One-shot learning for solution operators of partial differential equations

Lu Lu
Department of Mathematics
Massachusetts Institute of Technology
Cambridge, MA 02139, USA
lu_lu@mit.edu

Haiyang He, Priya Kasimbeg, Rishikesh Ranade & Jay Pathak
Ansys Inc.
San Jose, CA, USA

Abstract

Discovering governing equations of a physical system, represented by partial differential equations (PDEs), from data is a central challenge in a variety of areas of science and engineering. Current methods require either some prior knowledge (e.g., candidate PDE terms) to discover the PDE form, or a large dataset to learn a surrogate model of the PDE solution operator. Here, we propose the first learning method that only needs one PDE solution, i.e., one-shot learning. We first decompose the entire computational domain into small domains, where we learn a local solution operator, and then find the coupled solution via a fixed-point iteration. We demonstrate the effectiveness of our method on different PDEs, and our method exhibits a strong generalization property.

1 Introduction

Discovering governing equations of a physical system from data is a central challenge in a variety of areas of science and engineering. These governing equations are usually represented by partial differential equations (PDEs). For the first scenario, where we know all the terms of the PDE and only need to infer unknown coefficients from data, many effective methods have been proposed. For example, we can enforce physics-based constraints to train neural networks that learn the underlying physics (Pang et al., 2019; Zhang et al., 2019; Chen et al., 2020a; Yazdani et al., 2020; Rao et al., 2020; Wu et al., 2018; Qian et al., 2020; Lu et al., 2021b). In the second scenario, we do not know all the PDE terms, but we have a prior knowledge of all possible candidate terms, several approaches have also developed recently (Brunton et al., 2016; Rudy et al., 2017; Chen et al., 2020b).

In a general setup, discovering PDEs only from data without any prior knowledge is much more difficult. To address this challenge, instead of discovering the PDE in an explicit form, in most practical cases, it is sufficient to have a surrogate model of the PDE solution operator that can predict PDE solutions repeatedly for different conditions (e.g., initial conditions). Very recently, several approaches have been developed to learn PDE solution operators by using neural networks such as DeepONet (Lu et al., 2021a) and Fourier neural operator (Li et al., 2020). However, these approaches require large amounts of data to train the networks.

In this work, we propose a novel approach to learn PDE solution operators from only one data point, i.e., one-shot learning. To our knowledge, the use of one-shot learning in this space is very limited. Wang et al. (2019) and Fang et al. (2017) used few shot learning to learn PDEs and applied it to face recognition. Our method leverages the locality of PDEs and use neural networks to learn the system governed by PDEs at a small computational domain. Then for a new PDE condition, a fixed point iterative algorithm is proposed to couple all local domains to find the PDE solution. Our method exhibits strong generalization properties. Moreover, the one-shot local learning approach trains very fast and extends to multi-dimensional, linear and non-linear PDEs. In this paper, we describe, in detail, the one-shot local learning approach and demonstrate on different PDEs for a range of conditions.

2 Methods

We first introduce the problem setup of learning solution operators of partial differential equations (PDEs) and then present our one-shot learning method.

2.1 Learning solution operators of PDEs

We consider a physical system governed by a PDE defined on a spatio-temporal domain $\Omega \subset \mathbb{R}^d$:

$$\mathcal{F}[u(\mathbf{x}); f(\mathbf{x})] = 0, \quad \mathbf{x} = (x_1, x_2, \ldots, x_d) \in \Omega$$

with suitable initial and boundary conditions. $u(\mathbf{x})$ is the solution of the PDE and $f(\mathbf{x})$ is a forcing term. The solution u depends on f, and thus we define the solution operator as $\mathcal{G} : f(\mathbf{x}) \mapsto u(\mathbf{x})$. For nonlinear PDEs, $\mathcal{G}$ is a nonlinear operator.

In many problems, the PDE of a physical system is unknown or computationally expensive to solve, and instead, sparse data representing the physical system is available. Specifically, we consider a dataset $\mathcal{T} = \{(f_i, u_i)\}_{i=1}^{|\mathcal{T}|}$, and (f_i, u_i) is the i-th data point, where $u_i = \mathcal{G}(f_i)$ is the PDE solution for f_i. Our goal is to learn $\mathcal{G}$ from the training dataset $\mathcal{T}$, such that for a new f, we can predict the corresponding solution $u = \mathcal{G}(f)$. When $\mathcal{T}$ is sufficiently large, then we can learn $\mathcal{G}$ straightforwardly by using neural networks, whose input and output are f and u, respectively. Many networks have been proposed in this manner such as DeepONet (Lu et al., 2021a) and Fourier neural operator (Li et al., 2020). In this study, we consider an extreme scenario where we have only one data point for training, i.e., one-shot learning with $|\mathcal{T}| = 1$, and we let $\mathcal{T} = \{(f_\mathcal{T}, u_\mathcal{T})\}$. Learning from only one data point is impossible in general, and here we consider that $\mathcal{T}$ is not given, and we can select $f_\mathcal{T}$. In addition, instead of learning $\mathcal{G}$ for the entire input space, we only predict f in a neighborhood of some f_0, where we know the solution $u_0 = \mathcal{G}(f_0)$.

2.2 One-shot learning based on locality

To overcome the difficulty of training a machine learning model based on only one data point, we consider the fact that derivatives and PDEs are defined locally, i.e., the same PDE is satisfied in an arbitrary small domain inside Ω. In our method, we partition the entire domain Ω into many small domains, i.e., a mesh of Ω. In this study, we only consider the structured equispaced grid to demonstrate our method, but our method can be easily extended to unstructured mesh.

Learning the local solution operator via a neural network. To demonstrate the idea, we consider a mesh node at the location $\mathbf{x}^*$ (the red node in Fig. 1). In order to predict the solution $u(\mathbf{x}^*)$, instead of considering the entire computational domain, we only consider a small domain $\tilde{\Omega}$ surrounding $\mathbf{x}^*$ with several mesh cells. If we know the solution u at the boundary of $\tilde{\Omega}$ ($\partial\tilde{\Omega}$) and f within $\tilde{\Omega}$, then $u(\mathbf{x}^*)$ is determined by the PDE. Here, we use a neural network to represent this relationship $\tilde{\mathcal{G}} : \{u(\mathbf{x}) : \mathbf{x} \in \partial\tilde{\Omega}\} \cup \{f(\mathbf{x}) : \mathbf{x} \in \tilde{\Omega}\} \mapsto u(\mathbf{x}^*)$. In addition, considering the flexibility of neural networks, we may use other local information as network inputs. For example, we can only use the value of f at $\mathbf{x}^*$ and add more solutions inside $\tilde{\Omega}$ as network inputs: $\tilde{\mathcal{G}} : \{u(\mathbf{x}) : \mathbf{x} \in \tilde{\Omega} \text{ and } \mathbf{x} \neq \mathbf{x}^*\} \cup \{f(\mathbf{x}^*)\} \mapsto u(\mathbf{x}^*)$. The size and shape of $\tilde{\Omega}$ are also hyperparameters to be chosen. We will compare the performance of several different choices of network inputs in our numerical experiments.

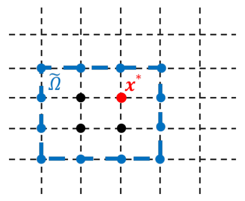

Figure 1: **PDE in a local domain $\tilde{\Omega}$.** A neural network $\tilde{\mathcal{G}}$ is trained to learn the mapping from all or some of $u(\mathbf{x})$ and $f(\mathbf{x})$ for $\mathbf{x} \in \tilde{\Omega}$ to $u(\mathbf{x}^*)$.

Because the network learns for a small local domain, by traversing the entire domain Ω, we can generate many input-output pairs for training the network, which makes it possible to learn a network from only one PDE solution. In addition, to make the network generalizable to different f, the selection of $f_\mathcal{T}$ in $\mathcal{T}$ is also important. In this study, we choose $f_\mathcal{T}$ to be uniform random between -1 and 1 on each mesh node, i.e., $f_\mathcal{T}(\mathbf{x})$ is sampled from $U(-1, 1)$. The reason is that if $f_\mathcal{T}$ is smooth, then the training data points in adjacent meshes become similar, and thus the "effective" number of training data points is small, while a randomly sampled $f_\mathcal{T}$ can induce a more "diverse" training dataset. The choice of a random $f_\mathcal{T}$ is critical for the learning as we will elaborate on this in the numerical examples.

Prediction via a fixed-point iteration. For a new $f = f_0 + \Delta f$, we cannot directly predict its solution u using the trained network, because the network inputs also include the solution to be predicted. Here, we propose an iterative algorithm to find the solution (Algorithm 1). Because f is close to f_0, we use u_0 as the initial guess of u, and then in each iteration, we apply the trained network on the current solution as the input to get a new solution. This algorithm can be viewed as a fixed-point iteration. When the solution is converged, u and f are consistent with respect to the local operator $\mathcal{G}$, and thus the current u is the solution of our PDE.

Algorithm 1: Predicting the solution $u = \mathcal{G}(f)$ for a new f.

```
Initiate: $u(\mathbf{x}) \leftarrow u_0(\mathbf{x})$ for all $\mathbf{x} \in \Omega$
while $u$ has not converged do
    for $\mathbf{x} \in \Omega$ do
        $\hat{u}(\mathbf{x}) \leftarrow \tilde{\mathcal{G}}$(the inputs of $u$ and $f$ in $\tilde{\Omega}$)
    Update: $u(\mathbf{x}) \leftarrow \hat{u}(\mathbf{x})$ for all $\mathbf{x} \in \Omega$
```

3 Results

In this section, we show the effectiveness of our proposed method for a few problems, and compare the accuracy and convergence rate of different choices of the local solution operator $\tilde{\mathcal{G}}$.

3.1 Poisson equation

We first consider a pedagogical example of a one-dimensional Poisson equation $\Delta u = f(x), \quad x \in [0, 1]$, with the zero Dirichlet boundary condition $u(0) = u(1) = 0$, and the solution operator is $\mathcal{G} : f \mapsto u$. We consider an equispaced grid with a step size $h = 0.01$, and choose the simplest local operator as $\tilde{\mathcal{G}} : [u(x_{i-1}), u(x_{i+1}), f(x_i)] \mapsto u(x_i)$ with $x_i = ih$, i.e., $\tilde{\Omega}$ only has three nodes and we use $u(x_{i-1})$, $u(x_{i+1})$ and $f(x_i)$ to predict $u(x_i)$. We use a single layer linear network for $\tilde{\mathcal{G}}$, and the training dataset $\mathcal{T}$ only has one data point $(f_\mathcal{T}, \mathcal{G}(f_\mathcal{T}))$, where $f_\mathcal{T}(x_i)$ is sampled randomly in $U(-1, 1)$. One example of $\mathcal{T}$ is shown in Figs. 2A and B, and we use the finite difference method (FDM) to compute the numerical solution.

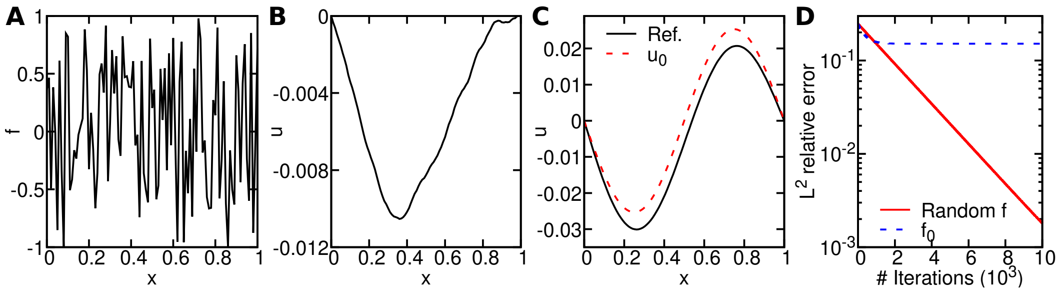

Figure 2: **Learning the Poisson equation.** (**A** and **B**) (A) A random $f_\mathcal{T}$ and (B) the corresponding solution $u_\mathcal{T}$ used for training. (**C**) The initial guess of u_0 has a L^2 relative error of 25% compared to the reference solution. (**D**) Red solid line: The L^2 relative error decreases exponentially during the iteration, when the network is trained on a random f. Blue dash line: The error is large when the network is trained on f_0.

After the network is trained, we use the iterative Algorithm 1 to predict the solution for $f = \sin(2\pi x) + 0.05$. We assume we have the solution $u_0 = -\frac{1}{4\pi^2}\sin(2\pi x)$ for $f_0 = \sin(2\pi x)$, i.e., $\Delta f = 0.05$, and then u_0 is used as the initial guess of the iteration, which has an L^2 relative error of 25% compared to the reference solution (Fig. 2C). During the iteration, the error of the solution decreases exponentially with respect to the number of iterations (Fig. 2D red solid line), and it requires about 6000 iterations to reach 1% error.

We notice that the learned linear network is $u(x_i) = \tilde{\mathcal{G}}(u(x_{i-1}), u(x_{i+1}), f(x_i)) = [u(x_{i-1}), u(x_{i+1}), f(x_i)] \cdot [0.5, 0.5, -5 \times 10^{-5}] + 10^{-9}$, which is the second-order finite difference scheme for the Poisson equation. However, as we will show in other examples, the network does not learn a finite difference scheme in general, especially for nonlinear PDEs or the local domain $\tilde{\Omega}$ with more nodes. To show the necessity of using a random f for training, we also use a smooth f_0 and u_0 to train the network, and the error can only decrease to 15% (Fig. 2D blue dash line). Although, both $f(x)$ and $f_0(x)$ span from -1 to 1, the noisy f generates a diverse training set and thus improves generalizability of the neural network.

3.2 Nonlinear diffusion-reaction equation

We consider a nonlinear diffusion-reaction equation with a source term $f(x)$: $\frac{\partial u}{\partial t} = D\frac{\partial^2 u}{\partial x^2} + ku^2 + f(x), \quad x \in [0,1], t \in [0,1]$, with zero initial and boundary conditions, where $D = 0.01$ is the diffusion coefficient, and $k = 0.01$ is the reaction rate. The solution operator is the mapping from $f(x)$ to $u(x,t)$. We use a grid with $\Delta x = \Delta t = 0.01$. To generate the training dataset, $f_{\mathcal{T}}(x)$ is randomly sampled from $U(0,1)$ (Fig. 5A), and the reference solution (Fig. 5B) is obtained from finite differences with the Crank-Nicolson method.

In this example, we consider different choices of the local domain $\tilde{\Omega}$ and the local solution operator $\tilde{\mathcal{G}}$. For the target u_i^j at $(i\Delta x, j\Delta t)$, the simplest choice of $\tilde{\Omega}$ is $\{(i, j-1), (i-1, j), (i, j), (i+1, j)\}$ (Fig. 3A), and then the simplest choice of $\tilde{\mathcal{G}}$ is $\tilde{\mathcal{G}}_1 : [u_i^{j-1}, u_{i-1}^j, u_{i+1}^j, f_i^j] \mapsto u_i^j$. We also consider a larger domain of 6 nodes as shown in (Fig. 3B), and choose $\tilde{\mathcal{G}}_2 : [u_{i-1}^{j-1}, u_i^{j-1}, u_{i+1}^{j-1}, u_{i-1}^j, u_{i+1}^j, f_i^j] \mapsto u_i^j$. After we train the networks $\tilde{\mathcal{G}}_1$ and $\tilde{\mathcal{G}}_2$, we predict the solution for f in the neighborhood of $f_0 = 0.9\sin(2\pi x)$, and the solution $u_0 = \mathcal{G}(f_0)$ is used as the initial guess of the iteration. Instead of using a constant Δf, we randomly sample Δf from a mean-zero Gaussian random field (GRF): $\Delta f \sim \mathcal{GP}(0, k(x_1, x_2))$, where the covariance kernel $k(x_1, x_2) = \sigma^2 \exp(-\|x_1 - x_2\|^2/2l^2)$ has a standard deviation σ and a correlation length l. Examples of some random f with different σ and l are shown in Fig. 6.

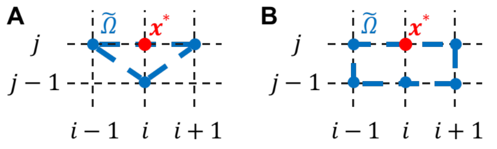

Figure 3: **Local domains $\tilde{\Omega}$ of the diffusion-reaction equation.** (**A**) Domain with 4 nodes. (**B**) Domain with 6 nodes.

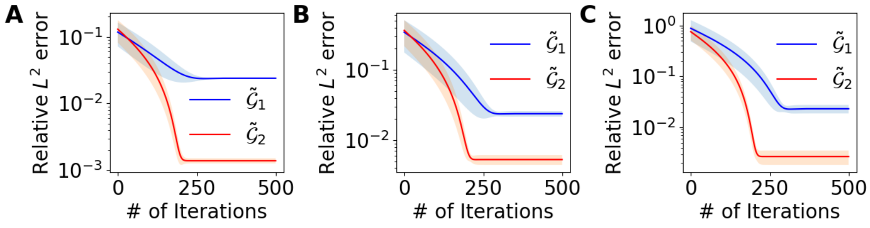

Figure 4: **Learning the diffusion-reaction equation.** (**A**) L^2 relative errors of $\tilde{\mathcal{G}}_1$ and $\tilde{\mathcal{G}}_2$ tested on 100 random Δf sampled from GRF of $\sigma = 0.1$ and $l = 0.1$. (**B**) $\sigma = 0.3$. (**C**) $\sigma = 1$. The shaded band is one standard deviation.

We first test $\tilde{\mathcal{G}}_1$ and $\tilde{\mathcal{G}}_2$ on 100 random Δf sampled from a GRF with $\sigma = 0.1$ and $l = 0.1$. The L^2 relative error of $\tilde{\mathcal{G}}_1$ is more than one-order larger than the error of $\tilde{\mathcal{G}}_2$ (Fig. 4A). If we increse the magnitude of Δf to $\sigma = 0.3$ or 1, $\tilde{\mathcal{G}}_2$ always has better performance than $\tilde{\mathcal{G}}_1$ (Figs. 4B and C). We note that when Δf is sampled from GRF with $\sigma = 1$, f and f_0 are significantly different (Fig. 6C), but $\tilde{\mathcal{G}}_2$ can still achieve an L^2 relative error of 0.3% in about 200 iterations, which demonstrates the robustness and generalizability of our proposed method.

4 Conclusion

In this study, we propose, to the best of our knowledge, the first one-shot method to learn solution operators from only one PDE solution. In future, we will carry out further validation of on different types of PDEs, extend the method to unstructured meshes, and improve Algorithm 1 for faster convergence and better computational efficiency.

REFERENCES

S. L. Brunton, J. L. Proctor, and J. N. Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 113(15):3932–3937, 2016.

Y. Chen, L. Lu, G. E. Karniadakis, and L. Dal Negro. Physics-informed neural networks for inverse problems in nano-optics and metamaterials. *Optics Express*, 28(8):11618–11633, 2020a.

Z. Chen, Y. Liu, and H. Sun. Deep learning of physical laws from scarce data. *arXiv preprint arXiv:2005.03448*, 2020b.

C. Fang, Z. Zhao, P. Zhou, and Z. Lin. Feature learning via partial differential equation with applications to face recognition. *Pattern Recognition*, 69:14–25, 2017.

Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020.

L. Lu, P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3): 218–229, 2021a.

L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis. DeepXDE: A deep learning library for solving differential equations. *SIAM Review*, 63(1):208–228, 2021b.

G. Pang, L. Lu, and G. E. Karniadakis. fPINNs: Fractional physics-informed neural networks. *SIAM Journal on Scientific Computing*, 41(4):A2603–A2626, 2019.

E. Qian, B. Kramer, B. Peherstorfer, and K. Willcox. Lift & learn: Physics-informed machine learning for large-scale nonlinear dynamical systems. *Physica D: Nonlinear Phenomena*, 406: 132401, 2020.

C. Rao, H. Sun, and Y. Liu. Physics-informed deep learning for incompressible laminar flows. *Theoretical and Applied Mechanics Letters*, 10(3):207–212, 2020.

S. H. Rudy, S. L. Brunton, J. L. Proctor, and J. N. Kutz. Data-driven discovery of partial differential equations. *Science Advances*, 3(4):e1602614, 2017.

H. Wang, Z. Zhao, and Y. Tang. An effective few-shot learning approach via location-dependent partial differential equation. *Knowledge and Information Systems*, pp. 1–21, 2019.

J. Wu, H. Xiao, and E. Paterson. Physics-informed machine learning approach for augmenting turbulence models: A comprehensive framework. *Physical Review Fluids*, 3(7):074602, 2018.

A. Yazdani, L. Lu, M. Raissi, and G. E. Karniadakis. Systems biology informed deep learning for inferring parameters and hidden dynamics. *PLoS Computational Biology*, 16(11):e1007575, 2020.

D. Zhang, L. Lu, L. Guo, and G. E. Karniadakis. Quantifying total uncertainty in physics-informed neural networks for solving forward and inverse stochastic problems. *Journal of Computational Physics*, 397:108850, 2019.

A DIFFUSION-REACTION EQUATION

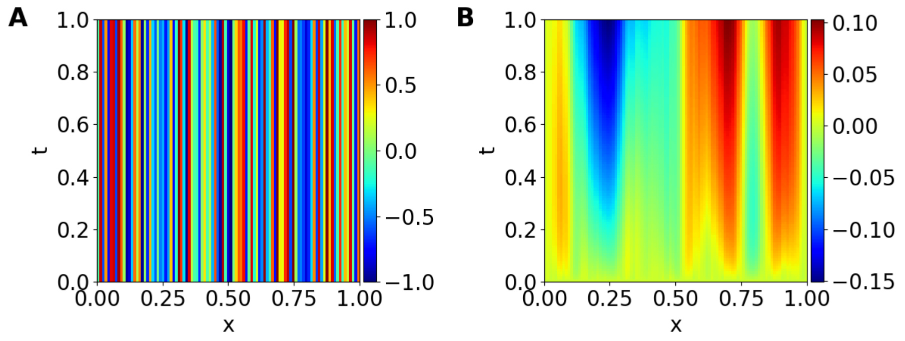

Figure 5: **Training data for the diffusion-reaction equation.** (**A**) A random $f_{\mathcal{T}}$. (**B**) The corresponding solution $u_{\mathcal{T}}$.

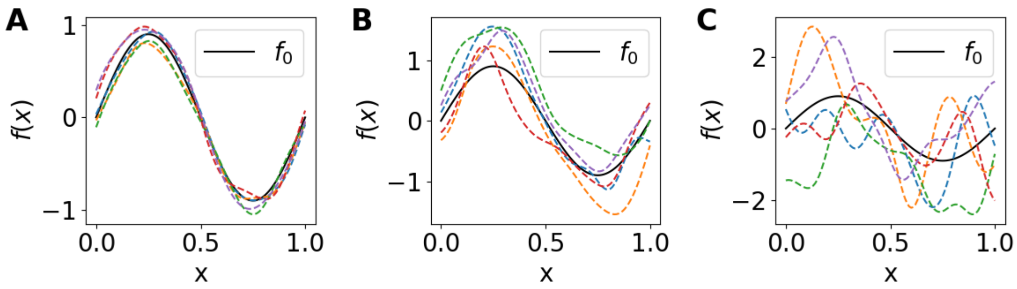

Figure 6: **Examples of random $f = f_0 + \Delta f$ with Δf sampled from GRF of different σ and $l = 0.1$.** (**A**) $\sigma = 0.1$. (**B**) $\sigma = 0.3$. (**C**) $\sigma = 1$. The black solid line is f_0.