DS 2: Learning Operators

CS 598: Deep Generative and Dynamical Models

Instructor: Arindam Banerjee

November 30, 2021

Introduction

- Classical ML maps vectors to vectors
  - Extensions map structures to other structures
  - Examples: ranking, sequence to sequence, graphs, etc.
- Operators map functions to functions
- Examples: integration, ODEs, PDEs, stochastic PDEs
- In practice, often need to solve inverse problems
  - Parameterized PDEs to model real world phenomena
  - Parameters are unknown, need to be estimated
  - Inverse problem, need to solve the PDE several times
- Existing methods: fixed resolution, specific parameterization

Learning Operators

- $D \subset \mathbb{R}^d$ is bounded, $\mathcal{A} = \mathcal{A}(D; \mathbb{R}^{d_a}), \mathcal{U} = \mathcal{U}(D; \mathbb{R}^{d_u})$
  - $\mathcal{A}, \mathcal{U}$ are Banach spaces of functions in $R^{d_a}, \mathbb{R}^{d_u}$ respectively
- $G^\dagger : \mathcal{A} \mapsto \mathcal{U}$, e.g., solution operator of parametric PDEs
- Observations $\{a_i, u_j\}_{j=1}^N$ with $u_j = G^\dagger(a_j)$
- Goal: Learn a paramteric map to approximate $G^\dagger$

$$G : \mathcal{A} \times \Theta \mapsto \mathcal{U} \qquad \equiv \qquad G_\theta : \mathcal{A} \mapsto \mathcal{U} \ , \theta \in \Theta$$

- Suitable cost function to ensure $G(\cdot, \theta^\dagger) = G_{\theta^\dagger} \approx G^\dagger$

$$\min_{\theta \sim \Theta} \mathbb{E}_a[C(G(a, \theta), G^\dagger(a))]$$

- (a_j, u_j) are functions, need evaluations at $D_j = \{x_1, \ldots, x_n\}$
  - Observations $a_j|_{D_j} \in \mathbb{R}^{n \times d_a}$, $u_j|_{D_j} \in \mathbb{R}^{n \times d_v}$

Neural Operator

- Neural operators as layer-wise transforms
  - Map input $a \in \mathcal{A}$ to high-d representation $v_0(x) = P(a(x))$
  - Sequence of maps $v_0 \mapsto v_1 \mapsto \cdots \mapsto v_T$
  - Map v_T to output $u(x) = Q(V_T(x))$, $Q : \mathbb{R}^{d_v} \mapsto \mathbb{R}^{d_u}$

Definition 1 (Iterative updates) *Define the update to the representation* $v_t \mapsto v_{t+1}$ *by*

$$v_{t+1}(x) := \sigma\Big(W v_t(x) + \big(\mathcal{K}(a;\phi)v_t\big)(x)\Big), \qquad \forall x \in D \tag{2}$$

where $\mathcal{K} : \mathcal{A} \times \Theta_{\mathcal{K}} \to \mathcal{L}(\mathcal{U}(D;\mathbb{R}^{d_v}),\mathcal{U}(D;\mathbb{R}^{d_v}))$ *maps to bounded linear operators on* $\mathcal{U}(D;\mathbb{R}^{d_v})$ *and is parameterized by* $\phi \in \Theta_{\mathcal{K}}$, $W : \mathbb{R}^{d_v} \to \mathbb{R}^{d_v}$ *is a linear transformation, and* $\sigma : \mathbb{R} \to \mathbb{R}$ *is a non-linear activation function whose action is defined component-wise.*

Neural Operator (Contd.)

- Kernel integral operator $\mathcal{K}$

$$(\mathcal{K}(a;\phi)v_t)(x) = \int_D \kappa_\phi(x, y, a(x), a(y))v_t(y)dy \ , \quad \forall x \in D$$

- $\kappa_\phi : \mathbb{R}^{2(d+d_a)} \mapsto \mathbb{R}^{d_v \times d_v}$ is a deep network with parameter ϕ
- Specific choice of radial kernel, avoiding dependence on a

$$\kappa_\phi(x, y, a(x), a(y)) = \kappa_\phi(x - y)$$

- Then, $(\mathcal{K}(a;\phi)v_t)(x)$ is a convolution operator
  - Efficiently computed as product in Fourier space

Fourier Neural Operator (FNO)

- Compute convolution as product in Fourier space
- Fourier transform and inverse for $f : D \mapsto \mathbb{R}^{d_v}$

$$(\mathcal{F}f)_j(k) = \int_D f_j(x) e^{-2i\pi\langle x,k\rangle} dx$$

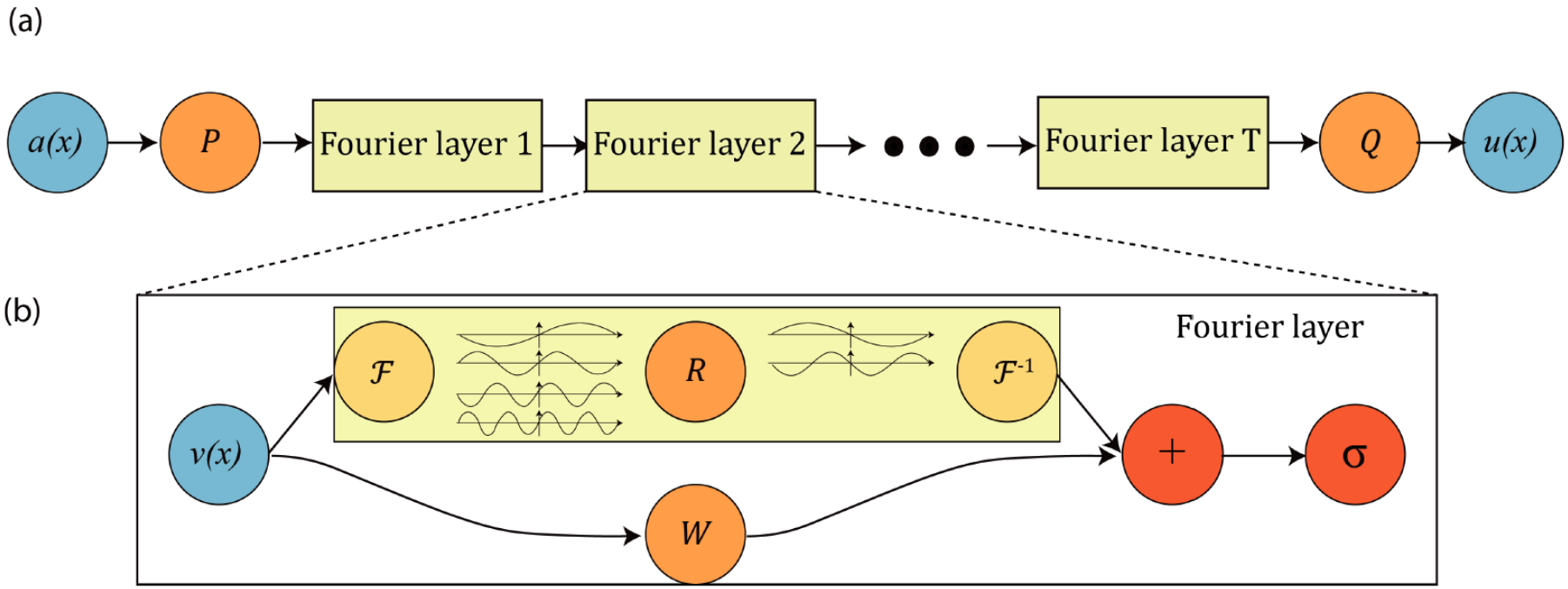
$$(\mathcal{F}^{-1}f)_j(x) = \int_D f_j(k) e^{2i\pi\langle x,k\rangle} dx$$

- Fourier integral operator $\mathcal{K}$:

$$(\mathcal{K}(\phi)v_t)(x) = \mathcal{F}^{-1}\left(R_\phi \cdot (\mathcal{F}v_t)\right)(x) \ , \quad \forall x \in D$$

  - For $k \in D$, $(\mathcal{F}v_t)(k) \in \mathbb{C}^{d_v}$ and $\mathbb{R}_\phi(k) \in \mathbb{C}^{d_v \times d_v}$
- R_ϕ is the Fourier transform of a periodic $\kappa_\phi : D \mapsto \mathbb{R}^{d_v \times d_v}$
  - Allows Fourier series expansion
  - Model uses finite-d representation with $k_{\max}$ modes

FNO Architecture



(a) The full architecture of neural operator: start from input a. 1. Lift to a higher dimension channel space by a neural network P. 2. Apply four layers of integral operators and activation functions. 3. Project back to the target dimension by a neural network Q. Output u. **(b) Fourier layers**: Start from input v. On top: apply the Fourier transform $\mathcal{F}$; a linear transform R on the lower Fourier modes and filters out the higher modes; then apply the inverse Fourier transform $\mathcal{F}^{-1}$. On the bottom: apply a local linear transform W.

FNO: Discrete setting, FFT

- Domain $D \subset \mathbb{R}^d$ is discretized with n points
  - Specific case: uniform grid per dimension, $n = \prod_{i=1}^{d} s_i$
- With $v_t \in \mathbb{R}^{n \times d_v}$, $\mathcal{F}(v_t) \in \mathbb{C}^{n \times d_v}$
  - With mode truncation $R \in \mathbb{C}^{k_{\max} \times d_v \times d_v}$, use $\mathcal{F}(v_t) \in \mathbb{R}^{k_{\max} \times d_v}$
  - Computation is matrix-vector product

$$(R \cdot \mathcal{F}(v_t))_{k,l} = \sum_{j=1}^{d_v} R_{k,l,j} (\mathcal{F}(v_t))_{k,j} , \quad k = 1, \ldots, k_{\max}, \;\; j = 1, \ldots, d_v$$

- For uniform grid, $\mathcal{F}$ can be replaced by Fast Fourier Transform

Example: Burger's Equation

- Modeling 1-d flow of viscous fluid

$$\begin{aligned} \partial_t u(x,t) + \partial_x(u^2(x,t)/2) &= \nu \partial_{xx} u(x,t), && x \in (0,1), t \in (0,1] \\ u(x,0) &= u_0(x), && x \in (0,1) \end{aligned}$$

- Initial condition $u_0 \in L^2_{\text{per}}((0,1);\mathbb{R})$
- $\nu \in \mathbb{R}_+$ is the viscosity coefficient
- Operator learning: initial condition to solution at $t = 1$

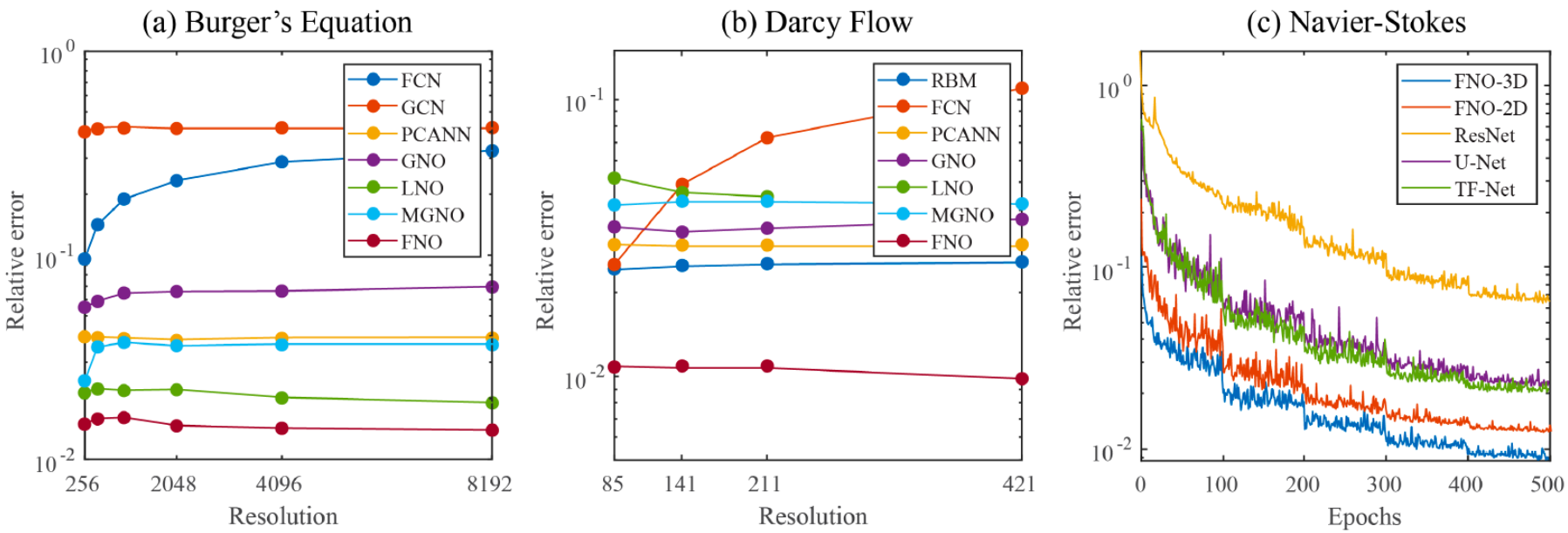
Example: Navier-Stokes Equation

- Modeling 2-d Navier-Stokes equation

$$\begin{aligned} \partial_t w(x,t) + u(x,t) \cdot \nabla w(x,t) &= \nu \Delta w(x,t) + f(x), && x \in (0,1)^2, t \in (0,T] \\ \nabla \cdot u(x,t) &= 0, && x \in (0,1)^2, t \in [0,T] \\ w(x,0) &= w_0(x), && x \in (0,1)^2 \end{aligned}$$

- Key components
  - u: the velocity field
  - w_0: initial vorticity
  - f: the forcing function
  - $\nu \in \mathbb{R}_+$: viscosity coefficient
- Operator learning: Vorticity till t=10 to vorticity at $T > 10$

Results: Relative Error



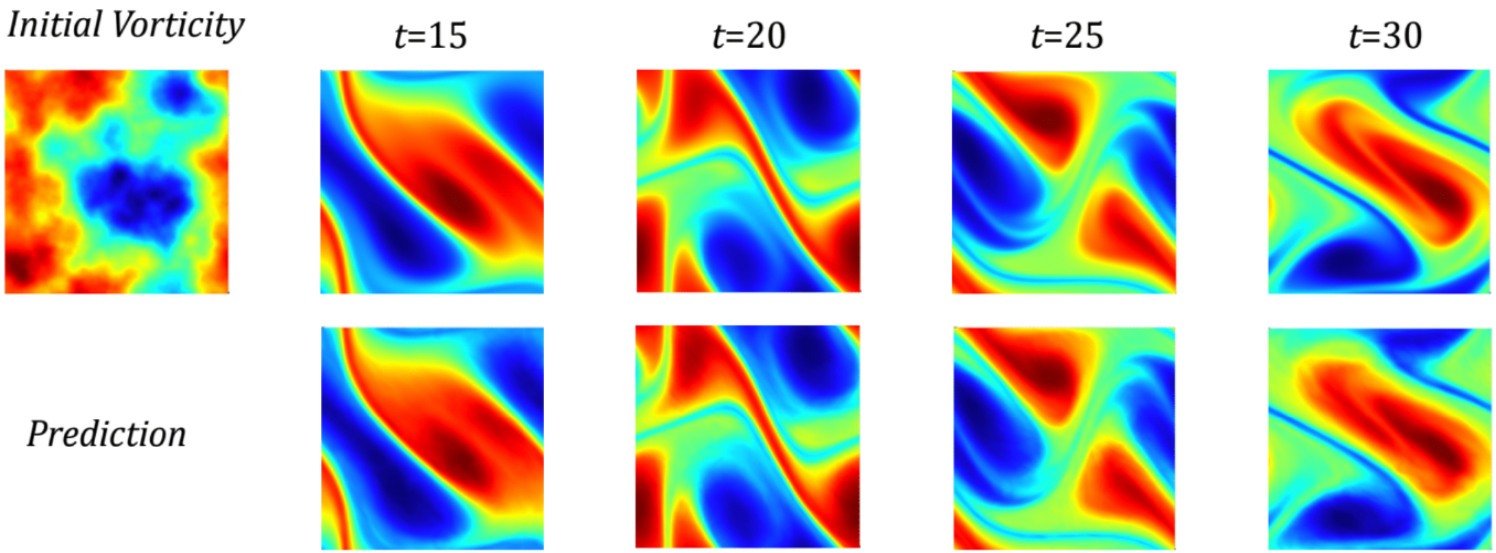
Left: benchmarks on Burgers equation; **Mid:** benchmarks on Darcy Flow for different resolutions; **Right:** the learning curves on Navier-Stokes $\nu = 1\text{e}{-3}$ with different benchmarks. Train and test on the same resolution. For acronyms, see Section 5; details in Tables 1, 3, 4.

Results: Navier Stokes

Table 1: Benchmarks on Navier Stokes (fixing resolution 64×64 for both training and testing)

Config	**Parameters**	**Time per epoch**	$\nu = 1\text{e}{-3}$ $T = 50$ $N = 1000$	$\nu = 1\text{e}{-4}$ $T = 30$ $N = 1000$	$\nu = 1\text{e}{-4}$ $T = 30$ $N = 10000$	$\nu = 1\text{e}{-5}$ $T = 20$ $N = 1000$
FNO-3D	$6,558,537$	$38.99s$	**0.0086**	0.1918	**0.0820**	0.1893
FNO-2D	$414,517$	$127.80s$	0.0128	**0.1559**	0.0834	**0.1556**
U-Net	$24,950,491$	$48.67s$	0.0245	0.2051	0.1190	0.1982
TF-Net	$7,451,724$	$47.21s$	0.0225	0.2253	0.1168	0.2268
ResNet	$266,641$	$78.47s$	0.0701	0.2871	0.2311	0.2753

Results: Zero-Shot Super Resolution



Zero-shot super-resolution: Navier-Stokes Equation with viscosity $\nu = 1e{-}4$; Ground truth on top and prediction on bottom; trained on $64 \times 64 \times 20$ dataset; evaluated on $256 \times 256 \times 80$ (see Section 5.4).

Figure 1: **top:** The architecture of the Fourier layer; **bottom:** Example flow from Navier-Stokes.

Learning Operators with DeepONets

- Given a function u, operator G maps to function $G(u)$
  - Realization of u represented as $[u(x_1) \cdots u(x_m)]$
  - $G(u)$ evaluated at some y, with value $G(u)(y)$
- Goal: Deep model $G_\theta(u, y) \approx G(u)(y)$ for all y
- Training phase:
  - Different u at sensor locations $\{x_1, \ldots, x_m\}$
  - Corresponding $G(u)(y)$ at locations y
- Test phase: Given function u and y, predict $G_\theta(u, y)$

Universal Approximation

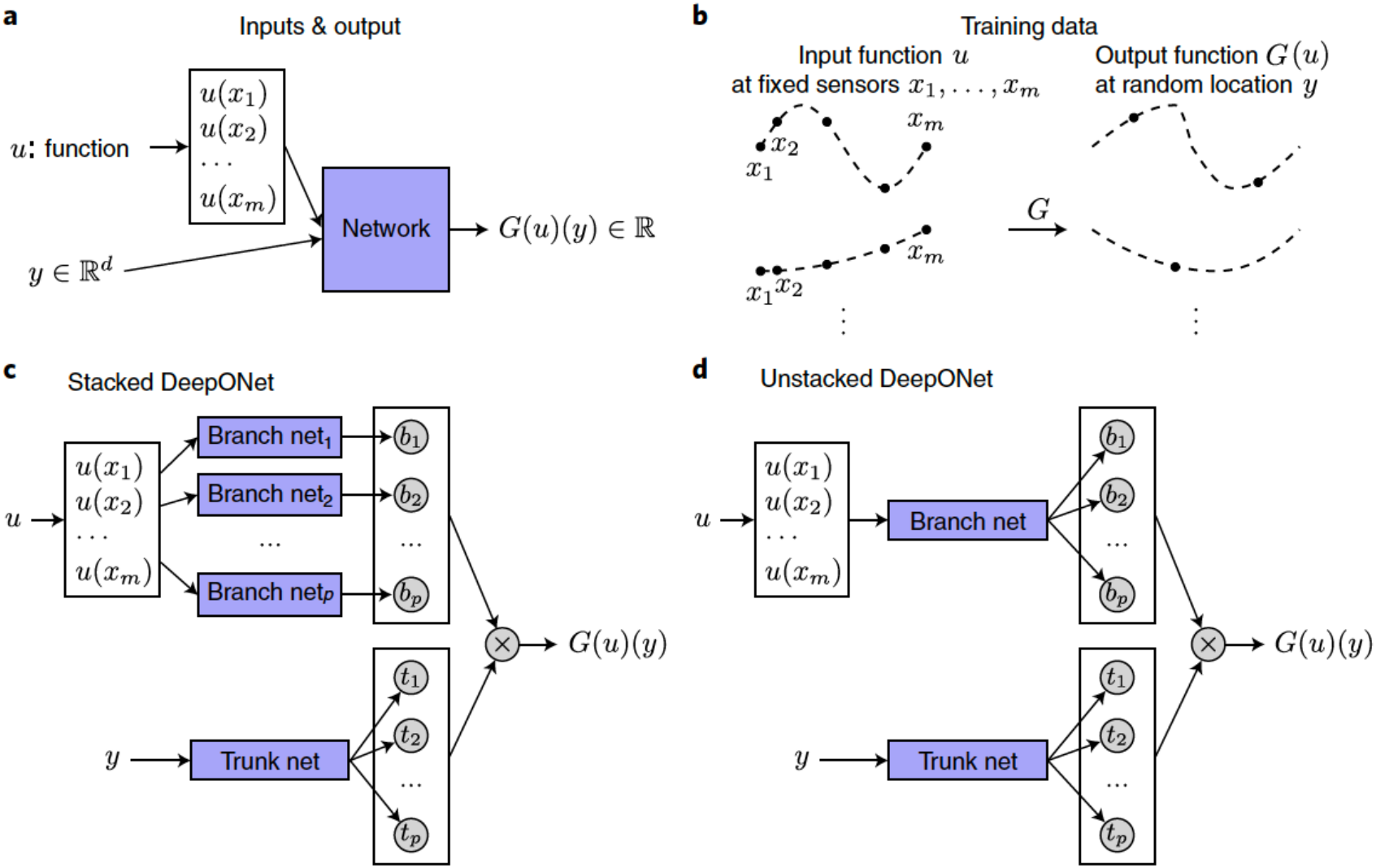
Theorem 1 (Universal Approximation Theorem for Operator). *Suppose that σ is a continuous non-polynomial function, X is a Banach space, $K_1 \subset X$, $K_2 \subset \mathbb{R}^d$ are two compact sets in X and $\mathbb{R}^d$, respectively, V is a compact set in $C(K_1)$, G is a nonlinear continuous operator,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

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

holds for all $u \in V$ and $y \in K_2$. Here, $C(K)$ is the Banach space of all continuous functions defined on K with norm $\| f \|_{C(K)} = \max_{x \in K} |f(x)|$.

- Three sources of error: approximation, optimization, generalization
  - Illustrates approximation error will be small

DeepONet Overview



DeepONet Architecture

- Sensor locations $\{x_1, \ldots, x_m\}$ are the same for all functions u
  - Does not need to be on a lattice
  - Can be avoided by modeling u, e.g., interpolation, basis functions
- Output y is d-dimensional, does not need to match u
- Can feed $[u(x_1), \ldots, u(x_2), y]^T$ as input
  - Output is $G(u)(y)$, train using backprop, e.g., Adam
  - Used as baseline with different architectres, e.g., CNNs, Seq2Seq
- DeepONet architecture is based on branch-trunk from approximation result

$$G(u)(y) \approx \sum_{k=1}^{p} \underbrace{b_k(u(x_1), \ldots, u(x_m))}_{\text{branch}} \underbrace{t_k(y)}_{\text{trunk}}$$

  - In practice, also use a bias term $G(u)(y) \approx \sum_{k=1}^{p} b_k t_k + b_0$

Generalized Universal Approximation

Theorem 2 (Generalized Universal Approximation Theorem for Operator). *Suppose that X is a Banach space, $K_1 \subset X$, $K_2 \subset \mathbb{R}^d$ are two compact sets in X and $\mathbb{R}^d$, respectively, V is a compact set in $C(K_1)$. Assume that $G: V \to C(K_2)$ is a nonlinear continuous operator. Then, for any $\epsilon > 0$, there exist positive integers m, p, continuous vector functions $\mathbf{g} : \mathbb{R}^m \to \mathbb{R}^p$, $\mathbf{f} : \mathbb{R}^d \to \mathbb{R}^p$, and $x_1, x_2, \ldots, x_m \in K_1$, such that*

$$\left| G(u)(y) - \langle \underbrace{\mathbf{g}(u(x_1), u(x_2), \cdots, u(x_m))}_{\text{branch}}, \underbrace{\mathbf{f}(y)}_{\text{trunk}} \rangle \right| < \epsilon$$

holds for all $u \in V$ and $y \in K_2$, where $\langle \cdot, \cdot \rangle$ denotes the dot product in $\mathbb{R}^p$. Furthermore, the functions $\mathbf{g}$ and $\mathbf{f}$ can be chosen as diverse classes of neural networks, which satisfy the classical universal approximation theorem of functions, for example, (stacked/unstacked) fully connected neural networks, residual neural networks and convolutional neural networks.

- Allows for multi-layer networks, representation in the same space

Applications: Problem Types

- Explicit operators
  - Integration
  - Legendre transforms
  - Fraction derivatives, Laplacians
- Implicit operators
  - Determistic ODEs
  - PDEs
  - Stochstic PDEs

Example: Anti-derivative (Integral) Operator

- Explicit operator example

$$\frac{ds(x)}{dx} = g(s(x), u(x), x) \ , \quad x \in (0, 1]$$

- Example: $g(s(x), u(x), x) = u(x)$, anti-derivative (integral) operator

$$G : u(x) \mapsto s(x) \ , \quad s(x) \equiv s_0 + \int_0^x u(\tau) d\tau, \ x \in [0, 1]$$

Results: Anti-derivative Operator

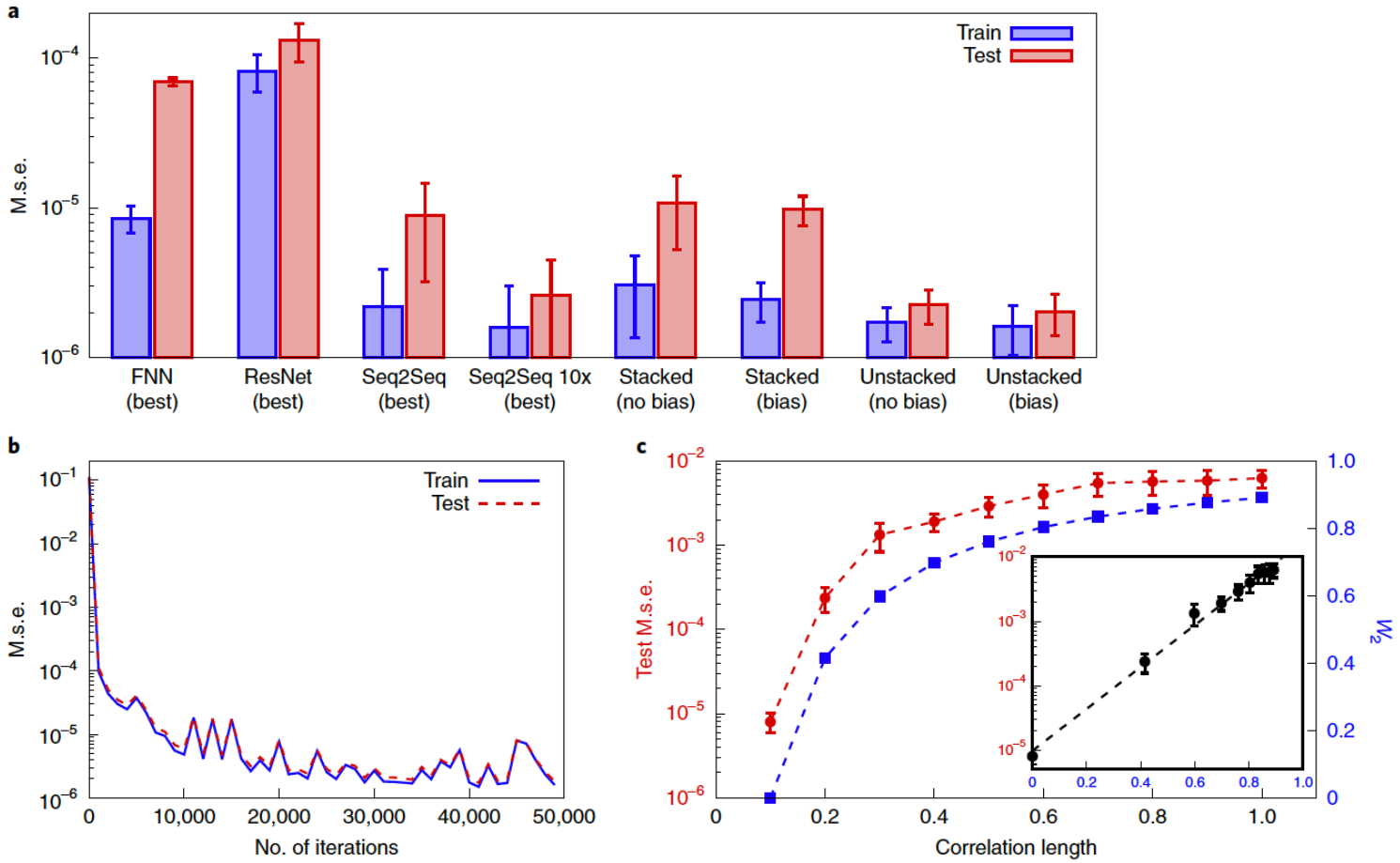


Fig. 2 | Learning explicit operators using different *V* spaces and different network architectures. a, Errors of different network architectures trained to learn the antiderivative operator (linear case). The training/test errors of stacked/unstacked DeepONets with/without bias compared with the best test error and the corresponding training error of FNNs, ResNets and Seq2Seq models. The 'Seq2Seq 10x' is a Seq2Seq model with 10 times more training data points. The error bars show the one standard deviation from 10 runs with different training/test data and network initialization. **b**, The training trajectory of an unstacked DeepONet with bias (m.s.e., mean squared error). **c**, The error (mean and standard deviation) tested on the space of Gaussian random fields (GRFs) with the correlation length $l = 0.1$ for DeepONets trained with GRF spaces of different correlation length l (red curve). The 2-Wasserstein metric between the GRF of $l = 0.1$ and a GRF of different correlation length l is shown as a blue curve. The test error grows exponentially with respect to the W_2

Results: Gravity Pendulum

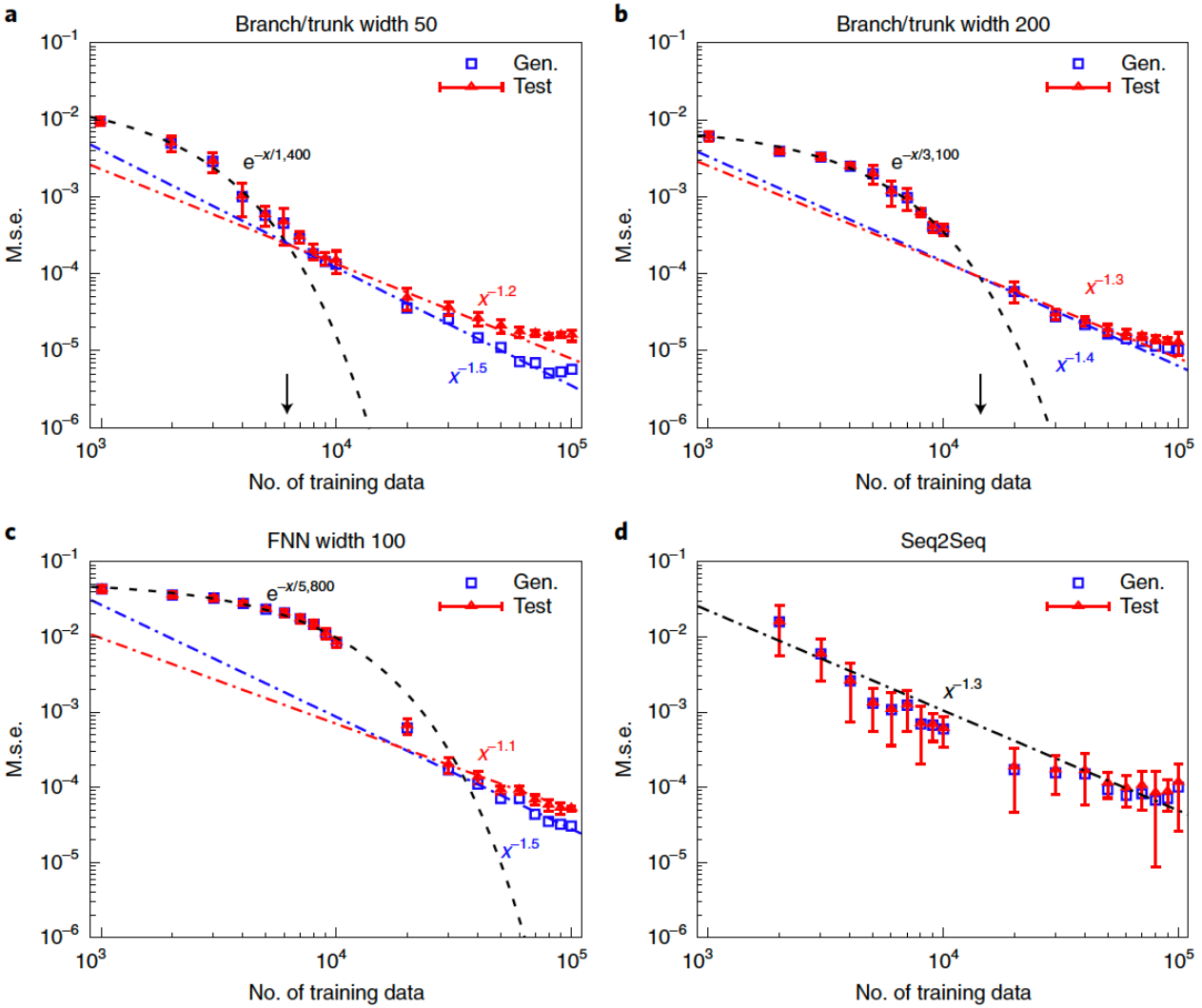


Fig. 3 | Fast learning of implicit operators in a nonlinear pendulum ($k=1$ and $T=3$). a,**b**, The test and generalization errors of DeepONets have exponential convergence for small training datasets, and then converge with polynomial rates. The transition point from exponential to polynomial (indicated by the arrow) convergence depends on the width (branch/trunk width of 50 in **a** and 200 in **b**), and a bigger network has a later transition point. **c**, FNNs also have an initial exponential error decay, but with much larger error and much slower convergence speed. **d**, Seq2Seq models have a roughly polynomial convergence rate, and the test errors have a large variation for different runs. x is the number of training data.

Results: Diffusion-Reaction PDE

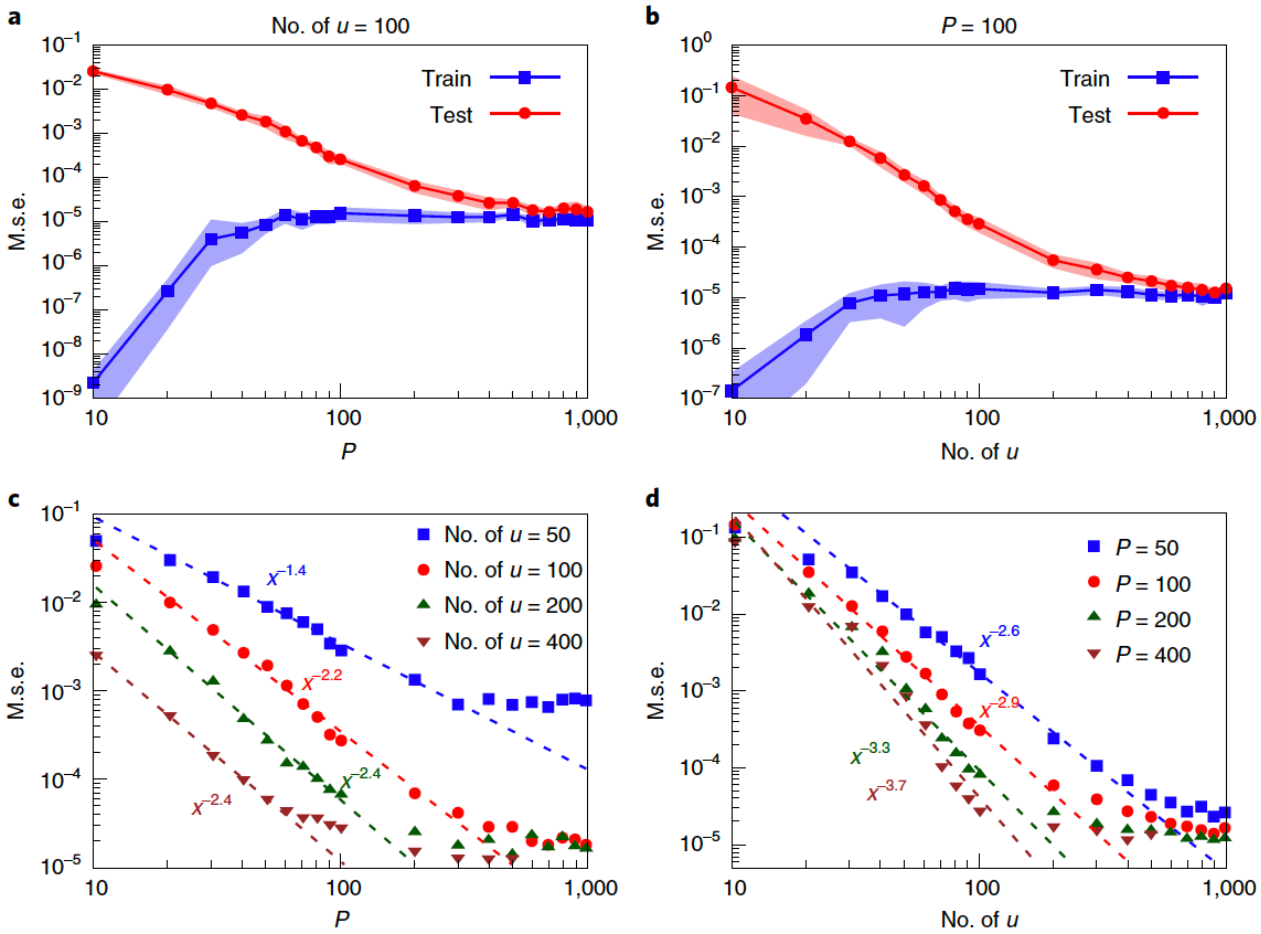


Fig. 4 | Fast learning of implicit operators in a diffusion-reaction system. a,**b**, Comparison of training (blue) and testing (red) errors for different values of the number of random points P when 100 random u samples are used (**a**) and for different numbers of u samples when P=100 (**b**). The shaded regions denote one standard deviation. **c**,**d**, The algebraic decay of test errors in terms of the number of sampling points P and the number of input functions $u(x, t)$: convergence of test error with respect to P for different numbers of u samples (**c**) and with respect to the number of u samples for different values of P (**d**).

References

- Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, A.Anandkumar, Fourier neural operator for parametric partial differential equations, ICLR, 2021.
- L. Lu, R. Jin, G.Pang, Z. Zhang, G. E. Karniadakis, Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators, NMI, 2021.