Universität Bremen

Center for Industrial Mathematics (ZeTeM)

Faculty 03

Mathematics / Computer science

Operator Learning and DeepONets

Janek Gödeke, Nick Heilenkötter, Tom Freudenberg

Heidelberg, 08.11.2023

Yesterday: PINNs for Single PDE

Poisson equation:

$$\begin{aligned} \Delta u(x) &= f(x) && \text{on } \Omega \subset \mathbb{R}^2, \\ u(x) &= c && \text{for } x \in \partial\Omega. \end{aligned}$$

Task: Learn u for **fixed** $c \in \mathbb{R}$ and $f : \Omega \to \mathbb{R}$

Yesterday: PINNs for Single PDE

Poisson equation:

$$\begin{aligned} \Delta u(x) &= f(x) \quad \text{on } \Omega \subset \mathbb{R}^2, \\ u(x) &= c \quad \text{for } x \in \partial\Omega. \end{aligned}$$

Task: Learn u for **fixed** $c \in \mathbb{R}$ and $f : \Omega \to \mathbb{R}$

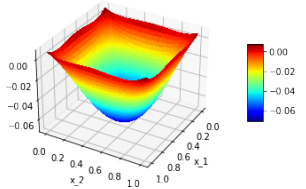


Figure: Learned solution for $f = 1$ and $c = 0$

Yesterday: PINNs for Single PDE

Poisson equation:

$$\Delta u(x) = f(x) \quad \text{on } \Omega \subset \mathbb{R}^2,$$
$$u(x) = c \quad \text{for } x \in \partial\Omega.$$

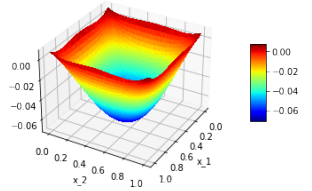


Figure: Learned solution for $f = 1$ and $c = 0$

Task: Learn u for **fixed** $c \in \mathbb{R}$ and $f : \Omega \to \mathbb{R}$

Neural network:

$$u_\theta : \Omega \longrightarrow \mathbb{R}$$
$$u_\theta(x) \approx u(x)$$

Drawback: New c or f —> retrain u_θ ...

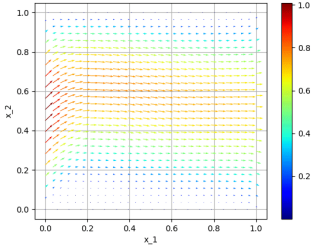
This Morning: PINNs for Multiple PDEs

Stokes problem:

$$\nabla p_\alpha = \frac{\mu}{2}\text{div}(\nabla u_\alpha + \nabla u_\alpha^T) \quad \text{in } \Omega$$

$$u_\alpha = u_{in}(\alpha, x) \quad \text{at } \partial\Omega_{left}$$

Task: Learn velocity u_α and pressure p_α for **all** in-flow angles $\alpha \in [\frac{\pi}{4}, \frac{\pi}{2}]$.

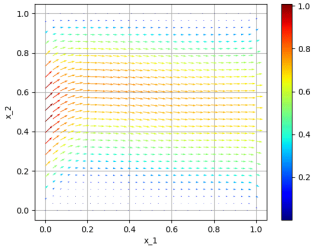


This Morning: PINNs for Multiple PDEs

Stokes problem:

$$\nabla p_\alpha = \frac{\mu}{2}\text{div}(\nabla u_\alpha + \nabla u_\alpha^T) \quad \text{in } \Omega$$

$$u_\alpha = u_{in}(\alpha, x) \quad \text{at } \partial\Omega_{left}$$



Task: Learn velocity u_α and pressure p_α for **all** in-flow angles $\alpha \in [\frac{\pi}{4}, \frac{\pi}{2}]$.

- Parameter α as additional input for Neural Network:

$$\varphi_\theta : \Omega \times \left[\frac{\pi}{4}, \frac{\pi}{2}\right] \longrightarrow \mathbb{R}^3$$

$$\varphi_\theta(x, \alpha) \approx \begin{pmatrix} u_\alpha(x) \\ p_\alpha(x) \end{pmatrix}$$

Outlook: Operator Learning

Poisson equation:

$$\begin{aligned} \Delta u(x) &= f(x) && \text{on } \Omega \subset \mathbb{R}^2, \\ u(x) &= c && \text{for } x \in \partial\Omega. \end{aligned}$$

Task: Learn u_f for **many** $f : \Omega \to \mathbb{R}$

Outlook: Operator Learning

Poisson equation:

$$\begin{aligned} \Delta u(x) &= f(x) && \text{on } \Omega \subset \mathbb{R}^2, \\ u(x) &= c && \text{for } x \in \partial\Omega. \end{aligned}$$

Task: Learn u_f for **many** $f : \Omega \to \mathbb{R}$

Neural network?

$$\varphi_\theta : \Omega \times F \longrightarrow \mathbb{R}$$
$$\varphi_\theta(x, f) \approx u_f(x)$$

F set of functions, e.g. in $\mathcal{C}(\Omega, \mathbb{R})$

Problem: Input of NNs must be from $\mathbb{R}^d$, but f is a function...

Outlook: Operator Learning

Poisson equation:

$$\begin{aligned} \Delta u(x) &= f(x) && \text{on } \Omega \subset \mathbb{R}^2, \\ u(x) &= c && \text{for } x \in \partial\Omega. \end{aligned}$$

Operator Learning: Learn mappings with function input, e.g.,

$$\begin{aligned} G : \Omega \times F &\longrightarrow \mathbb{R} \\ G(x, f) &= u_f(x). \end{aligned}$$

Task: Learn u_f for **many** $f : \Omega \to \mathbb{R}$

Neural network?

$$\begin{aligned} \varphi_\theta : \Omega \times F &\longrightarrow \mathbb{R} \\ \varphi_\theta(x, f) &\approx u_f(x) \end{aligned}$$

F set of functions, e.g. in $\mathcal{C}(\Omega, \mathbb{R})$

Problem: Input of NNs must be from $\mathbb{R}^d$, but f is a function...

Operator Learning

Why Operator Learning?

Classical Parameter-Identification
Given: Solution u of PDE
Task: Find underlying parameters f

- Iterative algorithms: Solve many similar PDEs
- Classical PDE solver like FDM or FEM: Time-consuming
- Replace by trained NN
- Less time-consuming

First Natural Idea

Poisson equation:

$$\begin{aligned} \Delta u(x) &= f(x) \quad && \text{on } \Omega \subset \mathbb{R}^2, \\ u(x) &= c && \text{for } x \in \partial\Omega. \end{aligned}$$

Operator to be learned

$$\begin{aligned} G : \Omega \times F &\longrightarrow \mathbb{R} \\ (x, f) &\longmapsto u_f(x) \end{aligned}$$

First Natural Idea

Poisson equation:

$$\begin{aligned} \Delta u(x) &= f(x) && \text{on } \Omega \subset \mathbb{R}^2, \\ u(x) &= c && \text{for } x \in \partial\Omega. \end{aligned}$$

Operator to be learned

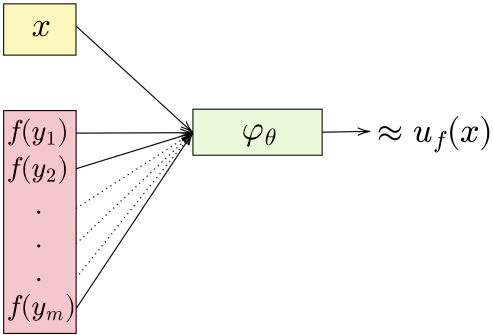
$$\begin{aligned} G : \Omega \times F &\longrightarrow \mathbb{R} \\ (x, f) &\longmapsto u_f(x) \end{aligned}$$

- Sample f at $y_1, \ldots, y_m \in \Omega$
- Use samples as input of NN

$$\varphi_\theta : \Omega \times \mathbb{R}^m \longrightarrow \mathbb{R}$$

$$\varphi_\theta \left(x, \begin{pmatrix} f(y_1) \\ \vdots \\ f(y_m) \end{pmatrix} \right) \approx u_f(x)$$

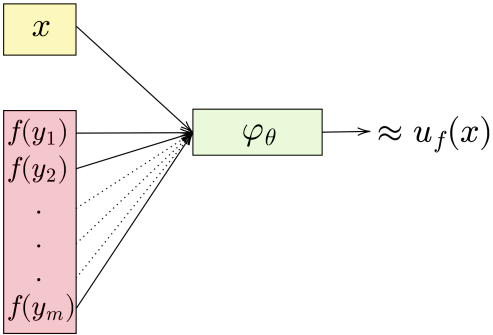
First Natural Idea



Feed x & samples of f into NN φ_θ

- Note: $y_1, ..., y_m$ are fixed
- Free evaluation of u_f at x

First Natural Idea



Feed x & samples of f into NN φ_θ

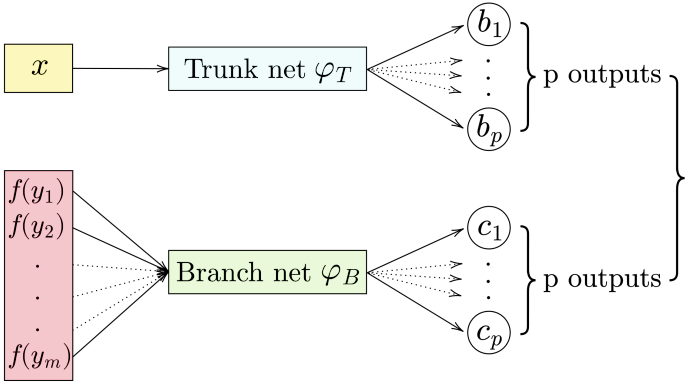
- Note: $y_1, ..., y_m$ are fixed
- Free evaluation of u_f at x

Problematic:

- Many $f(y_i)$-inputs versus $x \in \mathbb{R}^2$
  $\rightarrow$ Imbalance

DeepONets

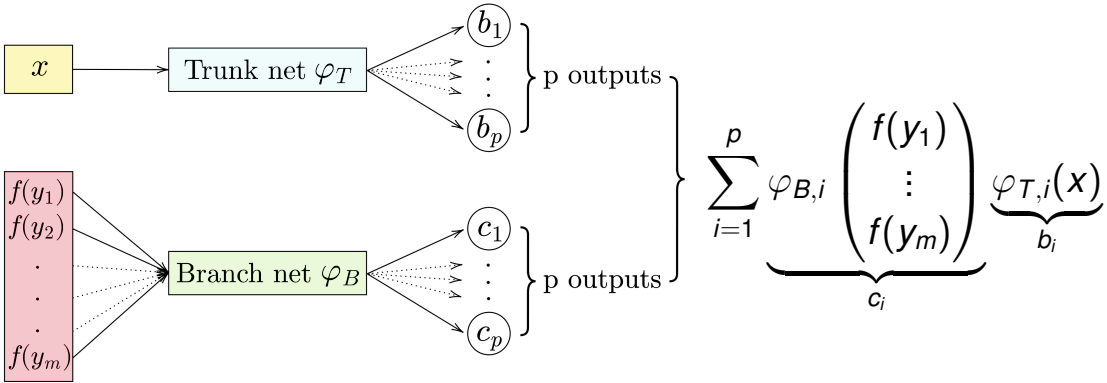
DeepONets[1] - Divide and Conquer



$$\sum_{i=1}^{p} c_i b_i \approx u_f(x)$$

[1] Lu, Jin, Karniadakis, *Learning Nonlinear Operators (...)*, 2019

DeepONets[1] - Divide and Conquer



[1] Lu, Jin, Karniadakis, *Learning Nonlinear Operators (...)*, 2019

DeepONets - Interpretation

Goal: Learn operator $(x, f) \mapsto u_f(x)$ (e.g., of Poisson problem $\Delta u(x) = f(x)$).

$$u_f(x)$$

$$\approx$$

$$\sum_{i=1}^{p} \varphi_{B,i} \left(\begin{pmatrix} f(y_1) \\ \dots \\ f(y_m) \end{pmatrix} \right) \varphi_{T,i}(x)$$

DeepONets - Interpretation

Goal: Learn operator $(x, f) \mapsto u_f(x)$ (e.g., of Poisson problem $\Delta u(x) = f(x)$).

$$u_f(x)$$

$$\approx$$

$$\sum_{i=1}^{p} \varphi_{B,i} \left(\begin{pmatrix} f(y_1) \\ \dots \\ f(y_m) \end{pmatrix} \right) \boxed{\varphi_{T,i}(x)}$$

1) $u_f \approx$ linear combination of trunk nets $\boxed{\varphi_{T,i}}$

DeepONets - Interpretation

Goal: Learn operator $(x, f) \mapsto u_f(x)$ (e.g., of Poisson problem $\Delta u(x) = f(x)$).

$$u_f(x) \approx \sum_{i=1}^{p} \boxed{\varphi_{B,i}\left(\begin{pmatrix} f(y_1) \\ \dots \\ f(y_m) \end{pmatrix}\right)} \varphi_{T,i}(x)$$

1) $u_f \approx$ linear combination of trunk nets $\varphi_{T,i}$

2) *Branch* nets $\boxed{\varphi_{B,i}}$ for coefficients w.r.t. trunk net "basis"

DeepONets - Interpretation

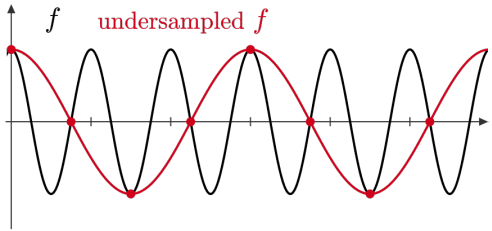
Goal: Learn operator $(x, f) \mapsto u_f(x)$ (e.g., of Poisson problem $\Delta u(x) = f(x)$).

$$u_f(x) \approx \sum_{i=1}^{p} \varphi_{B,i}\left(\begin{pmatrix} f(y_1) \\ \dots \\ f(y_m) \end{pmatrix}\right) \varphi_{T,i}(x)$$

1) $u_f \approx$ linear combination of trunk nets $\varphi_{T,i}$
2) *Branch* nets $\varphi_{B,i}$ for coefficients w.r.t. trunk net "basis"
3) Input: Always sample f at fixed points $y_1, ..., y_m \in \Omega_1$

Enough Info for Branch Net

- Sufficiently many sampling points $y_1, ..., y_m$ in domain of f



- E.g. band-limited functions f
  $\rightarrow$ Sample with Shannon-Nyquist rate[2] $(2 \cdot \text{bandwidth})^{-1}$

[2] Shannon, *Communication in the Presence of Noise*, 1949

Physics-informed Training of DeepONets φ_θ

Set-up

- Given f_k at fixed sampling points $y_1, ..., y_m$,
- Choose sampling points $x_1, ..., x_L$ for PDE conditions.

Poisson problem:

$$\begin{aligned} \Delta u(x) &= f(x) && \text{on } \Omega \subset \mathbb{R}^2, \\ u(x) &= c && \text{for } x \in \partial\Omega. \end{aligned}$$

PDE-loss for Poisson condition $\Delta u = f$:

$$\frac{1}{KL} \sum_{k=1}^{K} \sum_{l=1}^{L} \left| \Delta \varphi_\theta \left(x_l, \begin{pmatrix} f_k(y_1) \\ \vdots \\ f_k(y_m) \end{pmatrix} \right) - f_k(x_l) \right|^2$$

Training with Data

Set-up:

- Given f_k at fixed sampling points $y_1, ..., y_m$,
- Given u_k at sampling points $x_1, ..., x_L$. (x_l could depend on k)

Poisson problem:

$$\begin{aligned} \Delta u(x) &= f(x) && \text{on } \Omega \subset \mathbb{R}^2, \\ u(x) &= c && \text{for } x \in \partial\Omega. \end{aligned}$$

Data loss for K tuples (f_k, u_k):

$$\frac{1}{KL} \sum_{k=1}^{K} \sum_{l=1}^{L} \left| \varphi_\theta \left(x_l, \begin{pmatrix} f_k(y_1) \\ \vdots \\ f_k(y_m) \end{pmatrix} \right) - u_k(x_l) \right|^2$$

Training with Data

Set-up:

- Given f_k at fixed sampling points $y_1, ..., y_m$,
- Given u_k at sampling points $x_1, ..., x_L$. (x_l could depend on k)

Poisson problem:

$$\begin{aligned} \Delta u(x) &= f(x) && \text{on } \Omega \subset \mathbb{R}^2, \\ u(x) &= c && \text{for } x \in \partial\Omega. \end{aligned}$$

Data loss for K tuples (f_k, u_k):

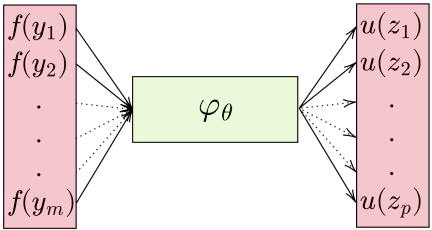
$$\frac{1}{KL} \sum_{k=1}^{K} \sum_{l=1}^{L} \left| \varphi_\theta \left(x_l, \begin{pmatrix} f_k(y_1) \\ \vdots \\ f_k(y_m) \end{pmatrix} \right) - u_k(x_l) \right|^2$$

TORCHPHYSICS provides both & combination

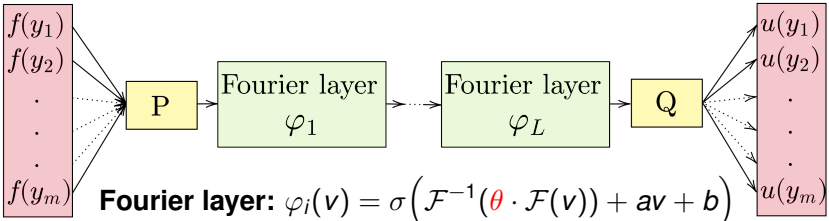
Other Operator Learning Approaches

Fully Discrete Approach

- Fixed sampling points $y_1, ..., y_m$ for functions f
- Fixed sampling points $z_1, ..., z_r$ for u
- Map sampled f to sampled u

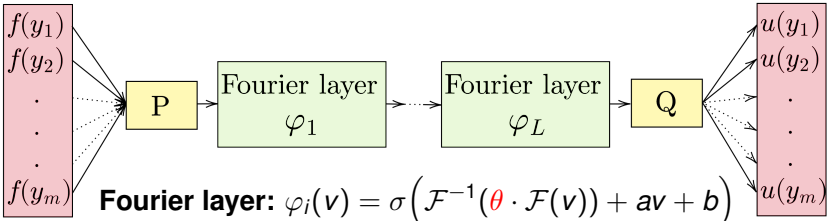


Fourier Neural Operators (FNO)[3]



[3] Li et al, *Fourier neural operator for parametric partial differential equations*, 2021

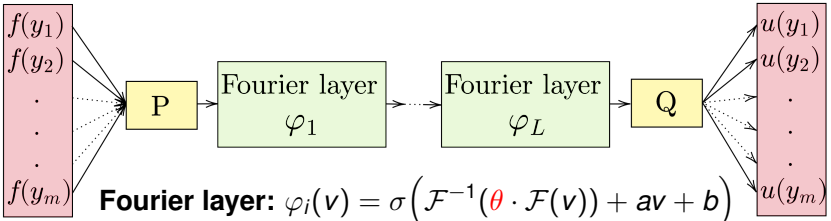
Fourier Neural Operators (FNO)[3]



- $y_1, ..., y_m$ are not fixed (next slide)

[3] Li et al, *Fourier neural operator for parametric partial differential equations*, 2021

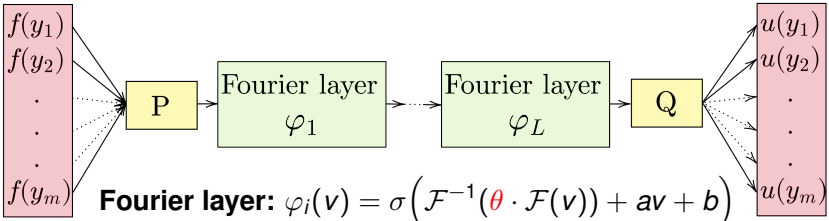
Fourier Neural Operators (FNO)[3]



- $y_1, ..., y_m$ are not fixed (next slide)
- Fourier layers: Convolution $\kappa * v = \mathcal{F}^{-1}(\mathcal{F}(\kappa) \cdot \mathcal{F}(v))$
- Filter θ in Fourier space $\longrightarrow$ Efficient global feature extraction

[3] Li et al, *Fourier neural operator for parametric partial differential equations*, 2021

FNO - Discretization Invariance

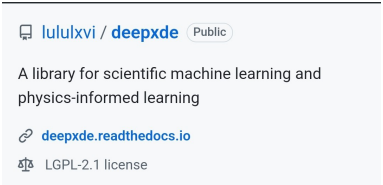


- FNOs have flexible input size m
- Weight $\theta \in \mathbb{R}^{k_{fix}}$ has fixed length; zero-padded to different sizes of $\mathcal{F}(v)$

Comparison to Other Python Libraries

DeepXDE[4]

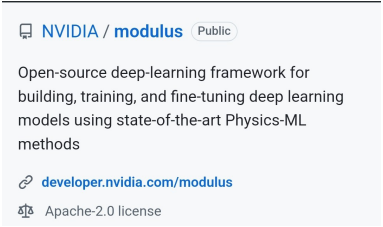
- Developed by L. Lu, supervised by G. Karniadakis, Brown University, USA
- Available on GitHub
- Implements all methods developed in the working group at Brown University
  - PINNs (and multiple extensions)
  - DeepONets (and multiple extensions)
  - ...
- Often used in research papers



[4] Lu et al, *DeepXDE: A deep learning library for solving differential equations*, 2021

NVIDIA MODULUS[5]

- Developed by the ©2021-2022, NVIDIA Corporation
- Previously known as NVIDIA SIMNET
- Available on GitHub
- Focus on operator learning and complex problems
  - FNO, DeepONets, ... (also physics-informed)
  - Used in weather and ocean forecasting, ...
- Bosch is currently developing a Wrapper that merges functionalities of TorchPhysics and Modulus



[5] Modulus Contributors, *NVIDIA Modulus: An open-source framework for physics-based deep ...*, 2023

Comparison

General information

- Underlying framework:
  - TorchPhysics, Modulus based on PyTorch
  - DeepXDE mainly TensorFlow and PaddlePaddle, but also JAX, PyTorch
- Installation:
  - All through pip-pipeline installable
  - Modulus: Docker and Modulus Sym package recommended
- Problem formulation:
  - All contain similar building blocks, Domain, Conditions, etc. (different names)
  - Differences in workflow

Feature comparison

Domains and sampling

	TorchPhysics	DeepXDE	Modulus
Domain operations $\cup, \cap, \setminus$	✓ Cartesian product	✓	✓
Time-dependent domains	✓	✗	✗
STL geometry	✓	✗	✓
Grid sampling	✓	✓	✗
Random sampling	✓	✓	✓
Adaptive sampling (e.g. loss-dependent)	✓	✓	(✓) Manually

Feature comparison

Pre-Implemented Methods

	TorchPhysics	DeepXDE	Modulus
PINN	✓	✓ Extensions	✓
DeepRitz	✓	Manually	Manually
(PI)DeepONet	✓	✓ Extensions	✓
FNO	coming soon	Fourier-DeepONet	✓
(PI)NO	work in progress	✗	✓
PCA-Net	✗	✗	✗

Comparison

Performance on simple PDE

$$-\Delta u = f \text{ in } \Omega$$
$$u = 0 \text{ on } \partial\Omega$$

- Starting example from yesterday
- Train for 5000 iterations on T4-GPU
- Same training procedure and network architecture in all frameworks

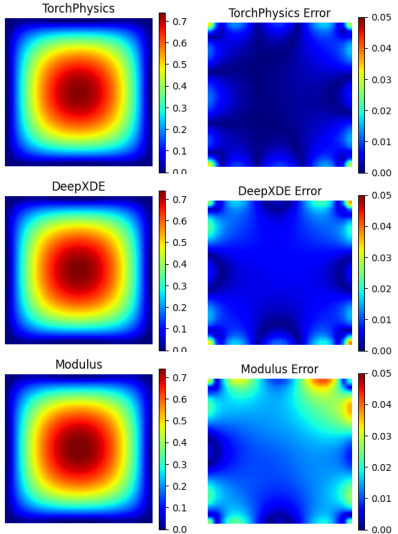
Comparison

Performance on simple PDE

$$-\Delta u = f \text{ in } \Omega$$
$$u = 0 \text{ on } \partial\Omega$$

- Starting example from yesterday
- Train for 5000 iterations on T4-GPU
- Same training procedure and network architecture in all frameworks

	TorchPhysics	DeepXDE	Modulus
Max error	0.033	0.037	0.041
Train time	66 s	39 s	83 s



Comparison

But...

- **Not** completely fair and does not generalize to other problems!
- Different aims and scopes of software packages
  - E.g. MODULUS aiming at complex and larger problems, parallelization, etc.
  - MODULUS and TORCHPHYSICS some indirect overhead through default logging and checkpoints
- PyTorch 2.0 functionalities in TORCHPHYSICS still under development
  → compiling to speed up training
- Other important aspects: tutorials, code clarity, expandability, provided features, ...

⇒ Each software has advantages and disadvantages, selection is user-dependent

Thank you for participating

Any final questions?

Thank you for participating

Any final questions?

We are:

- available by email or on GitHub for future questions
- always open for suggestions and feedback
- appreciating all contributions

TorchPhysics-Website

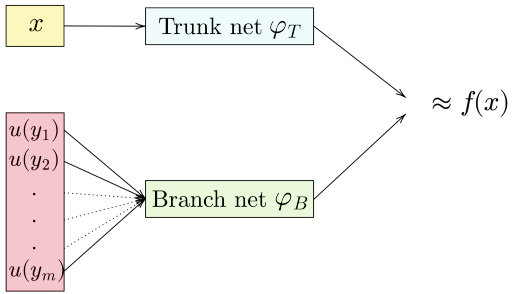
Appendix

DeepONets for Parameter Identification

(A) Learning the inverse Operator
(Direct method)

- Switch roles of f and u
- Train $\varphi_\theta(x, u) \approx f(x)$ data-driven
- Straightforward to implement

DeepONet $\varphi_\theta(x, u) \approx f(x)$

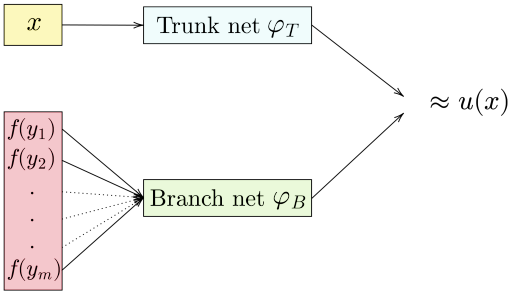


DeepONets for Parameter Identification

(B) Tikhonov scheme

- Train φ_θ on forward problem
  $$\varphi_\theta(x, f) \approx u(x)$$
  (data or physics-informed)

DeepONet $\varphi_\theta(x, f) \approx u(x)$



DeepONets for Parameter Identification

(B) Tikhonov scheme

- Train φ_θ on forward problem
  $$\varphi_\theta(x, f) \approx u(x)$$
  (data or physics-informed)

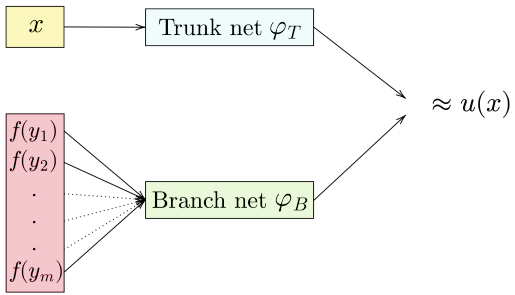
Given data $u^\delta(x_1), ..., u^\delta(x_l)$:

- Gradient descent for

$$\min_{f \in F} \sum_{k,l} \left|\varphi_\theta(x_l, f) - u^\delta(x_l)\right|^2 + \lambda \left|f(y_k)\right|^2$$

- Control noise robustness by λ

DeepONet $\varphi_\theta(x, f) \approx u(x)$

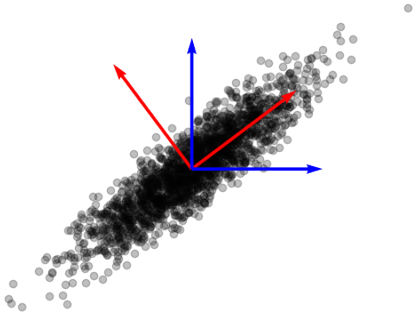


Principle Component Analysis (PCA) Networks[6]

1) Given data (f_k, u_k), find PCA-basis b_n

$$f_k \approx \sum_{n=1}^{N_{in}} c_n(f_k) b_n^F, \qquad u_k \approx \sum_{n=1}^{N_{out}} d_n(u_k) b_n^U.$$

Capture "nature" of data (figure, red).



[6] Bhattacharya et al, *Model reduction and neural networks for parametric PDEs*, 2020

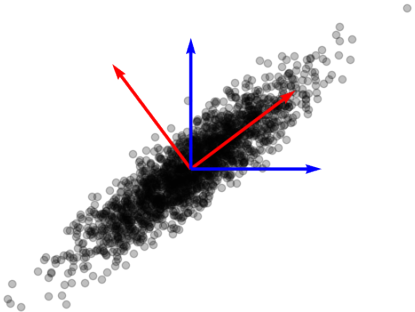
Principle Component Analysis (PCA) Networks[6]

1) Given data (f_k, u_k), find PCA-basis b_n

$$f_k \approx \sum_{n=1}^{N_{in}} c_n(f_k) b_n^F, \qquad u_k \approx \sum_{n=1}^{N_{out}} d_n(u_k) b_n^U.$$

Capture "nature" of data (figure, red).

2) Learn coefficient mapping $\varphi_\theta : c(f_k) \mapsto d(u_k)$



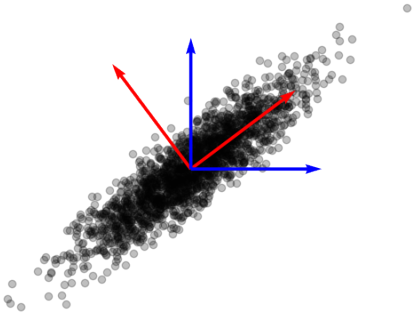
[6] Bhattacharya et al, *Model reduction and neural networks for parametric PDEs*, 2020

Principle Component Analysis (PCA) Networks[6]

1) Given data (f_k, u_k), find PCA-basis b_n

$$f_k \approx \sum_{n=1}^{N_{in}} c_n(f_k) b_n^F, \qquad u_k \approx \sum_{n=1}^{N_{out}} d_n(u_k) b_n^U.$$

Capture "nature" of data (figure, red).



2) Learn coefficient mapping $\varphi_\theta : c(f_k) \mapsto d(u_k)$

New f $\longrightarrow$ estimate $c(f)$ $\longrightarrow$ evaluate PCA-net φ_θ $\longrightarrow$ **solution** $\mathbf{u_f} \approx \sum_{n=1}^{N_{out}} \varphi_\theta\big(c(f)\big) b_n^U$

[6] Bhattacharya et al, *Model reduction and neural networks for parametric PDEs*, 2020

Appendix: Compare Performance of Different Methods

Performance of different approaches[7]

- Consider Darcy flow equation

$$\begin{aligned} -\nabla \cdot (f \nabla u) &= 1, \quad \text{in } (0,1)^2 \\ u &= 0, \quad \text{on } \partial(0,1)^2 \end{aligned}$$

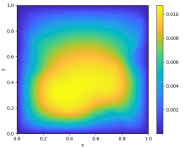
- **Forward problem:** map $f \to u$

[7] Nganyu Tanyu et al, *Deep Learning Methods for Partial Differential Equations (...)*, 2023

Performance of different approaches[7]

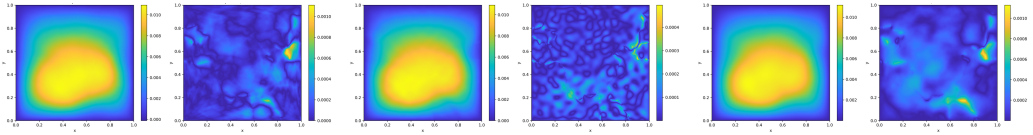
- Consider Darcy flow equation

$$-\nabla \cdot (f \nabla u) = 1, \quad \text{in } (0,1)^2$$
$$u = 0, \quad \text{on } \partial(0,1)^2$$



Ground Truth

- **Forward problem:** map $f \to u$



(a) DeepONet (b) FNO (c) PCA

[7] Nganyu Tanyu et al, *Deep Learning Methods for Partial Differential Equations (...)*, 2023

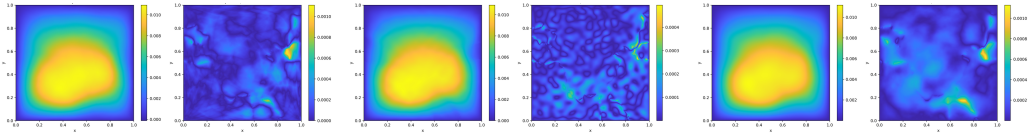
Performance of different approaches[7]

- Consider Darcy flow equation

$$-\nabla \cdot (f \nabla u) = 1, \quad \text{in } (0,1)^2$$
$$u = 0, \quad \text{on } \partial(0,1)^2$$

	Rel. L^2 error	Evaluation time [s]
DeepONet	0.029	0.001
FNO	0.011	0.017
PCA	0.025	0.611

- **Forward problem:** map $f \to u$



(a) DeepONet (b) FNO (c) PCA

[7] Nganyu Tanyu et al, *Deep Learning Methods for Partial Differential Equations (...)*, 2023

Performance of different approaches[6]

- Consider Darcy flow equation

$$\begin{aligned} -\nabla \cdot (f \nabla u) &= 1, \quad \text{in } (0,1)^2 \\ u &= 0, \quad \text{on } \partial(0,1)^2 \end{aligned}$$

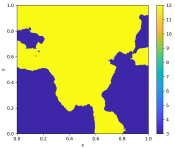
- **Inverse problem:** map $u \to f$

[6] Nganyu Tanyu et al, *Deep Learning Methods for Partial Differential Equations (...)*, 2023

Performance of different approaches[6]

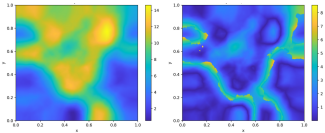
- Consider Darcy flow equation

$$-\nabla \cdot (f \nabla u) = 1, \quad \text{in } (0,1)^2$$
$$u = 0, \quad \text{on } \partial(0,1)^2$$

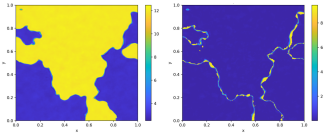


Ground Truth

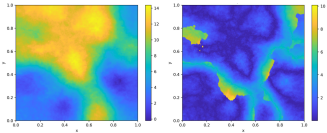
- **Inverse problem:** map $u \to f$



(a) DeepONet



(b) FNO



(c) PCA

[6] Nganyu Tanyu et al, *Deep Learning Methods for Partial Differential Equations (...)*, 2023

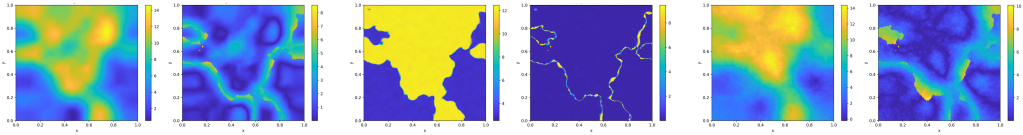
Performance of different approaches[6]

- Consider Darcy flow equation

$$-\nabla \cdot (f \nabla u) = 1, \quad \text{in } (0,1)^2$$
$$u = 0, \quad \text{on } \partial(0,1)^2$$

	Rel. L^2 error	Evaluation time [s]
DeepONet	0.222	0.001
FNO	0.149	0.016
PCA	0.099	0.154

- **Inverse problem:** map $u \to f$



(a) DeepONet (b) FNO (c) PCA

[6] Nganyu Tanyu et al, *Deep Learning Methods for Partial Differential Equations (...)*, 2023