CSCI653 Course project

Using DeepONet to infer unknown parameters of partially-observable physical system

Chenchen Huang, Haotian Hang
Email address: chencheh@usc.edu, haotianh@usc.edu

December 14, 2022

1 Motivation

Previous work on physics informed neural network (PINN) can infer solution of given physics problems assuming the whole differential equation is known in advance [1, 2]. However, in most dynamics or fluid dynamics problem, though we can affirm the governing differential equation, some parameters might remain unknown. Meanwhile, in some experiments, there are limitations that we cannot measure all the states. Therefore, it is worthwhile developing an architecture that can solve or approximate the solution only based on limited information. Moreover, field data is difficult to measure in a large extend, usually what we can measure are time snapshots at sparse locations. This motivates us to find a method which can be helping in inferring both the parameter and the whole physics field in this situation.

For example, for a time-evolving physical system, a swimming fish in liquid, we can estimate the governing equation of the flow field, but the parameters are unknown. And we only can experimentally measure the velocity at limited locations, for example using PIV.

How to infer unknown parameters of partially-observable physical system? We propose this DeepONet in the following sections.

2 Objective

For a system with knowing governing equation $\mathcal{L}[\mathbf{u}(\mathbf{x},t);\boldsymbol{\alpha}]=0$, unknown parameters $\boldsymbol{\alpha}$, and some experimental or numerical measurements $\mathbf{u}_s$, we want to recover the full field and find the unknown parameters. Particularly, we are solving 1D phase separation evolution system. The Allen–Cahn equation is a mathematical model for phase separation processes. The equation describes the time evolution of a scalar-valued state variable u on 1D space domain x during a time interval t.

$$u_t = \epsilon \triangle u - f'(u), \tag{1}$$

where ϵ is a small number, and $f'(u)=\frac{df(u)}{du}$, with $f(u)=\frac{1}{4}(u^2-1)^2$, as a double-well potential.

Here, we consider dimensionless governing equation of state $u(x,t)$ on domain $x\in[-1,1],\ t\in[0,1]$ with periodic boundary conditions. Choose an initial state, we have the system

$$\begin{aligned} u_t &= \alpha_1 u_{xx} - \alpha_2(u^3-u),\\ u(t=0) &= x^2\cos(\pi x),\\ u(x=-1) &= u(x=1),\\ u_x(x=-1) &= u_x(x=1). \end{aligned} \tag{2}$$

where $\alpha_1 \ll 1$, and $\alpha_2 \gg \alpha_1$.

Assume we have some measurements of this system, and we want to use these measurements to recover the whole field and find the associated parameters set. The schematic diagram is in Fig.1; assume we only have spatially sparse distributed sensors along these four lines where the arrows are pointing at. Thus, we can measure the field at these four specific locations along the whole time.

The goal is using those measurements and above Allen-Cahn equation with IC&BCs to recover the field u and parameters α_1, α_2.

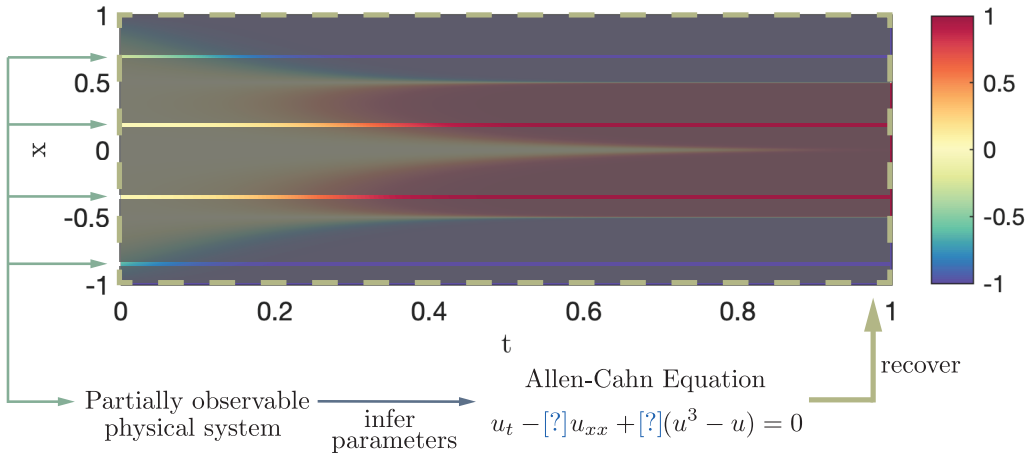

Figure 1: Schematic of sample problem

3 Deliverable

We will first train a Deep Operator Network using a dataset of the the Allen equation for a set of different parameters. And then, based on this pre-trained model, we will perform gradient descent using the mean squared error (MSE) of the solution and the measurement as loss function and solve for the parameters.

4 Research plan

4.1 Brief description of the algorithm

The algorithm can be decomposed into two parts, a deep operator network and a optimizer. The deep operator network is fitted using a dataset of Allen equation of a set of different parameters. After that, we pick the solution of Allen equation at sparse locations for a field which is not in the training set. We randomly give an initial guess of the parameters and use gradient descent to find the optimized parameter set which minimize the MSE between the output of the network and the measurement. The process is given in Algorithm 1

Algorithm 1 Algorithm

Initialize the Deep Operator Network
for i=0, 1,..., Maximum step **do**
    Update θ_b and θ_t to minimize Π_1
end for
Sample the measurement from the field of one set of parameter which we need to solve
Randomly set an initial value of parameters α_1 and α_2
for i=0, 1,..., Maximum step **do**
    Calculate the output of DeepONet at the given location for the given parameter set
    Use gradient decent to update the value of the parameters
end for

4.2 Architecture

The neural network is a Deep Operator Net, which contains a branch net for encoding the parameters and a trunk net for encoding the spatial and temporal information [3]. The outputs of the two parts are combined by dot product. The structure of the network is given in Figure 2. For more details, we refer the readers to read [3]. In our study, the input dimension of both trunk and branch

are both 2, and the shape of trunk and branch is exactly the same, which has 15 hidden layers of width 15. The dimension for the dot product is 30. We will discuss this hyper parameters choice in 4.4.

For the second part of optimization, there is no neural network, instead, the MSE loss is calculated based on the output of the DeepONet and the measured(observable) data.

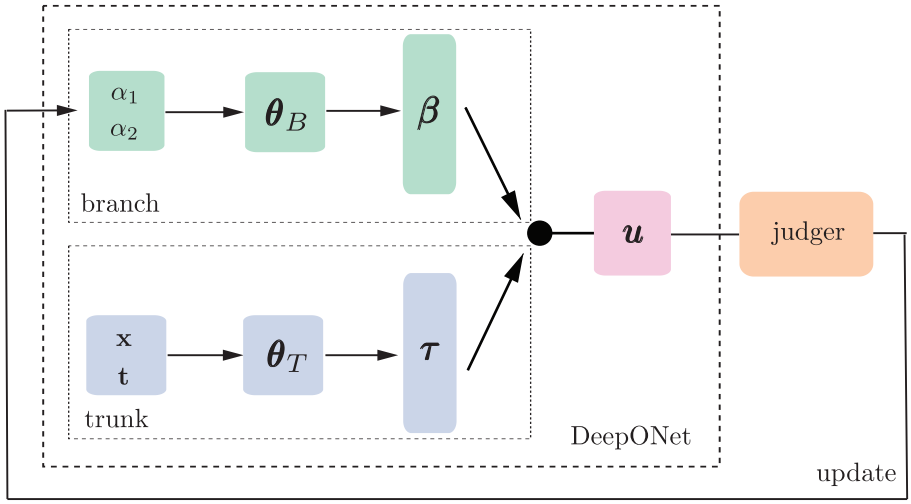

Figure 2: Basic architecture

4.3 Loss function

For the first part of training, the loss function is defined as MSE between the output of the neural network and the dataset.

$$\Pi_1(\theta_b, \theta_t) = \sum_{i \in \text{dataset}} ||u^{(i)} - \hat{u}(\alpha_1^{(i)}, \alpha_2^{(i)}, x^{(i)}, t^{(i)}; \theta_b, \theta_t)||^2 \tag{3}$$

For the second part of gradient descent, the loss is defined as MSE between the output of the neural network and the measurement. However, the trainable parameter is not the weight of the neural network, in stead, it is the parameter of the equation

$$\Pi_2(\alpha_1, \alpha_2) = \sum_{i \in \text{dataset}} ||u^{(i)} - \hat{u}(\alpha_1, \alpha_2, x^{(i)}, t^{(i)}; \theta_b, \theta_t)||^2 \tag{4}$$

4.4 Training

Next, we train a DeepONet that is able to recover the whole field by using some given measurements.

For the first part of training, the goal is to find the optimal value of the weights which minimize the loss function Π_1

$$\theta_b^*, \theta_t^* = \text{argmin}_{\theta_b, \theta_t} \Pi_1(\theta_b, \theta_t) \tag{5}$$

This part is trained by labeled data generated by a numerical solver. For the second part of training, the goal is to find the optimal value of α_1 and α_2 which minimize the loss function Π_2

$$\alpha_1^*, \alpha_2^* = \text{argmin}_{\alpha_1, \alpha_2} \Pi_1(\alpha_1, \alpha_2) \tag{6}$$

We use the MATLAB built-in ode45 as numerical solver to generate training datasets. We discrete the space and time with number of nodes $N_t = N_x = N = 1000$. The parameters set ranges of our dataset are $\alpha_1 \in [0.0001, 0.1]$, $\alpha_2 \in [1, 10]$. To formulate the problem as an ODE, we write

$$\frac{\partial u}{\partial t} = \alpha_1 \frac{\partial^2 u}{\partial x^2} - \alpha_2(u^3 - u), \tag{7}$$

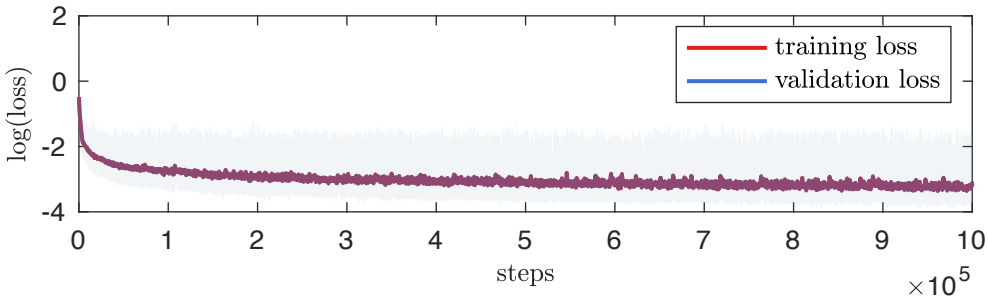

Figure 3: Training loss and validation loss of DeepONet

where the Laplacian term is written as discrete form using central difference. Substitute the periodic boundary conditions, we can write the discrete form of central points and boundary points as

$$\frac{\partial^2 u_j}{\partial x^2} = \frac{u_{j+1} - 2u_j + u_{j-1}}{(\triangle x)^2}, \frac{\partial^2 u_1}{\partial x^2} = \frac{u_2 - 2u_1 + u_N}{(\triangle x)^2}, \frac{\partial^2 u_N}{\partial x^2} = \frac{u_1 - 2u_N + u_{N-1}}{(\triangle x)^2} \tag{8}$$

4.5 Validation

For all the dataset we calculated, we use 20% of the fields as the validation data and 30% as the training data. We tested several different sizes of Neural Network and determined the optimistic setup. (See appendix)

Since we are not using the validation data for hyperparameter turning, after the first training, we can use the the validation set to validate our DeepONet is neither overfitting nor underfitting.

4.6 Testing

After training, we test trained model by our testing datasets which are seperated from training datasets. As shown in Figure 4, our Deep Operator Network can fit the data well.

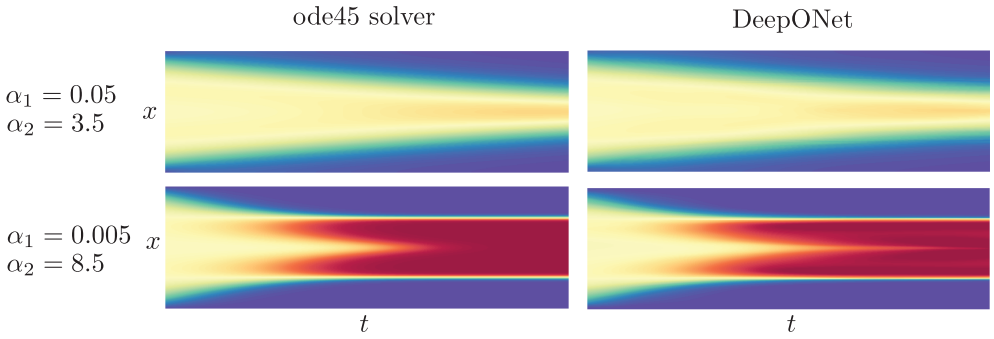

Figure 4: Inferred parameters based on partial observation

5 Results and Discussion

We first inspect the performance of DeepONet as in Figure 4. By comparing the output of DeepONet and the data in our dataset but not in training set, we found that the DeepONet can reproduce the field of great accuracy. The region where the output is not very accurate is as shows in the second case, when the structure of the field is inherently a two peak mode, the output of the DeepONet will elongate the valley in the middle for longer time.

The performance of our method in inferring unknown parameters is given in Figure 6 and Figure 7. We use two metrics to evaluate its performance, the first metrics is the Euclidean distance between the inferred parameter and the actual parameter. For the second metrics, we use our inferred parameters

to recover the whole field, and calculate the mean square error between the recovered field and the actual data. As shown in these two figures, we can find that for both cases, our method is very accurate in terms of both metrics except for the boundary of the parameter space. Interestingly, the biggest error for two metrics does not happen at the same location in the parameter space. That is because in some location in the parameter space, the accurate numerical solution is not sensitive to the parameters, and in other locations, they are very sensitive.

By comparing the results shown in Figure 6 and Figure 7, we found that the with only 0.5% of the data, our method is able to recover the unknown parameters and the whole physical field simultaneously. Ironically, the result inferred from partially observable system is a little bit more accurate than the fully observable system although this conclusion is not clearly illustrated in the figures. Probably it is because the imperfection of the DeepONet. Giving it full data as loss will let the optimizer be distracted by these small disturbance.

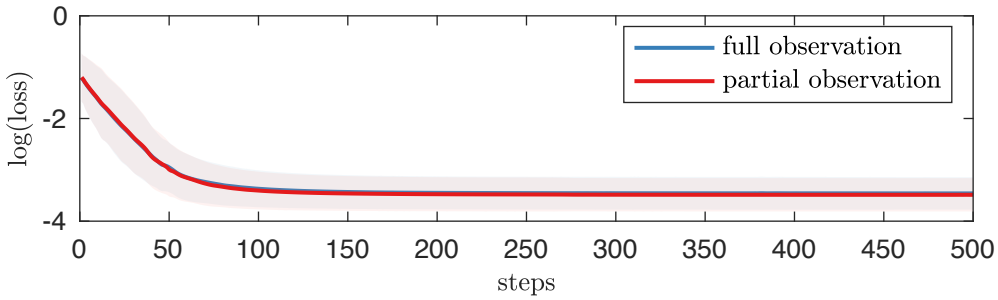

Figure 5: Optimization loss with full observation and partial observation

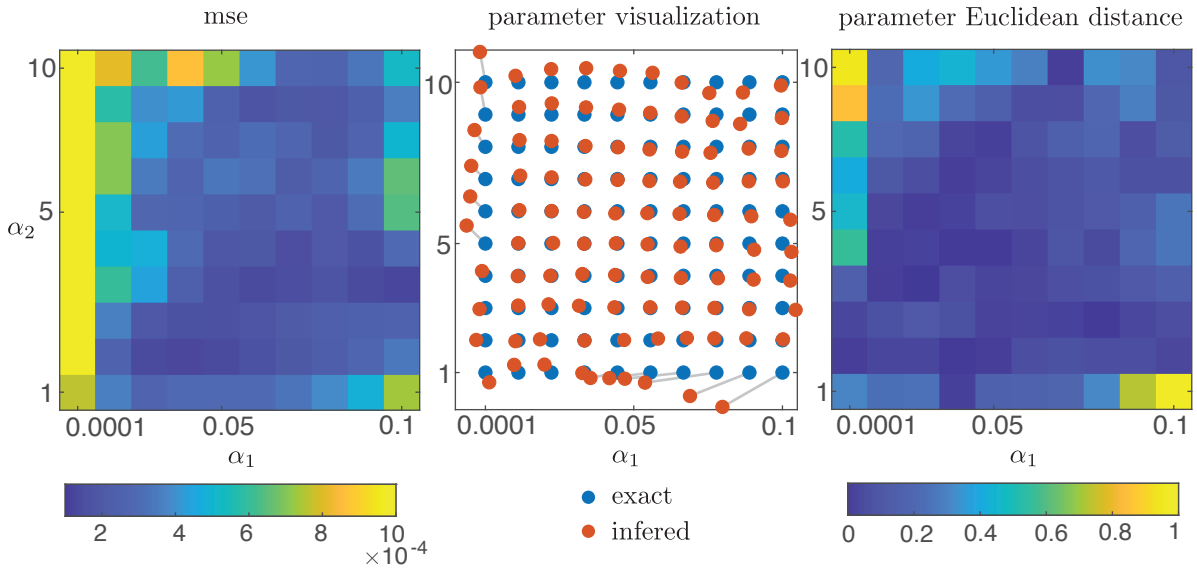

Figure 6: Inferred parameters based on full observation

6 Conclusions and Future work

Based on our work, there are several conclusions.

Firstly, using DeepONet, we are capable of fitting the Allen–Cahn equation accurately.

Secondly, based on the trained DeepONet, we can use gradient descent to find the unknown parameters from measurement which contains only 0.5% of the whole field data.

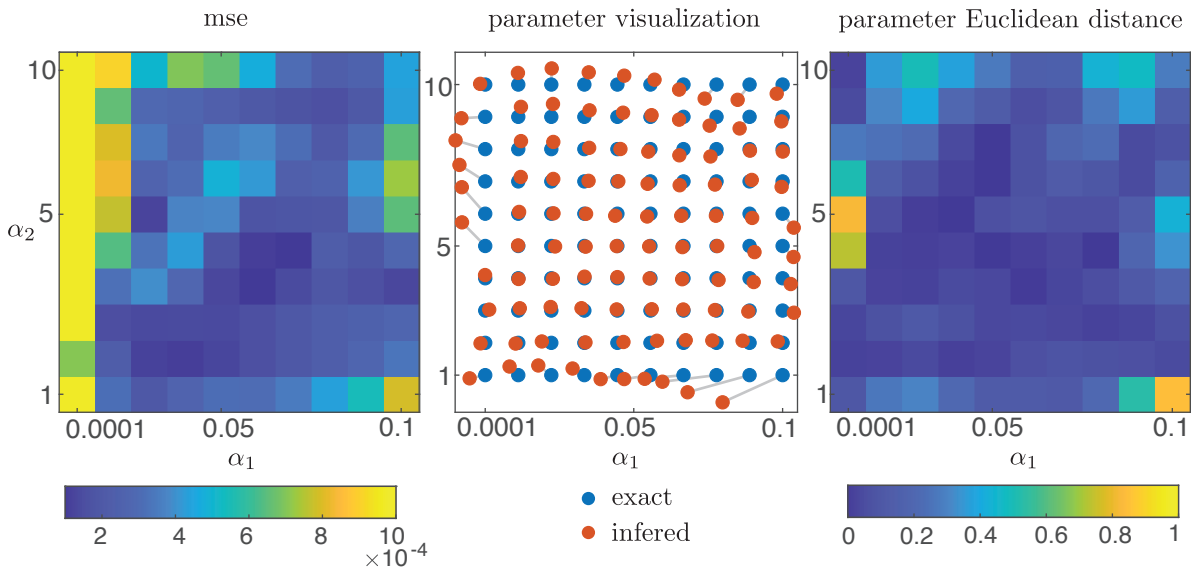

Figure 7: Inferred parameters based on partial observation (only 0.5% of the dataset)

For now, the second part is done on a linearly-transformed space of the parameter space. For future work, inspired by the results we have now, we are going to do a nonlinear transformation to make more emphasis on the region in parameter space where the solution is most sensitive to parameters. Furthermore, we also implemented to use Constrained optimization provided by TFCO (tensorflow constraint optimization) in this study. However, limited by our computational resources, we are unable to use it.

Another direction of extending this work is for the DeepONet part, we can change the loss function to not only include the MSE with the training data, but also like a Physics Informed Neural Network(PINN), directly use the equation as the loss of the neural network.

References

[1] Maziar Raissi, Alireza Yazdani, and George Em Karniadakis. Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations. *Science*, 367(6481):1026–1030, 2020.

[2] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

[3] Lu Lu, Pengzhan Jin, and George Em Karniadakis. Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. *arXiv preprint arXiv:1910.03193*, 2019.

Appendices

Some examples of dataset is given in Figure 8.

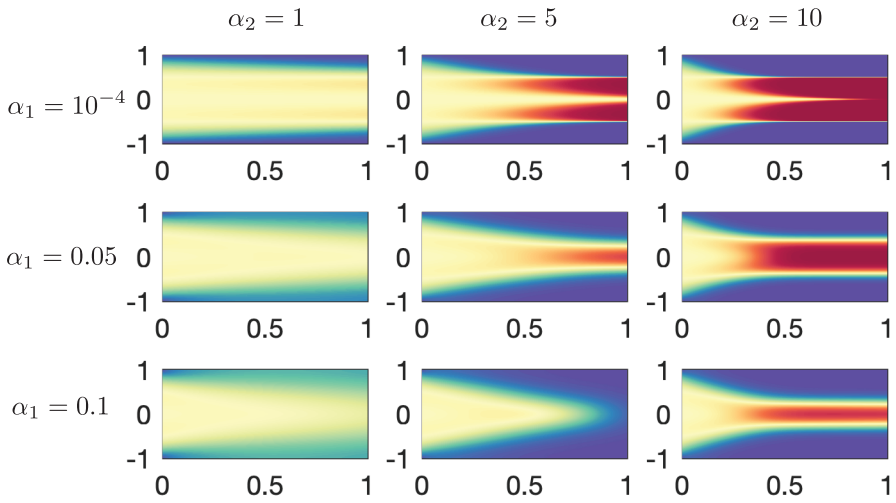

Figure 8: Examples of dataset

As shown in Figure 9, we compared four different sets of hyperparameters of DeepONet. With $15 \times 15(30)$, which has 15 hidden layers of width 15 and the dimension for the dot product is 30, the performance is close to $20 \times 20(40)$. However, the training cost of $20 \times 20(40)$ is higher. Ironically, with even larger dimension of Neural Networks, models are not able to provide accurate approximation of solution. Hence, we choose $15 \times 15(30)$ as our model setup.

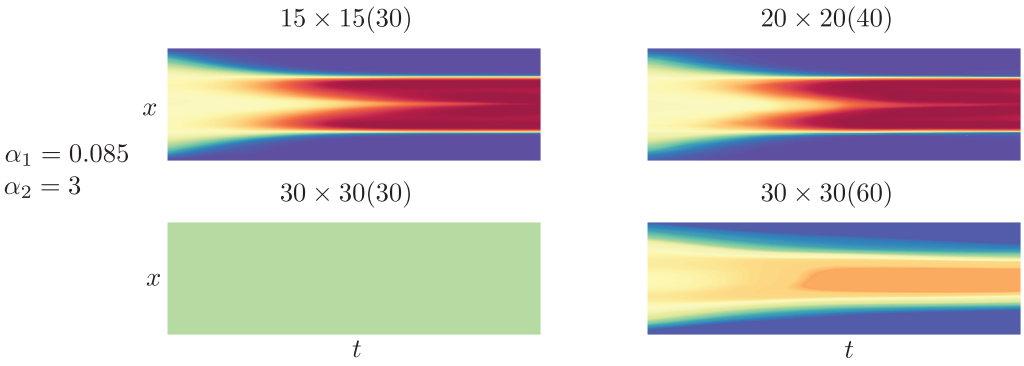

Figure 9: Hyperparameters comparison