Universität
Bremen

Center for Industrial
Mathematics (ZeTeM)
Faculty 03
Mathematics / Computer science

Deep Learning for PDE based Forward and Inverse Problems

Tom Freudenberg, N. Heilenkötter, D. Nganyu, J. Gödeke, P. Maaß, U. Iben

04.12.2023

Topics

1. Basics of Deep Learning and physics informed approaches
2. TorchPhysics: a software for PI-Deep Learning
3. Application study to inverse problems

Basics of Deep Learning

What is a Neural Network (NN)?

- Parameterized function

$$\begin{aligned} u : \Theta \times \mathbb{R}^n &\longrightarrow \mathbb{R}^m \\ (\theta, x) &\longmapsto u(\theta, x) \end{aligned}$$

  parameter space $\Theta \subseteq \mathbb{R}^p$

- **Notation:** $u_\theta(x) := u(\theta, x)$

Basics of Deep Learning

What is a Neural Network (NN)?

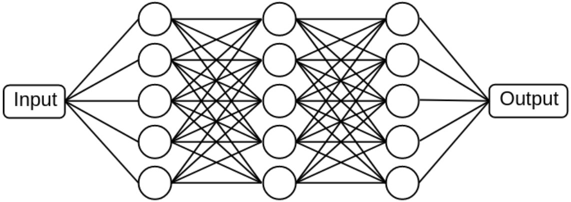
- Parameterized function

$$\begin{aligned} u : \Theta \times \mathbb{R}^n &\longrightarrow \mathbb{R}^m \\ (\theta, x) &\longmapsto u(\theta, x) \end{aligned}$$

parameter space $\Theta \subseteq \mathbb{R}^p$

- **Notation:** $u_\theta(x) := u(\theta, x)$

Fully Connected NN:



Basics of Deep Learning

What is a Neural Network (NN)?

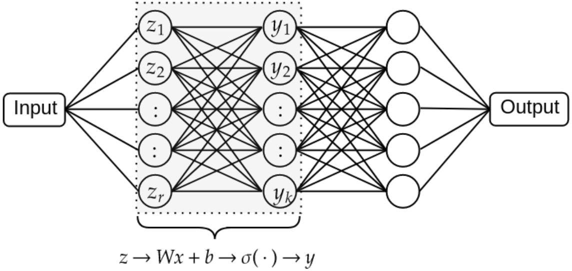
- Parameterized function

$$u : \Theta \times \mathbb{R}^n \longrightarrow \mathbb{R}^m$$
$$(\theta, x) \longmapsto u(\theta, x)$$

  parameter space $\Theta \subseteq \mathbb{R}^p$

- **Notation:** $u_\theta(x) := u(\theta, x)$

Fully Connected NN:



- Weight matrix $W \in \mathbb{R}^{k \times r}$ and bias $b \in \mathbb{R}^k$ belong to parameters θ
- Activation function $\sigma : \mathbb{R} \to \mathbb{R}$ applied coordinate-wise

Universal Approximation Theorem[1]

Theorem (Hornik, 1989)

Let $K \subset \mathbb{R}^n$ be compact and consider a continuous function

$$f : K \subset \mathbb{R}^n \to \mathbb{R}^m.$$

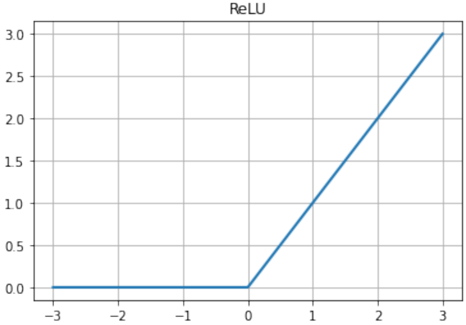
For each error ε ***there exists*** *a NN u_θ with N hidden neurons and* ***"specific"*** *activations that uniformly approximates f, i.e.*

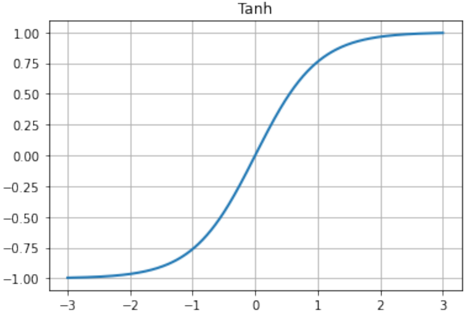
$$\|f(x) - u_\theta(x)\| < \varepsilon \quad \text{for every} \quad x \in K.$$

[1] Hornik: *Multilayer Feedforward Networks are Universal Approximators*, 1989

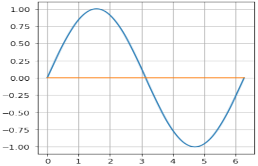
"Specific" activations

For example:

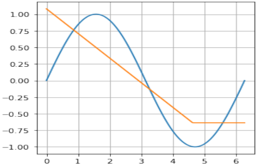




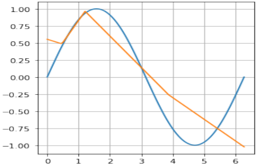
Example of Approximation Properties



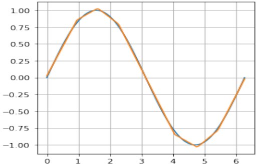
(a) 1 layer with 1 neuron



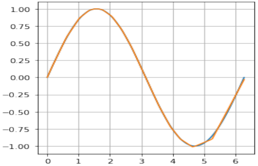
(b) 1 layer with 2 neurons



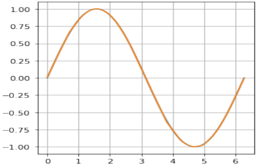
(c) 1 layer with 5 neurons



(d) 1 layer with 10 neurons



(e) 1 layer with 50 neurons



(f) 1 layer with 100 neurons

Motivation: Why Deep Learning for DEs?

Parameter identification/optimization problems

$\rightarrow$ Iterative algorithms: Solve many **similar** PDEs

$\rightarrow$ Classical methods like FDM or FEM: Time-consuming

$\rightarrow$ Replace by trained NN

$\rightarrow$ Less time-consuming

DL for DEs: Data-Driven Approach

- For example, Harmonic Oscillator: $\begin{cases} \partial_t^2 u(t) = -\lambda u(t) \\ u(0) = u_0,\ \partial_t u(0) = 0 \end{cases}$
- Generate/obtain data $\{\tilde{u}_j, t_j, \lambda_j\}_{j=1}^N$, with $\tilde{u}_j \approx u(t_j; \lambda_j)$
- Initialize a neural network $u_\theta : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$
- Train u_θ by minimizing

$$\mathcal{L}(\theta_i) = \frac{1}{N} \sum_{j=1}^{N} \left| u_{\theta_i}(t_j, \lambda_j) - \tilde{u}_j \right|^2 \quad \text{(Mean-Squared-Error)}$$

with gradient descent: $\theta_{i+1} = \theta_i - \eta \nabla_\theta \mathcal{L}(\theta_i)$

Problems of Data-Driven Approach

- Deep Learning generally needs lot of data
- Obtaining data of solution u is complicated
  - Through multiple experiments
  - Solving the equation with classical methods

  → Expensive and time consuming

Problems of Data-Driven Approach

- Deep Learning generally needs lot of data
- Obtaining data of solution u is complicated
  - Through multiple experiments
  - Solving the equation with classical methods

  → Expensive and time consuming

💡 Encode physical laws/PDEs into DL approaches?

→ **Physics-informed neural networks (PINNs)**

→ Plug neural network into the differential equation

Physics-Informed Neural Networks (PINNs)

PINNs[2] - Main Idea

- Find solution $\mathbf{u} : \Omega \subset \mathbb{R}^n \to \mathbb{R}^m$ of

$$\begin{aligned} \mathcal{N}[\mathbf{u}](x) &= 0, \text{ for } x \in \Omega, \\ \mathcal{B}[\mathbf{u}](x) &= 0, \text{ for } x \in \partial\Omega. \end{aligned}$$

- E.g. $\Omega = [0,1] \times [0,1], \mathbf{u} : \mathbb{R}^2 \to \mathbb{R}$

$$\begin{aligned} \mathcal{N}[\mathbf{u}](x) &= \Delta \mathbf{u}(x) - f(x), \text{ for } x \in \Omega, \\ \mathcal{B}[\mathbf{u}](x) &= \mathbf{u}(x) - u_0, \qquad \text{ for } x \in \partial\Omega. \end{aligned}$$

[2] Raissi, Perdikaris and Karniadakis: *Physics-informed neural networks: [...]*, 2019

PINNs - Main Idea

- Find solution $\mathbf{u} : \Omega \subset \mathbb{R}^n \to \mathbb{R}^m$ of

$$\begin{aligned} \mathcal{N}[\mathbf{u}](x) &= 0, \text{ for } x \in \Omega, \\ \mathcal{B}[\mathbf{u}](x) &= 0, \text{ for } x \in \partial\Omega. \end{aligned}$$

- Sample points $x_i^{\mathcal{N}} \in \Omega$ and $x_j^{\mathcal{B}} \in \partial\Omega$
- Train network $\mathbf{u}_\theta$ that minimizes the PDE-loss

$$\frac{1}{N_{\mathcal{N}}} \sum_{i=1}^{N_{\mathcal{N}}} \left\| \mathcal{N}[\mathbf{u}_\theta](x_i^{\mathcal{N}}) \right\|^2 + \frac{1}{N_{\mathcal{B}}} \sum_{j=1}^{N_{\mathcal{B}}} \left\| \mathcal{B}[\mathbf{u}_\theta](x_j^{\mathcal{B}}) \right\|^2$$

Physics-Informed Loss: We need to ...

- Compute differential operator $\mathcal{N}$ of NN u_θ, e.g. Laplacian Δu_θ
  - $\rightarrow$ Compute the derivatives of a neural network
  - $\rightarrow$ Generally possible with basics math operations

Physics-Informed Loss: We need to ...

- Compute differential operator $\mathcal{N}$ of NN u_θ, e.g. Laplacian Δu_θ
  - $\rightarrow$ Compute the derivatives of a neural network
  - $\rightarrow$ Generally possible with basics math operations
- Consider $u_\theta : \mathbb{R} \rightarrow \mathbb{R}$, with $u_\theta(x) = \sigma(W_2\sigma(W_1 x + b_1) + b_2)$, then:

$$\partial_x u_\theta(x) = \partial_x\sigma(W_2\sigma(W_1 x + b_1) + b_2)[W_2^T \partial_x\sigma(W_1 x + b_1) W_1]$$
$$\partial_x^2 u_\theta(x) = \dots$$

$\rightarrow$ Derivatives contain the same network parameters $\theta = (W_1, b_1, W_2, b_2)$

Parameter Studies

Realization with PINNs

- Many applications involve solving the same PDE with different parameters $c \in \mathbb{R}^d$:

$$\mathcal{N}[\mathbf{u}_c, c](x) = 0, \text{ for } x \in \Omega,$$

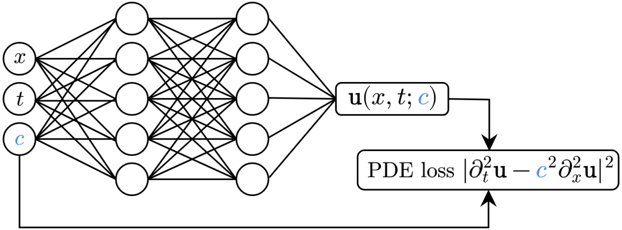
- For example:

Parameter-dependent wave equation:

$$\begin{cases} \partial_t^2 u = c^2\, \partial_x^2 u, & \text{in } I_x \times I_t, \\ u = 0 & \text{in } \partial I_x \times I_t, \\ \partial_t u(\cdot, 0) = 0 & \text{in } I_x, \\ u(\cdot, 0) = \sin(x) & \text{in } I_x, \end{cases}$$

Parameter Studies with PINNs

Solving the same PDE for many different choices of c

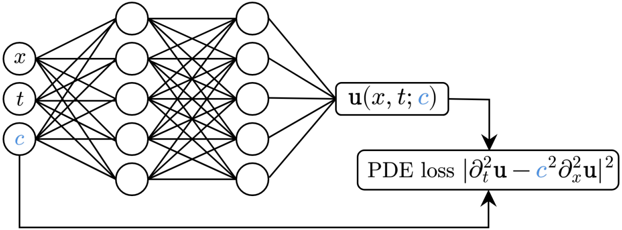


Method:

- Include parameter(s) as additional input(s) to the PINN
- Training: Sample parameter range together with function domain

Parameter Studies with PINNs

Solving the same PDE for many different choices of c

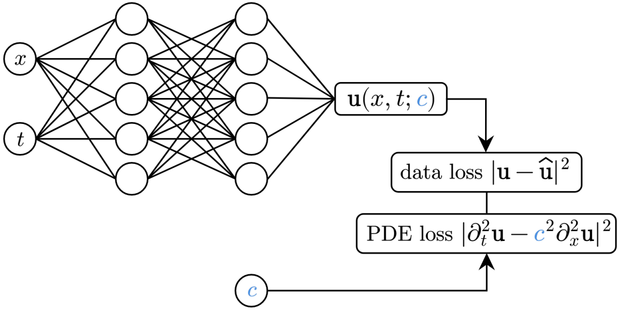


Result:

- Inference of solution for new parameter by a forward pass to the trained network
- Very little additional effort in evaluation of the network
- Increased amount of training points necessary

Parameter Identification with PINNs

Finding the c that leads to given solution data $\{\hat{u}_i\}$



Method:

- Include parameter(s) as learnable parameter(s)
- Training: Incorporate data loss in training
  $\rightsquigarrow$ Goal: Find a solution that fits data and solves PDE for the optimized parameter

TorchPhysics

Initiation of TorchPhysics

BOSCH

- 2021: Robert Bosch GmbH got interested in PINNs
- Technical applications:
  Car parts, electronics, injection molding, etc.

©factum

[a] Paszke et al., *PyTorch: An Imperative Style [...]*, 2019

Initiation of TORCHPHYSICS

- 2021: Robert Bosch GmbH got interested in PINNs
- Technical applications:
  Car parts, electronics, injection molding, etc.
- Student project: Deep Learning library for PDEs
- Main Developers: Nick Heilenkötter & Tom Freudenberg

BOSCH

©factum

[a] Paszke et al., *PyTorch: An Imperative Style [...]*, 2019

Initiation of TorchPhysics

- 2021: Robert Bosch GmbH got interested in PINNs
- Technical applications:
  Car parts, electronics, injection molding, etc.
- Student project: Deep Learning library for PDEs
- Main Developers: Nick Heilenkötter & Tom Freudenberg
- Open-Source on GitHub
- Build upon PyTorch[a]

[a] Paszke et al., *PyTorch: An Imperative Style [...]*, 2019

BOSCH

©factum

What is needed for a PINN library?

We need...

- to create different types of domains
- a way to sample points in a given domain
- to be able to define different network architectures
- implement the need differential equations and boundary conditions as training conditions

What is needed for a PINN library?

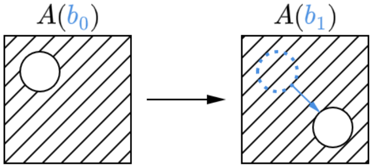
We need...

- to create different types of domains
- a way to **sample points in a given domain**
- to be able to define different network architectures
- implement the need differential equations and boundary conditions as training conditions

Domains

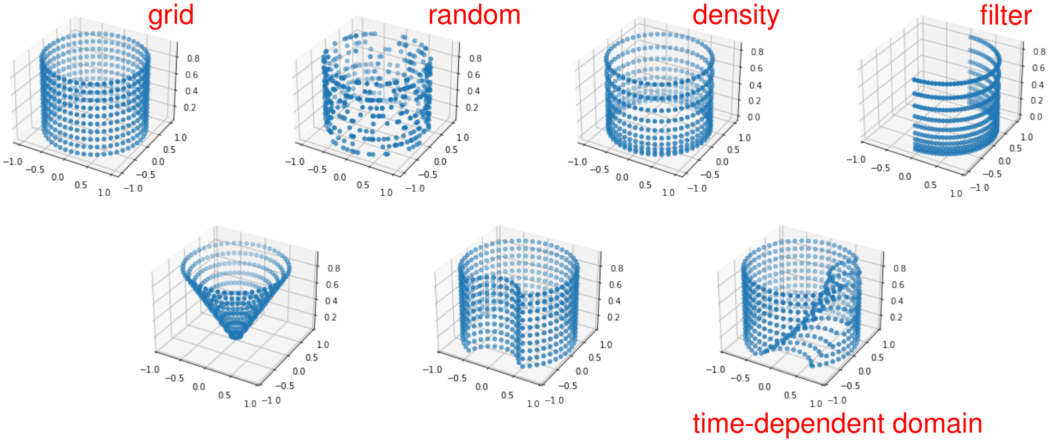
- Basic geometries implemented:
  - `Point, Interval, Parallelogram, Circle, ...`
- Complex domains via logical operators:

- Domains can depend on variables of other domains (e.g. time-dependent)



PointSampler

Flexible domain sampling

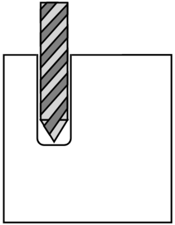


Applications

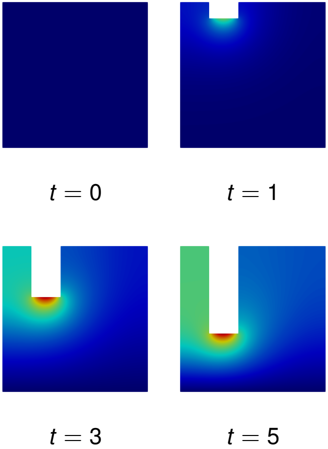
Temperature in a drilling process

- Drilling problem:

$$
\begin{aligned}
\partial_t u(x,t) - \kappa \Delta u(x,t) &= 0, && \text{in } \Omega(t),\\
u(x,0) &= 0, && \text{in } \Omega(0),\\
u(x,t) &= 0, && \text{on } \Gamma_D,\\
\kappa \nabla u(x,t) \cdot n &= f, && \text{on } \Gamma_N(t).
\end{aligned}
$$



- L^2–difference to FEM $\sim$7 %

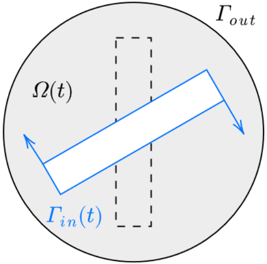


Applications

PINNs for Temperature–Navier–Stokes example

Bar heats up and rotates within some fluid:

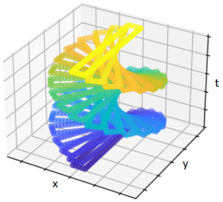
$$
\begin{aligned}
\partial_t u + (u \cdot \nabla) u &= \nu \Delta u - \nabla p, && \text{in } \Omega(t),\\
\nabla \cdot u &= 0, && \text{in } \Omega(t),\\
\partial_t T + u \cdot \nabla T &= \lambda \Delta T, && \text{in } \Omega(t),\\
u(0,\cdot), p(0,\cdot), T(0,\cdot) &= 0, 0, 270 && \text{in } \Omega(0),\\
u, T &= (0,0), 270, && \text{on } \Gamma_{\text{out}},\\
u, T &= u_{\text{in}}(t), T_{\text{in}}(t), && \text{on } \Gamma_{\text{in}}.
\end{aligned}
$$

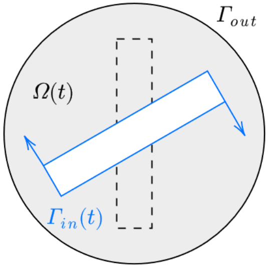


Time-Dependent Domain

Implementation inside TorchPhysics

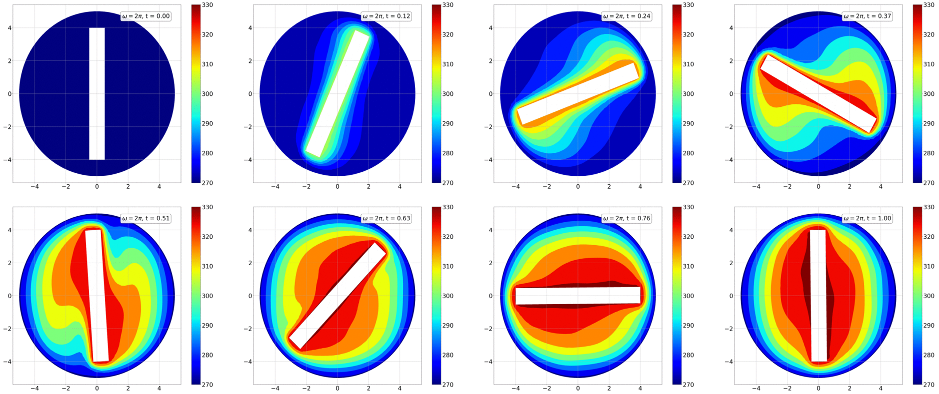
```
def corner1(t):
    return rotation_matrix(t) * start_position_1

bar    = tp.domains.Parallelogram(X, corner1, corner2, corner3)
circle = tp.domains.Circle(X, center, radius)

omega  = circle - bar
```





Applications

Learned Temperature Field



Advantages of PINNs

Compared to classical methods

- (usually) Grid/mesh independent, therefore more flexible & saving is usually more memory efficient
- General approach for different kinds of differential equations, especially nonlinear
- Learning parameter dependencies, useful for parameter studies
- Extension to optimization- & inverse problems easy to implement
- Interpolation and extrapolation of data

Disadvantages of PINNs
Compared to classical methods

- No convergence theory
- Error not arbitrarily small
- Sometimes optimal minimum difficult to find, poor convergence
- Much slower for single computation of forward solutions
- Often trial and error for finding good parameters

Application to Inverse Problems

Operator Learning

First natural idea

- Many problems include function valued parameters $f : \mathbb{R}^d \to \mathbb{R}^m$
- **Goal**: Learn operator on function set F

$$\begin{aligned} \Phi_\theta : \mathbb{R}^n \times F &\longrightarrow \mathbb{R}^m \\ (x, f) &\longmapsto u_f(x) \end{aligned}$$

- **Problem**: Inputs of NN has to be discrete

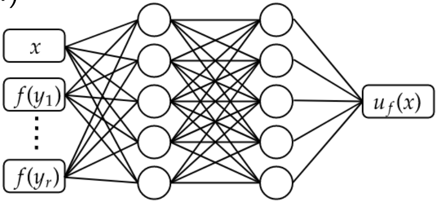
Operator Learning

First natural idea

- Many problems include function valued parameters $f : \mathbb{R}^d \to \mathbb{R}^m$
- **Goal**: Learn operator on function set F

$$\begin{aligned} \Phi_\theta : \mathbb{R}^n \times F &\longrightarrow \mathbb{R}^m \\ (x, f) &\longmapsto u_f(x) \end{aligned}$$

- **Problem**: Inputs of NN has to be discrete
- **Idea**: Discretize f on $y_1, \dots, y_r \in \mathbb{R}^d$



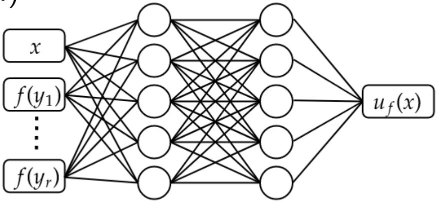
Operator Learning

First natural idea

- Many problems include function valued parameters $f : \mathbb{R}^d \to \mathbb{R}^m$
- **Goal**: Learn operator on function set F

$$\begin{aligned} \Phi_\theta : \mathbb{R}^n \times F &\longrightarrow \mathbb{R}^m \\ (x, f) &\longmapsto u_f(x) \end{aligned}$$

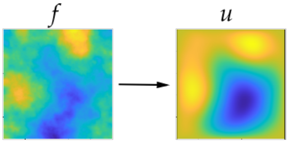
- **Problem**: Inputs of NN has to be discrete
- **Idea**: Discretize f on $y_1, \ldots, y_r \in \mathbb{R}^d$
- Many $f(y_i)$-inputs versus $x \to$ **Imbalance**



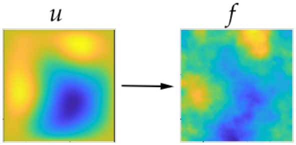
Operator Learning

State of the art

- Keep idea of discrete $f(y_i)$-inputs, change network architecture:
  - **DeepONet** [Lu et al. (2019)] - Divide and Conquer
  - **Fourier Neural Operator** (FNO) [Li et al. (2020)] - Discrete Fourier transform
  - **PCANN** [Bhattacharya et al. (2020)] - Principle component analysis
- Generally data-driven, but with physics informed extensions



Forward mapping



Inverse mapping

Operator Learning

For inverse problems

- Usually the mapping $u \mapsto f$ is unstable under noisy data u^δ
- **Idea**: Learn forward operator Φ_θ and use it in Tikhonov scheme

$$\min_{f \in F} \|\Phi_\theta(f) - u^\delta\|^2 + \alpha R(f)$$

[3] Nganyu et al., *Deep Learning Methods for Partial Differential Equations [...]*, 2023

Operator Learning

For inverse problems

- Usually the mapping $u \mapsto f$ is unstable under noisy data u^δ
- **Idea**: Learn forward operator Φ_θ and use it in Tikhonov scheme

$$\min_{f \in F} \|\Phi_\theta(f) - u^\delta\|^2 + \alpha R(f)$$

- We studied[3]:
  - Performance of different methods for forward and inverse problem
  - Influence of noise in the inverse problem
- Training: 1000 data pairs (f, u_f), Testing: 5000 additional data pairs

[3] Nganyu et al., *Deep Learning Methods for Partial Differential Equations [...]*, 2023

Forward Problem

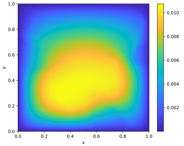
- Consider Darcy flow equation

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

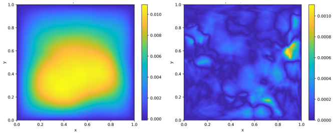
Forward Problem

- Consider Darcy flow equation

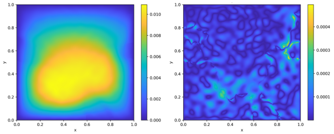
$$-\nabla \cdot (f \nabla u) = 1, \quad \text{in } (0,1)^2$$
$$u = 0, \quad \text{on } \partial(0,1)^2$$



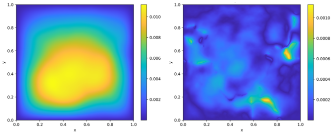
Ground Truth



(a) DeepONet



(b) FNO



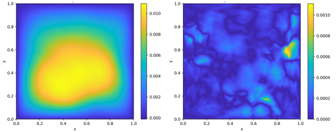
(c) PCANN

Forward Problem

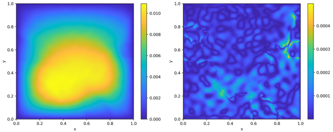
- Consider Darcy flow equation

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

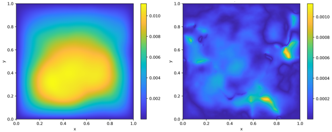
	Rel. L^2 error	Evaluation time [s]
DeepONet	0.029	0.001
FNO	0.011	0.017
PCANN	0.025	0.611



(a) DeepONet



(b) FNO



(c) PCANN

Inverse Problem

Without noise

- Consider Darcy flow equation

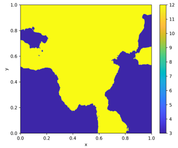
$$
\begin{aligned}
-\nabla \cdot (f \nabla u) &= 1, \quad \text{in } (0,1)^2 \\
u &= 0, \quad \text{on } \partial(0,1)^2
\end{aligned}
$$

Inverse Problem

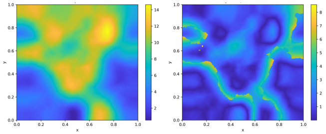
Without noise

- Consider Darcy flow equation

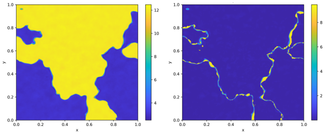
$$
\begin{aligned}
-\nabla \cdot (f \nabla u) &= 1, \quad \text{in } (0,1)^2 \\
u &= 0, \quad \text{on } \partial(0,1)^2
\end{aligned}
$$



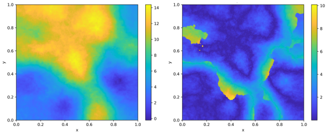
Ground Truth



(a) DeepONet



(b) FNO



(c) PCANN

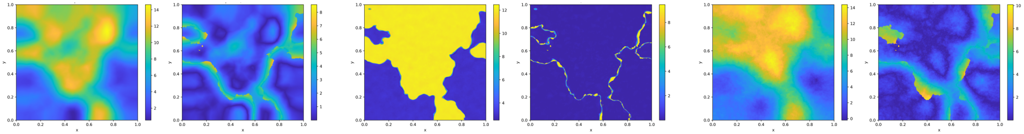
Inverse Problem

Without noise

- 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	0.093	0.016
PCANN	0.098	0.154



(a) DeepONet (b) FNO (c) PCANN

Inverse Problem

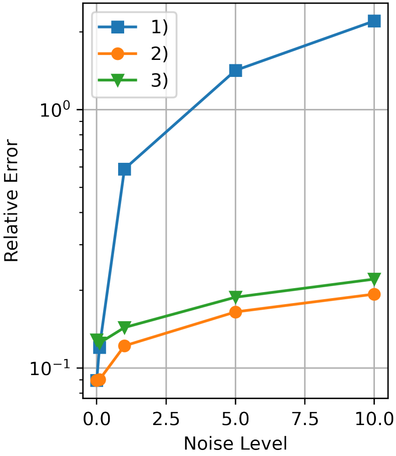
With noise

- Three different training strategies:
  1) $u \mapsto f$, with noise-free data
  2) $u \mapsto f$, with noisy data of the same noise level
  3) $f \mapsto u$ and then Tikhonov for the inverse problem
- Always evaluate on noisy data u^δ
- Only demonstrate FNO

Inverse Problem
With noise

- Three different training strategies:
  1) $u \mapsto f$, with noise-free data
  2) $u \mapsto f$, with noisy data of the same noise level
  3) $f \mapsto u$ and then Tikhonov for the inverse problem
- Always evaluate on noisy data u^δ
- Only demonstrate FNO

$\rightarrow$ Tikhonov helpful if noise level not previously known



Summary

- DL for differential equations is useful for parameter studies, control/inverse problems, and data extrapolation
- Disadvantages are sometimes poor convergence and not arbitrarily small error
- TorchPhysics is a open source framework that allows simple implementation of many different problems
- Use of learned forward operator in classical Tikhonov speeds up the computation