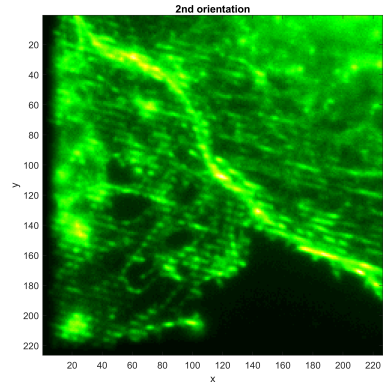
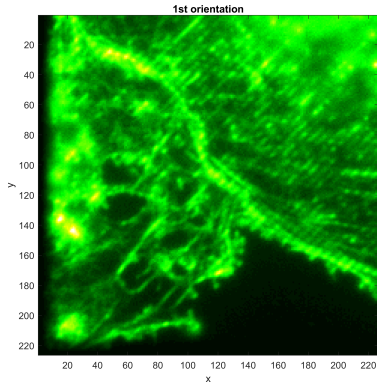
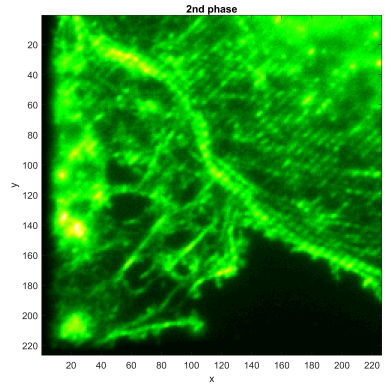
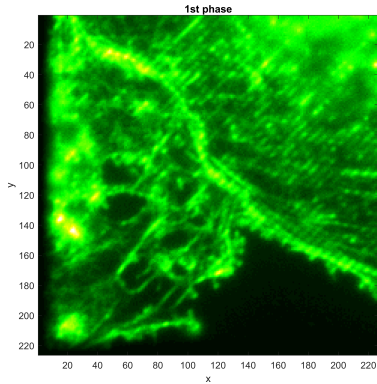


NN and Tensorflow in 3D-SIM

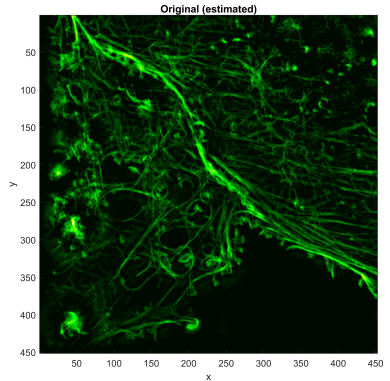
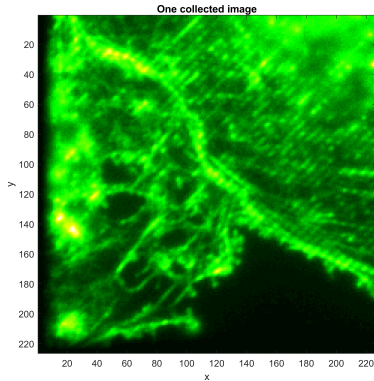
Collected images (of different orientation)



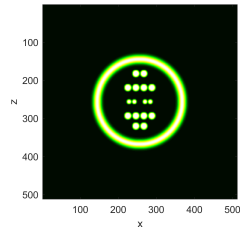
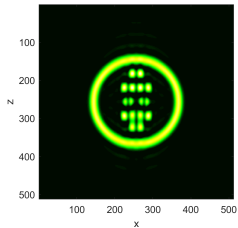
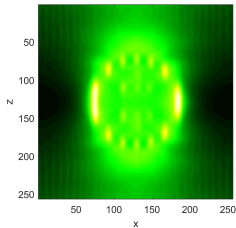
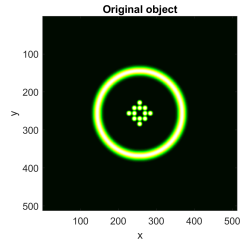
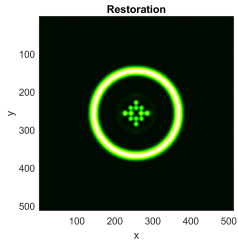
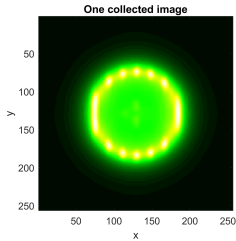
Collected images (of different phases)



Collected images vs Original object (estimated)



Simulated data



Problem statement

Forward model:

$$g(\mathbf{x}, z) = \sum_{m=1}^{N_m} [o(\mathbf{x}, z) j_m(\mathbf{x})] \circledast [h(\mathbf{x}, z) i_m(z)]. \quad (1.1)$$

Inverse problem: given $g(\mathbf{x}, z)$, restore $o(\mathbf{x}, z)$?

Problem statement

Forward model:

$$g(\mathbf{x}, z) = \sum_{m=1}^{N_m} [o(\mathbf{x}, z) j_m(\mathbf{x})] \circledast [h(\mathbf{x}, z) i_m(z)]. \quad (1.1)$$

Inverse problem: given $g(\mathbf{x}, z)$, restore $o(\mathbf{x}, z)$?

For example: given $g(\mathbf{x}, z)$ of size $128 \times 128 \times 32 \times 15$ (the 15 comes from 3 orientations and 5 phases), restore $o(\mathbf{x}, z)$ of size $256 \times 256 \times 64$?

Using NN (without the physical model)

Let $\mathcal{N}_{\mathbf{w}, \mathbf{b}}$ be a neural network, then one can do optimization on the training data:

$$\left(\tilde{\mathbf{w}}, \tilde{\mathbf{b}} \right) = \min_{(\mathbf{w}, \mathbf{b})} \left\| \mathcal{N}_{\mathbf{w}, \mathbf{b}}(g_{\text{train}}) - o_{\text{train}} \right\|. \quad (2.1)$$

Then one can estimate the original object (given any test data set g_{test}) by:

$$\hat{o}(\mathbf{x}, z) = \mathcal{N}_{\tilde{\mathbf{w}}, \tilde{\mathbf{b}}}(g_{\text{test}}(\mathbf{x}, z)). \quad (2.2)$$

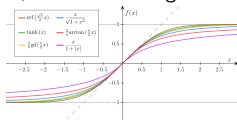
Next slide will explain the weights and bias \mathbf{w}, \mathbf{b} of a 1-layer neural network.

1-layer NN

1. Vectorize $g(\mathbf{x}, \mathbf{z})$ of size $128 \times 128 \times 32 \times 15$ into a vector \mathbf{g}_0 of size 7864320.
2. First layer neurons (of size 100000 for example):

$$\mathbf{g}_1 = \sigma(\mathbf{W}_1 \mathbf{g}_0 + \mathbf{b}_1),$$

where \mathbf{W}_1 is the first layer weight matrix of size 100000×7864320 , \mathbf{b}_1 is the first layer bias vector of size 100000, and σ is the sigmoid function.



3. The output layer is:

$$\mathbf{o} = \sigma(\mathbf{W}_{\text{out}} \mathbf{g}_1 + \mathbf{b}_{\text{out}}),$$

where \mathbf{W}_{out} is the output layer weight matrix of size 4194304×100000 , \mathbf{b}_{out} is the output layer bias vector of size 4194304.

4. The output vector \mathbf{o} of size 4194304 is reshaped into the estimated object $\mathbf{o}(\mathbf{x}, \mathbf{z})$ of size $256 \times 256 \times 64$.

The 1-layer NN is $\mathcal{N}_{\mathbf{w}, \mathbf{b}}$ where $\mathbf{w} = [\mathbf{W}_1, \mathbf{W}_{\text{out}}]$ and $\mathbf{b} = [\mathbf{b}_1, \mathbf{b}_{\text{out}}]$.

Training/test data and Tensorflow

Python/Tensorflow helps with solving the optimization problem

$$(\tilde{\mathbf{w}}, \tilde{\mathbf{b}}) = \min_{(\mathbf{w}, \mathbf{b})} ||\mathcal{N}_{\mathbf{w}, \mathbf{b}}(\mathbf{g}_{\text{train}}) - \mathbf{o}_{\text{train}}||$$

by providing different optimizer and the autodiff. Tensorflow basically builds a graph of computational operators and computes the derivative based on the graph.

```
def fct(x):  
    return sin(x*x) + cos(x - 1)  
  
def TFFunct(x):  
    return tf.sin(tf.multiply(x, x)) + tf.cos(x - 1)
```

Training data are used to solve the optimization problem above to get the best weights and bias. Test data are used to make sure the optimized weights and bias are good.

To prevent over-fitted weights and bias, training data and test data are stochastically taken (usually 80% - 20%) from the whole available data.

Using NN (with the physical model)




Let $\mathcal{N}_{\mathbf{w}, \mathbf{b}}$ be a neural network, $\mathcal{F}(o(\mathbf{x}, z))$ be the physical model, then one can do optimization on the training data (based on [1]):

$$\left(\tilde{\mathbf{w}}, \tilde{\mathbf{b}} \right) = \min_{(\mathbf{w}, \mathbf{b})} \left(||\mathcal{N}_{\mathbf{w}, \mathbf{b}}(g_{\text{train}}) - o_{\text{train}}|| + ||\mathcal{F}(\mathcal{N}_{\mathbf{w}, \mathbf{b}}(g_{\text{train}})) - g_{\text{train}}|| \right). \quad (2.3)$$

Then one can estimate the original object (given any test data set g_{test}) by:

$$\hat{o}(\mathbf{x}, z) = \mathcal{N}_{\tilde{\mathbf{w}}, \tilde{\mathbf{b}}}(g_{\text{test}}(\mathbf{x}, z)). \quad (2.4)$$

References

-  Karpatne, Anuj, William Watkins, Jordan S. Read and Vipin Kumar. "Physics-guided Neural Networks (PGNN): An Application in Lake Temperature Modeling." ArXiv abs/1710.11431 (2017): n. pag.
-  Y. Rivenson, Z. Göröcs, H. Günaydin, Y. Zhang, H. Wang, and A. Ozcan, "Deep learning microscopy," Optica 4, 1437-1443 (2017).
-  Yujia Xue, Shiyi Cheng, Yunzhe Li, and Lei Tian, "Reliable deep-learning-based phase imaging with uncertainty quantification", Optica 6, 618-629 (2019)