出（渗透率场）。实验结果表明，改进后的预测效果明显优于前者。

5 实验结果分析

实验结果主要分为两类，一类是利用论文中的代码复现的结果，另一类是按自己方法改进后的预测结果。利用论文中的代码复现的结果如图 9 和图 10 所示，预测的结果分别为压力和饱和度场。

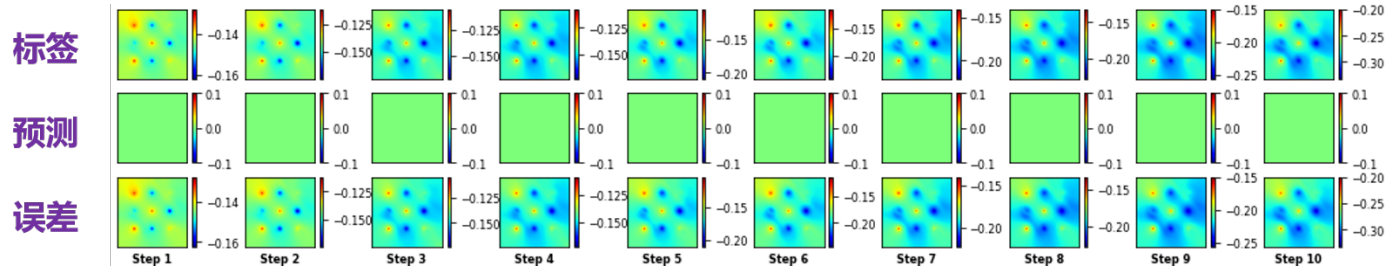


图 9: 利用论文代码复现的压力

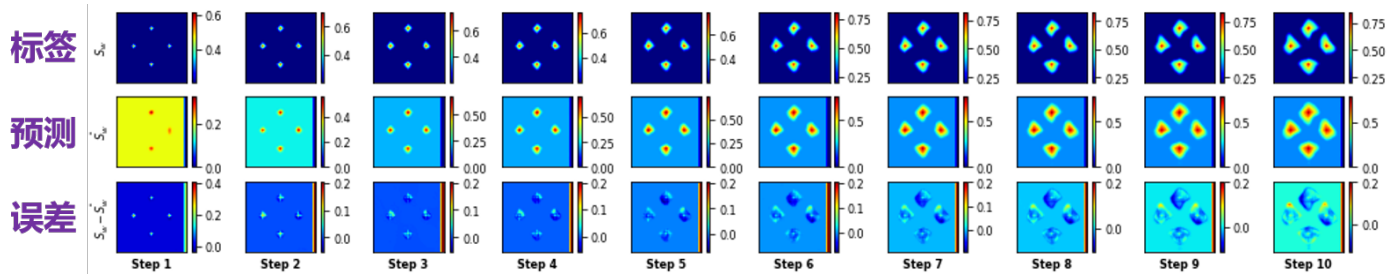


图 10: 利用论文代码复现的饱和度场

改进后的预测效果如图 11 和图 12 所示，分别为压力和饱和度场。

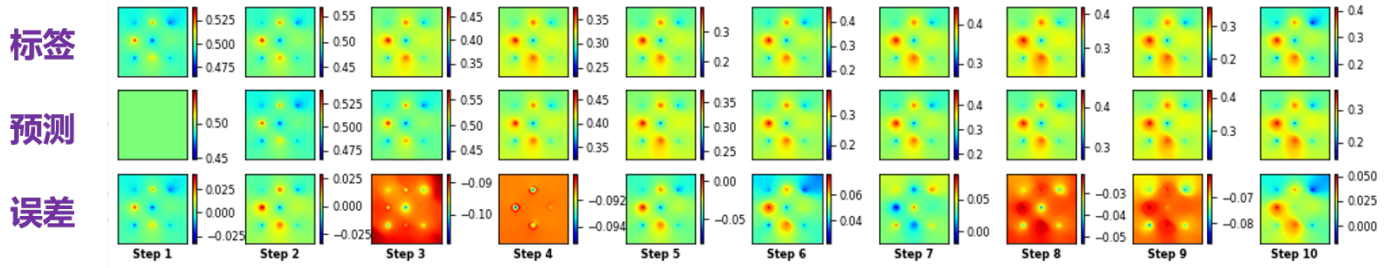


图 11: 改进后预测的压力

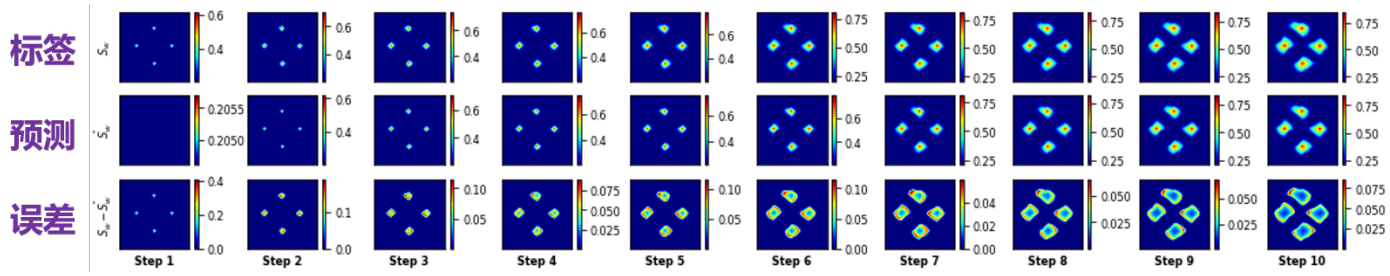


图 12: 改进后预测的饱和度场

6 总结与展望

论文利用一个全卷积网络实现了渗透率到压力和饱和度的时间序列的预测，它能基于前面时刻的信息预测后续时刻的饱和度场和压力场的状态。与传统的数值模拟相比，它的计算复杂度以及建模时间都得到大幅的缩减，同时它的预测精度也极具竞争力。然而，强有力的物理意义并没有在论文中得到体现，同时网络模型本身不具备任何的物理意义，一个潜在的方向可能是利用物理意义模型（https://www.physicsbaseddeeplearning.org）实现此类过程，让其预测过程更具备可解释性。

参考文献

[1] TANG M, LIU Y, DURLOFSKY L J. A deep-learning-based surrogate model for data assimilation in dynamic subsurface flow problems[J/OL]. Journal of Computational Physics, 2020, 413: 109456. https://www.sciencedirect.com/science/article/pii/S0021999120302308. DOI: https://doi.org/10.1016/j.jcp.2020.109456.

[2] ZHU Y, ZABARAS N. Bayesian deep convolutional encoder-decoder networks for surrogate modeling and uncertainty quantification[J/OL]. Journal of Computational Physics, 2018, 366: 415-447. https://www.sciencedirect.com/science/article/pii/S0021999118302341. DOI: https://doi.org/10.1016/j.jcp.2018.04.018.