Survey on Learning Operators - Focused on DeepONet

Hyemin Gu

hgu@umass.edu

November 5, 2021

Outline

PDEs and Neural Network Approaches

A PDE and Related Problems

A *diffusion-reaction system with a source term* $u(x)$ is given as

$$\frac{\partial s}{\partial t} = D\frac{\partial^2 s}{\partial x^2} + ks^2 + u(x),\ x \in [0,1], t \in [0,1] \tag{1}$$

with zero initial boundary conditions, where D is the diffusion coefficient, and k is the reaction rate.
We can consider

- Forward problem
- Inverse problem (given observations of $s(t,x)$ at some points, find the parameters D, k)
- Inferring the equation given observations of $s(t,x)$ only.

Conventional Numerical Solvers

Conventional numerical solvers for PDEs: FDM, FEM, FVM generally obtain solutions by discretization of domain/equation.

1. There is a tradeoff between resolution and accuracy.
   Computationally costly to obtain more accurate solution.
2. There is no universal method that works for any equations.
   e.g. CFL condition of methods for hyperbolic PDEs
3. The solution is a finite dimensional approximation.

Neural Network Approaches

Solving PDEs as well as ODEs using Neural Network is based on the theorem on 2 layer Neural Network.

Theorem (Universal Approximation Theorem (Cybenko, 1989))

σ is a sigmoidal function, $I_d \subset \mathbb{R}^d$ is the d-dimensional unit cube $[0,1]^d$, and $C(I_d)$ is the space of continuous functions on I_d.
Given $f \in C(I_d)$ and $\epsilon > 0$, there are $n \in \mathbb{N}$, m points $x_1, \cdots, x_m \in I_d$, and constants $c_i, \theta_i, \xi_{ij} \in \mathbb{R}$, $i = 1, \cdots, n$, $j = 1, \cdots m$, such that

$$\left| f(x) - \sum_{i=1}^{n} c_i \sigma \left(\sum_{j=1}^{m} \xi_{ij} f(x_j) + \theta_i \right) \right| < \epsilon \tag{2}$$

holds for all $x \in I_d$.

People empirically extrapolate this result on Deep Neural Networks.

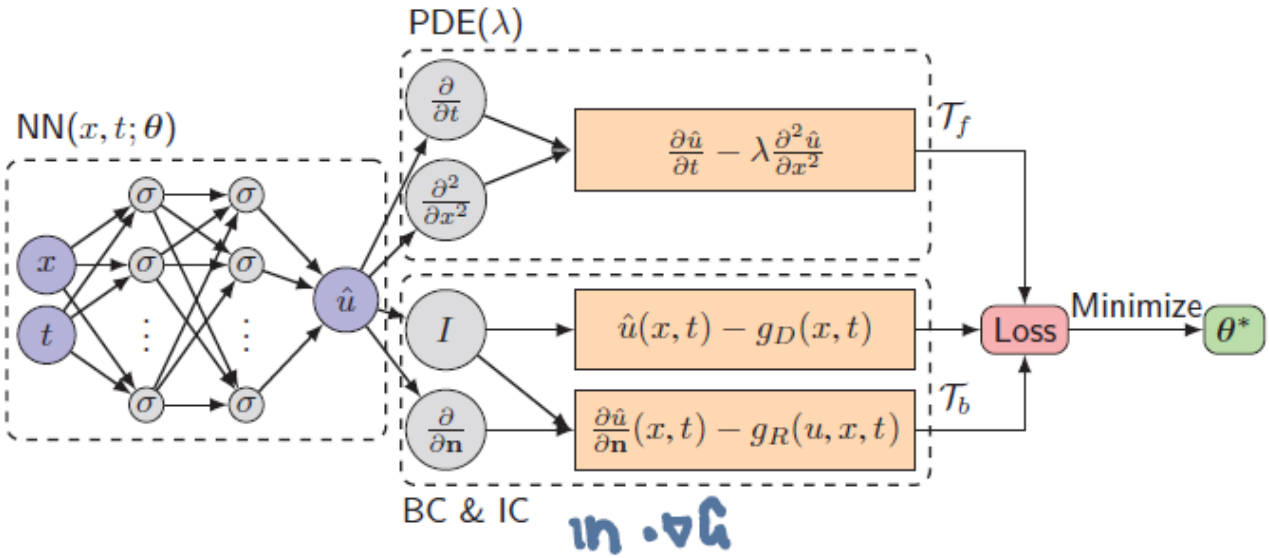
Numerical Solvers vs. Neural Network Approaches

The distinguished attributes between conventional numerical solvers and Neural Network approaches are summarized as below.

Conventional Solvers	**Attributes**	**NNs**
Linear Approximation	**Derivatives**	Automatic Differentiation
Mostly grid points	**Input**	Grid-free
Approximation on grid points	**Output**	Solution function of an instance
Rule-based	**Approach**	Data-driven
Long	**Time to evaluate**	Short

Physics Informed Neural Networks

Physics Informed Neural Networks(PINN) (Karniadakis group[1]) is one of the significant Neural Network approaches to solve differential equations by directly imposing the DE, ICs and BCs to the Loss function.



Applications of PINNs are :

- inverse problems
- intero-differential equations
- fractional differential equations
- stochastic differential equations

Learning Operators

* Still problematic!

- one instant solution function for a DE with a fixed parameter set
- finite dimensional approximation of f at $\{x_1, x_2, \cdots, x_m\}$

Consider learning an operator $G : u(x) \mapsto \mathbf{s}(x)$, $x \in [a, b]$.

Notations

- $V \subset C(X)$ where X Banach, and $W \subset C(\mathbb{R}^d)$ are some compact function spaces.
- $G : V \to W$ an operator that maps a function u to a function $G(u)$.
- For any point y in the domain of $G(u)$, $G(u)(y) \in \mathbb{R}$.
- An input function u is configured by its evaluations at finitely many sensors $\{x_1, x_2, \cdots, x_m\}$.

Examples of the Operator G

Operator G : function (∞-dim) $\longmapsto$ function (∞-dim)

- derivative: $u(x) \mapsto u'(x)$
- integral: $u(x) \mapsto \int K(y,x)u(y)dy$
- dynamical system:
  an external force term to the position of a gravity pendulum
  $\frac{ds_1}{dt} = s_2 \quad \frac{ds_2}{dt} = -k \sin s_1 + u(t)$
- PDE:
  a source term $u(x)$ to the solution $\mathbf{s}(t,x)$
  $\frac{\partial s}{\partial t} = D\frac{\partial^2 s}{\partial x^2} + ks^2 + u(x),\ x \in [0,1], t \in [0,1]$

Let's see a concrete form of G in our interested case!

Operator G for Dynamical systems

WANT: to predict $\mathbf{s}(x)$ over the whole domain $[a, b]$ for any $u(x)$.
A dynamical system on $\mathbf{s} : [a, b] \to \mathbb{R}^k$,

$$\frac{d\mathbf{s}}{dx} = \mathbf{g}(\mathbf{s}(x), u(x), x), \ x \in [a, b] \tag{3}$$

with $\mathbf{s}(a) = \mathbf{s_0}$, where $u \in V$ is a compact subset of $C[a, b]$.

What NN learns

G: the operator mapping the input u to the output $\mathbf{s}$ i.e., Gu satisfies

$$(Gu)(x) = \mathbf{s_0} + \int_a^x \mathbf{g}((Gu)(t), u(t), t)dt. \tag{4}$$

Conceptualize G for PDE problems analogously.

Universal Approximation Theorem for Operator

Theorem (Chen & Chen, 1995)

σ is a continuous non-plynomial function, X is a Banach space, $K_1 \subset X, K_2 \subset \mathbb{R}^d$ are compact sets in X and $\mathbb{R}^d$, respectively. V is a compact set in $C(K_1)$, G is a nonlinear continuous operator, which maps V into $C(K_2)$. Then for any $\epsilon > 0$, there are positive integers p, n, m, constants $c_i^k, \xi_{ij}^k, \theta_i^k, \zeta_k \in \mathbb{R}$, $w_k \in \mathbb{R}^d$, $x_j \in K_1$, $k = 1, \cdots, p$, $i = 1, \cdots, n$, $j = 1, \cdots m$, such that

$$\left| G(u)(y) - \sum_{k=1}^{p} \sum_{i=1}^{n} c_i^k \sigma \left(\sum_{j=1}^{m} \xi_{ij}^k u(x_j) + \theta_i^k \right) \sigma(w_k \cdot y + \zeta_k) \right| < \epsilon \qquad (5)$$

holds for all $u \in V$ and $y \in K_2$.

DeepONet and its Consequences

DeepONet Architecture

Karniadakis group[2] suggested *DeepONet* consisting of Branch Nets. and Trunk Nets., which

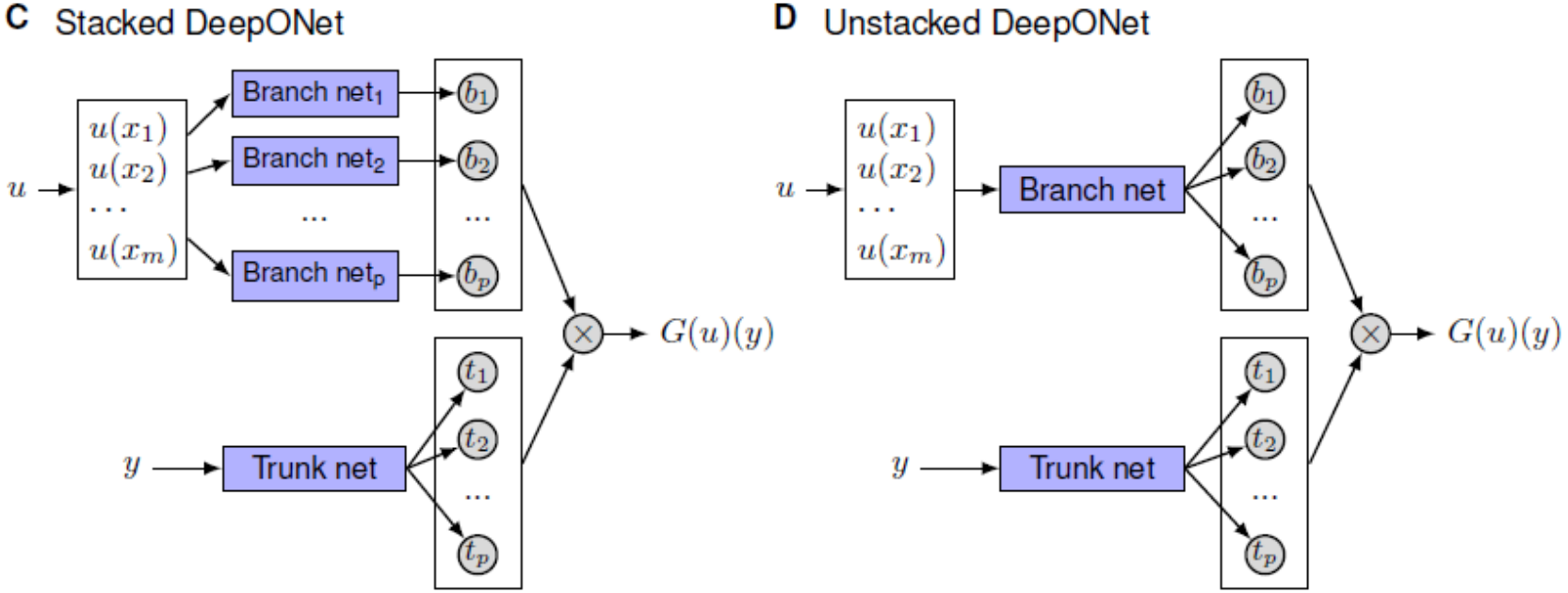
- embodies knowledge from Universal Theorem of Operator into Nets.
- improves generalization error ($e_{total} = e_{approximate} + e_{optimize} + e_{generalize}$).

$$\left| G(u)(y) - \underbrace{\sum_{k=1}^{p} \sum_{i=1}^{n} c_i^k \sigma \left(\sum_{j=1}^{m} \xi_{ij}^k u(x_j) + \theta_i^k \right)}_{branch} \underbrace{\sigma(w_k \cdot y + \zeta_k)}_{trunk} \right| < \epsilon$$

Figure: Branch and Trunk Networks in DeepONet

- $G(u)(y) \approx \sum_{k=1}^{p} b_k t_k$ can be viewed that a Trunk Network with each weight in the last layer is parametrized by another Branch Network (instead of the classical single parameter).

To further improve the generalization error, discriminated *Stacked and Unstacked* architectures on the Branch Networks

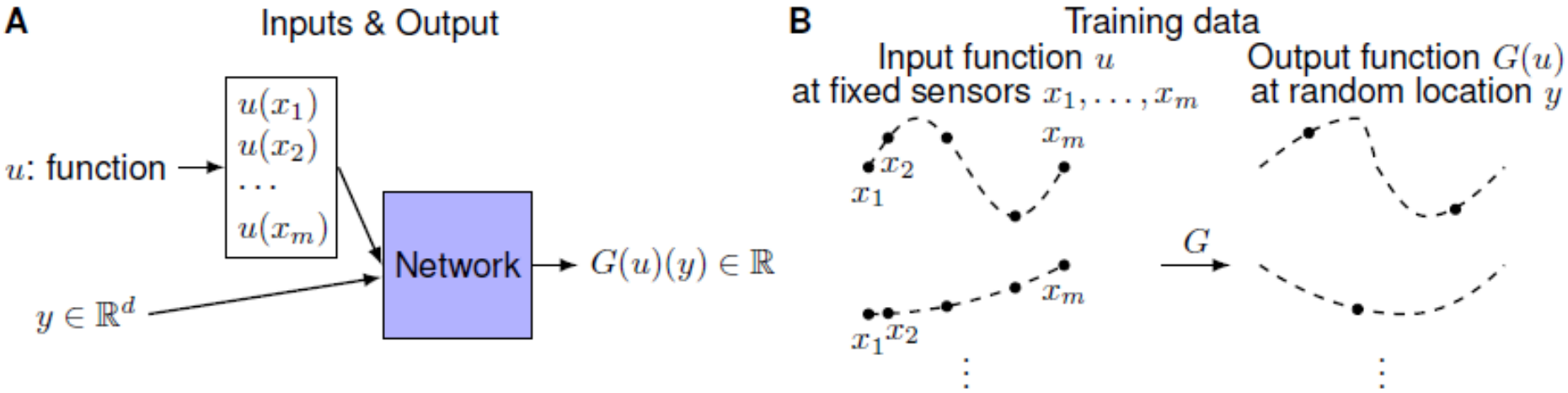


and added a *bias* term which was not included in the Universal Theorem of Operator : $G(u)(y) \approx \sum_{k=1}^{p} b_k t_k + b_0$

Input Data Generations

$G : u$ (function) $\mapsto G(u) \in C(\mathbb{R}^d)$

DeepONet requires u configured by m sensors (i.e. $[u(x_1), \cdots, u(x_m)]$) and $y \in \mathbb{R}^d$ as inputs.



Here, how to configure u with finitely many sensors is an issue!

Input Function Space V

Gaussian Random Field (GRF)

$$u \sim \mathcal{G}(0, \mathcal{K}_l(x, y)) \tag{6}$$

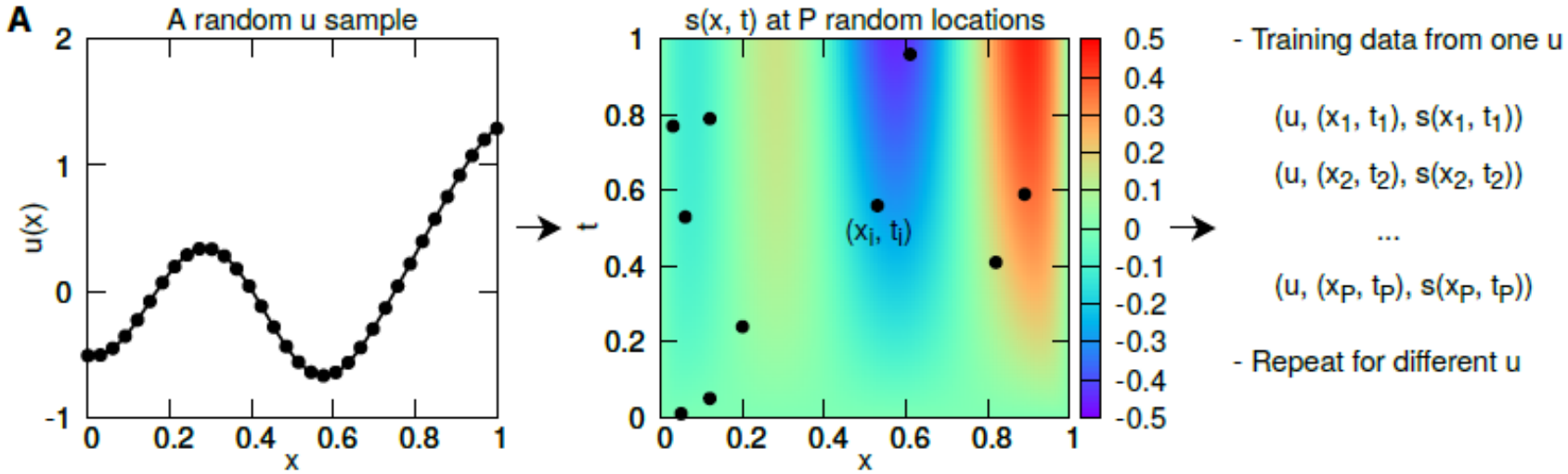
where the covariance kernel $\mathcal{K}_l(x, y) = exp(-\|x - y\|^2/2l^2)$ is a radial basis function (RBF) kernel with a length-scale parameter $l > 0$.
* l determines the smoothness of the sampled function in the sense that

$$\sup_{u \in V} \max_{x \in [a,b]} |u(x) - u_m(x)| \sim \frac{1}{m^2 l^2}. \tag{7}$$

* An error analysis on the number of sensors m is done within this function space.

Training Data Generation Process

1. Sample u from the function space.
2. Solve the ODE system by Runge-Kutta or PDEs by a second-order FDM to obtain a reference solution.
3. Pick P points from the reference solution.
   e.g.) Data Generations for PDEs



4. A data point is a triplet $(u, y, G(u)(y))$ and now train a DeepONet.

Numerical Experiments on the DeepONet Paper

* Example codes in Python-DeepXDE are available at Github: https://github.com/lululxvi/deeponet

Main numerical results

- significant generalization ability shown for a 1D linear ODE case
- effects of the number of sensors m for a dynamical system
- effects of $\#(\text{training data points}) = \#u \times P$ with different pairs for a PDE

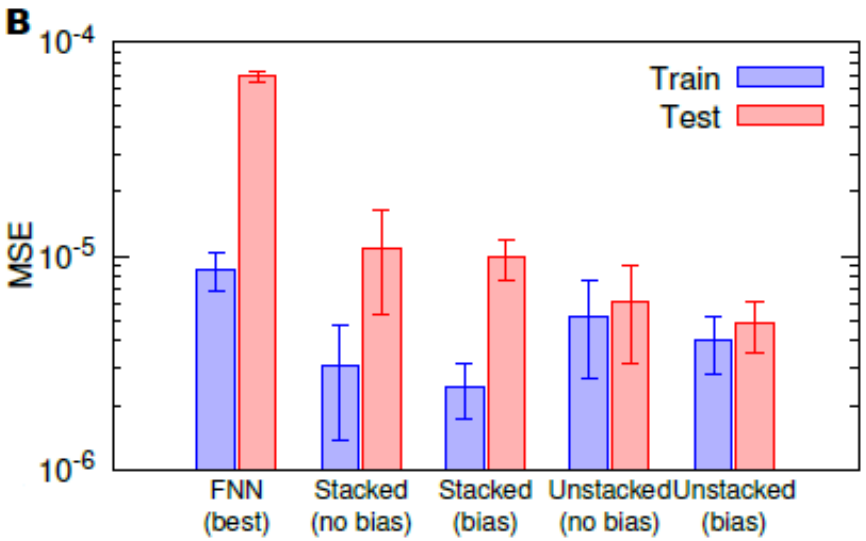
1D Linear ODE

$$\frac{ds(x)}{dx} = u(x), x \in [0, 1] \tag{8}$$

with $s(0) = 0$.

$$G : u(x) \mapsto s(x) = \int_0^x u(t)dt \tag{9}$$

with 100 sensors, $1 \times 10,000$ $G(u)(y)$ points, after 50,000 iterations.



- FFN (Left): standard neural network w/o PINN
- Unstacked w/ bias architecture got the best generalization property.

Gravity Pendulum with an External Force

$$\frac{ds_1}{dt} = s_2 \quad \frac{ds_2}{dt} = -k \sin s_1 + u(t) \tag{10}$$

with $\mathbf{s}(0) = \mathbf{0}$. Trained with different number of sensors m, after 100,000 iterations. $m \propto \sqrt{T}$, $m \propto l^{-1}$.

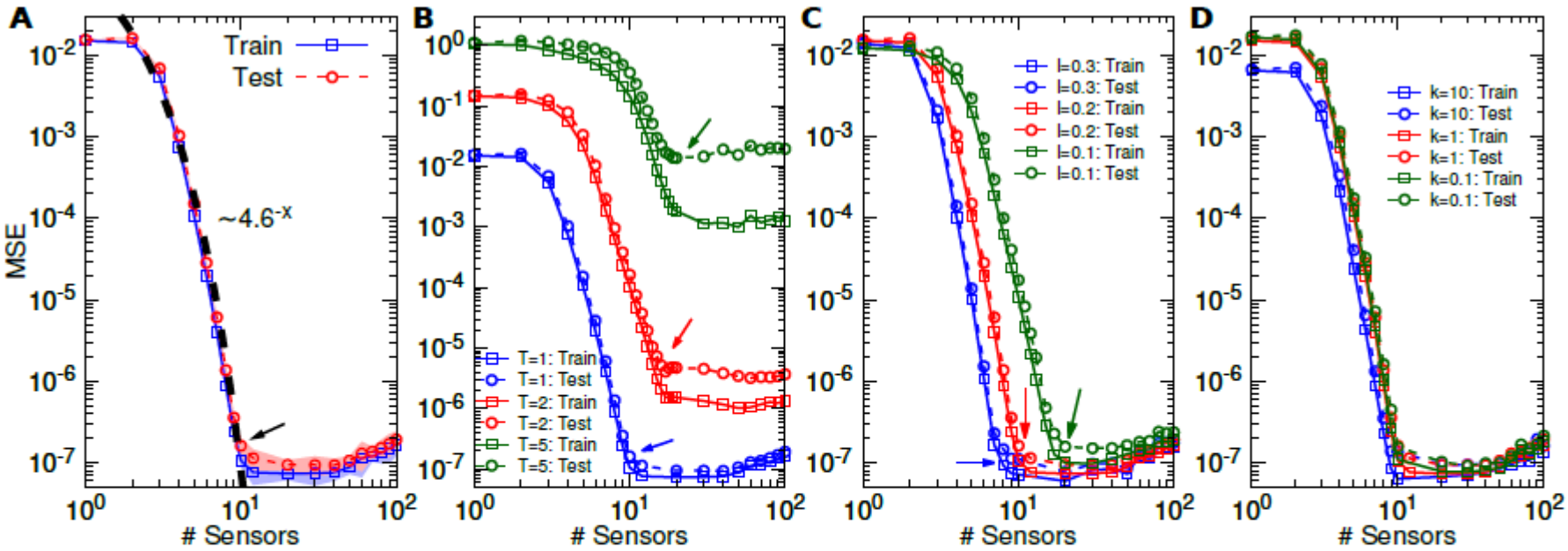
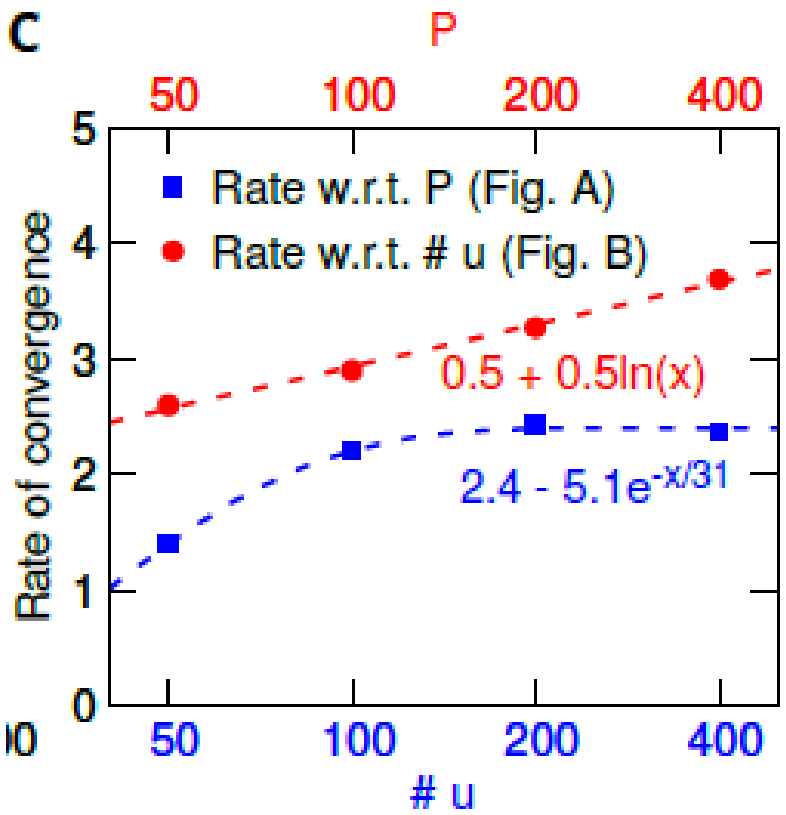


Figure 6: **Gravity pendulum: required number of sensors for different T, k and l.** (**A**) Training MSE (square symbols) and test MSE (circle symbols) decrease as the number of sensors increases in the case $k = 1$, $T = 1$ and $l = 0.2$. Training and test MSE versus the number of sensors in different conditions of (**B**) T, (**C**) l, and (**D**) k. For clarity, the SD is not shown. The arrow indicates where the rate of the error decay diminishes.

Diffusion-reaction system with a source term $u(x)$

$$\frac{\partial s}{\partial t} = D\frac{\partial^2 s}{\partial x^2} + ks^2 + u(x),\ x \in [0,1], t \in [0,1] \tag{11}$$

with zero initial boundary conditions. $G : u(x) \mapsto s(t,x)$ trained with 100 sensors, after 500,000 iterations.



- Red: The polynomial rates of convergence versus the values of P
- Blue: The polynomial rates of convergence versus the number of u samples
  More u samples induces faster convergence until it saturates.
- The rate of convergence w.r.t. P depends on the number of u samples, and vice versa.

Feed-forward Network architecture is used for the construction of Branch Networks and Trunk networks in DeepONet, and each FFN's depth and width for the examples are given as the table:

Eq. type	Trunk	Dep_{trunk}	Wid_{trunk}	Dep_{branch}	Wid_{branch}
1D L-ODE	Stacked/ Unstacked	3	40	2	40
1D NL-ODE					
Gravity pendulum	Unstacked				
PDE	Unstacked		100		100

Table: DeepONet size

Follow-up Studies

- **Fourier Neural Operators** (Zongyi Li et al.[3]): to improve efficiency of computing integral operators
- **Meta-learning**: to systemically choose NN training parameters and architectures
- **Error Estimate analysis**: to estimate the convergence of error with respect to model parameters

References I


Lu Lu, Xuhui Meng, Zhiping Mao, and George E. Karniadakis.
Deepxde: A deep learning library for solving differential equations.
CoRR, abs/1907.04502, 2019.


Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis.
Learning nonlinear operators via deeponet based on the universal approximation theorem of operators.
Nature Machine Intelligence, 3:218–229, 2021.


Zongyi Li, Nikola B. Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew M. Stuart, and Anima Anandkumar.
Fourier neural operator for parametric partial differential equations.
CoRR, abs/2010.08895, 2020.