Universität Bremen

Center for Industrial Mathematics (ZeTeM)

Faculty 03

Mathematics / Computer science

Introduction to TorchPhysics

Deep Learning Library for Differential Equations

Janek Gödeke, Nick Heilenkötter, Tom Freudenberg

Heidelberg, 07.11.2023

Goals of TorchPhysics-Workshop

Overall: Usage of TorchPhysics for

- Physics-informed DL
- Data-based DL

} for ODEs / PDEs / Parameter identification

- Function and Operator Learning

Goals of TorchPhysics-Workshop

Overall: Usage of TorchPhysics for

- Physics-informed DL
- Data-based DL

} for ODEs / PDEs / Parameter identification

- Function and Operator Learning

Today:

- Learning solution functions of PDEs (physics-informed)
- Flexible domain creation in TorchPhysics
- Implementing basic boundary conditions

Recall: Neural Networks

What is a Neural Network (NN)?

- Parameterized function

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

  parameter space $\Theta \subseteq \mathbb{R}^p$
- **Notation:** $u_\theta(x) := u(\theta, x)$

What is a Neural Network (NN)?

- Parameterized function

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

  parameter space $\Theta \subseteq \mathbb{R}^p$
- **Notation:** $u_\theta(x) := u(\theta, x)$
- Number of input neurons: d
- Number of output neurons: n

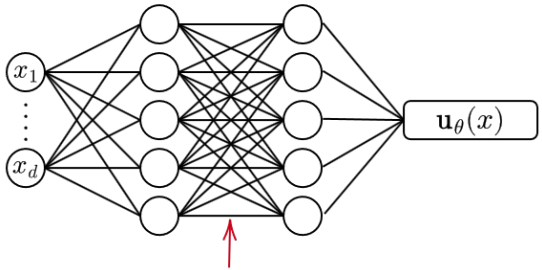
What is a Neural Network (NN)?

- Parameterized function

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

  parameter space $\Theta \subseteq \mathbb{R}^p$
- **Notation:** $u_\theta(x) := u(\theta, x)$
- Number of input neurons: d
- Number of output neurons: n

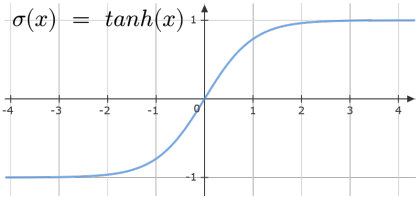
Fully Connected NN:



Typical Layers of NNs

- Layers of u_θ are functions of the form

$$\mathbb{R}^l \longrightarrow \mathbb{R}^m$$
$$y \longmapsto \sigma(Ay + b)$$



- Matrix A and bias b belong to parameters θ
- Activation function $\sigma : \mathbb{R} \to \mathbb{R}$ applied coordinate-wise

Training NNs with Data

Given: Data points

$$(x_1, u_1), ..., (x_k, u_k)$$

Goal: Adjust parameters $\theta \in \Theta$ s.t.

$$u_\theta(x_i) \approx u_i$$

Training NNs with Data

Given: Data points

$$(x_1, u_1), ..., (x_k, u_k)$$

Goal: Adjust parameters $\theta \in \Theta$ s.t.

$$u_\theta(x_i) \approx u_i$$

Minimize (data-)loss

$$\ell(\theta) := \frac{1}{k} \sum_{i=1}^{k} \left| u_\theta(x_i) - u_i \right|^2$$

by gradient descent algorithms, e.g.

$$\theta_{j+1} = \theta_j - \mathrm{lr} \cdot \nabla \ell(\theta_j), \quad \text{learning rate lr}$$

Deep Learning for PDEs

Different Learning Tasks - Function Learning

Poisson equation:

$$\begin{aligned} \Delta u(x) &= f(x) \quad \text{on } \Omega \subset \mathbb{R}^2, \\ u(x) &= u_0 \quad \text{for } x \in \partial\Omega. \end{aligned}$$

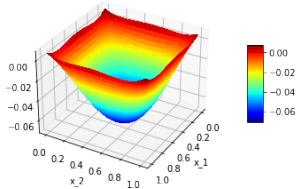


Figure: Learned solution for $f = 1$ and $u_0 = 0$

Different Learning Tasks - Function Learning

Poisson equation:

$$\Delta u(x) = f(x) \quad \text{on } \Omega \subset \mathbb{R}^2,$$
$$u(x) = u_0 \quad \text{for } x \in \partial\Omega.$$

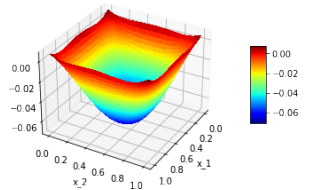


Figure: Learned solution for $f = 1$ and $u_0 = 0$

1) Find solution u for **single** parameter $u_0 \in \mathbb{R}$
function $f : \Omega \to \mathbb{R}$

Neural network u_θ, weights θ

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

Different Learning Tasks - Function Learning

Poisson equation:

$$\Delta u(x) = f(x) \quad \text{on } \Omega \subset \mathbb{R}^2,$$
$$u(x) = u_0 \quad \text{for } x \in \partial\Omega.$$

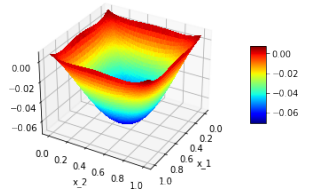


Figure: Learned solution for $f = 1$ and $u_0 = 0$

1) Find solution u for **single** parameter $u_0 \in \mathbb{R}$
function $f : \Omega \to \mathbb{R}$

Neural network u_θ, weights θ

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

2) Find solutions u_{u_0} for **all** $u_0 \in [a, b]$

$$u_\theta : \Omega \times [a, b] \longrightarrow \mathbb{R}$$
$$u_\theta(x, u_0) \approx u_{u_0}(x)$$

Different Learning Tasks - Operator Learning

Poisson equation:

$$\begin{aligned} \Delta u(x) &= f(x) && \text{on } \Omega \subset \mathbb{R}^2, \\ u(x) &= u_0 && \text{for } x \in \partial\Omega. \end{aligned}$$

3) Find solutions u_f for **many** functions $f\colon \Omega \to \mathbb{R}$

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

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

Different Learning Tasks - Operator Learning

Poisson equation:

$$\begin{aligned} \Delta u(x) &= f(x) \quad \text{on } \Omega \subset \mathbb{R}^2, \\ u(x) &= u_0 \quad \text{for } x \in \partial\Omega. \end{aligned}$$

3) Find solutions u_f for **many** functions $f\colon \Omega \to \mathbb{R}$

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

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

Problem: Input of NNs must be from $\mathbb{R}^n$

$\longrightarrow$ Considered tomorrow

Today's Learning Tasks

Function Learning with PINNs

1) Find solution u for **single** parameter $u_0 \in \mathbb{R}$ function $f : \Omega \to \mathbb{R}$

Poisson equation:

$$\begin{aligned} \Delta u(x) &= f(x) \quad \text{on } \Omega \subset \mathbb{R}^2, \\ u(x) &= u_0 \quad \text{for } x \in \partial\Omega. \end{aligned}$$

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

Today's Learning Tasks

Function Learning with PINNs

Poisson equation:

$$\begin{aligned} \Delta u(x) &= f(x) \quad \text{on } \Omega \subset \mathbb{R}^2, \\ u(x) &= u_0 \quad \text{for } x \in \partial\Omega. \end{aligned}$$

1) Find solution u for **single** parameter $u_0 \in \mathbb{R}$ function $f : \Omega \to \mathbb{R}$

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

- Get along with structure of TorchPhysics
- Get in touch with physics-informed DL

Physics-Informed Neural Networks (PINNs)

PINNs

- **Physics-informed neural networks: A deep learning framework for solving (...) partial differential equations**
  Authors: Raissi, Perdikaris, Karniadakis
  Journal of Computational Physics, 2019

PINNs

- **Physics-informed neural networks: A deep learning framework for solving (...) partial differential equations**
  Authors: Raissi, Perdikaris, Karniadakis
  Journal of Computational Physics, 2019

- **"Physics-Informed":** Include PDE into loss function

- No data is required
  (combination might be beneficial, though)

PINNs - Main Idea

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

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

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

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

PINNs - Main Idea

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

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

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

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

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

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

PINNs - Main Idea

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

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

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

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

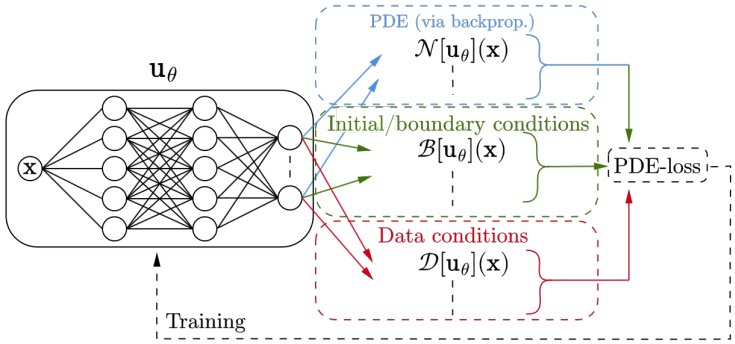
- Sample points $x_i^{\mathcal{N}} \in \Omega$ and $x_j^{\mathcal{B}} \in \partial\Omega$
- Train network $\mathbf{u}_\theta$ minimizing the PDE-loss

Data available?

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

PINN-Approach

Schematic depiction



$$u_\theta = \arg\min_{\mathbf{u}} \frac{\omega_{\mathcal{N}}}{N_{\mathcal{N}}} \sum_{i=0}^{N_{\mathcal{N}}} \left\| \mathcal{N}[\mathbf{u}]\left(x_i^{\mathcal{N}}\right) \right\|^2 + \frac{\omega_{\mathcal{B}}}{N_{\mathcal{B}}} \sum_{i=0}^{N_{\mathcal{B}}} \left\| \mathcal{B}[\mathbf{u}]\left(x_i^{\mathcal{B}}\right) \right\|^2 + \frac{\omega_{\mathcal{D}}}{N_{\mathcal{D}}} \sum_{i=0}^{N_{\mathcal{D}}} \left\| \mathcal{D}[\mathbf{u}]\left(x_i^{\mathcal{D}}\right) \right\|^2$$

Requirements for Physics-Informed Training

One needs to:

- Compute differential operators of u_θ, e.g. Laplacian Δu_θ

  $\rightarrow$ Enables PDE-loss computation

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

Requirements for Physics-Informed Training

One needs to:

- Compute differential operators of u_θ, e.g. Laplacian Δu_θ

  $\rightarrow$ Enables PDE-loss computation

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

- Differentiate PDE-loss w.r.t. network parameters θ

Requirements for Physics-Informed Training

One needs to:

- Compute differential operators of u_θ, e.g. Laplacian Δu_θ

  $\rightarrow$ Enables PDE-loss computation

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

- Differentiate PDE-loss w.r.t. network parameters θ

TorchPhysics does this for us!

- Makes use of PyTorch's autograd functionality

PINNs - Advantages

Compared to classical methods

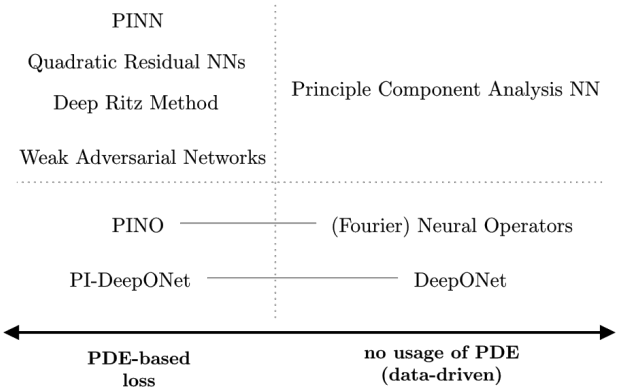
- Grid/mesh independent $\longrightarrow$ more flexible & saving more memory efficient
- General approach for different kinds of differential equations, e.g. nonlinear
- Learning parameter dependencies
- Extension to optimization- & inverse problems easy to implement
- Parallelizable on multiple GPUs

PINNs - Disadvantages

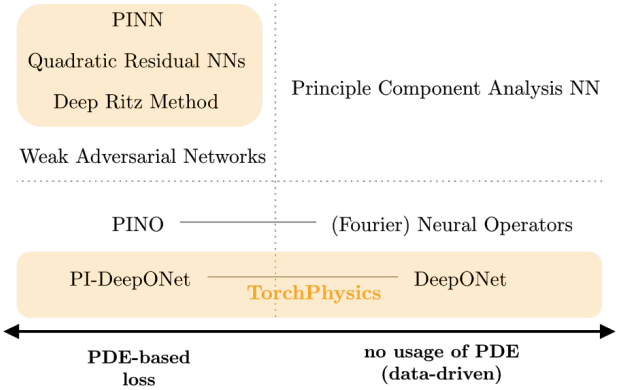
Compared to classical methods

- 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

Overview of DL-Methods for Differential Equations



Overview of DL-Methods for Differential Equations



TorchPhysics

Initiation of TorchPhysics

- 2021: Robert Bosch GmbH got interested in PINNs
- Technical applications:
  Car parts, electronics, etc.
- Current application: Injection Molding (Tomorrow)

BOSCH

© factum

Initiation of TorchPhysics

- 2021: Robert Bosch GmbH got interested in PINNs
- Technical applications:
  Car parts, electronics, etc.
- Current application: Injection Molding (Tomorrow)

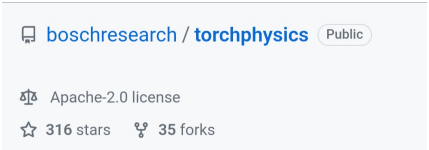
- Student project: Deep Learning library for PDEs
- Main Developers: Nick Heilenkötter & Tom Freudenberg

BOSCH

© factum

The Toolbox TORCHPHYSICS

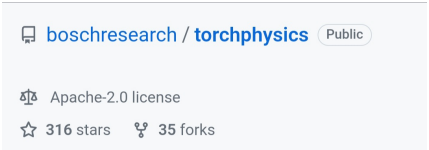
- Open-Source on GitHub
- Build upon PYTORCH[1]



[1] Paszke et al., *PyTorch: An Imperative Style, High-Performance Deep Learning Library*, 2019

The Toolbox TORCHPHYSICS

- Open-Source on GitHub
- Build upon PYTORCH[1]

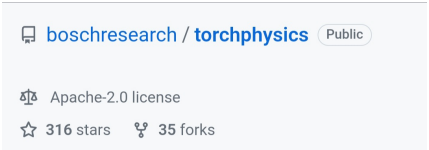


- **Goal:** User-friendly for everyone with basic mathematical knowledge

[1] Paszke et al., *PyTorch: An Imperative Style, High-Performance Deep Learning Library*, 2019

The Toolbox TORCHPHYSICS

- Open-Source on GitHub
- Build upon PYTORCH[1]



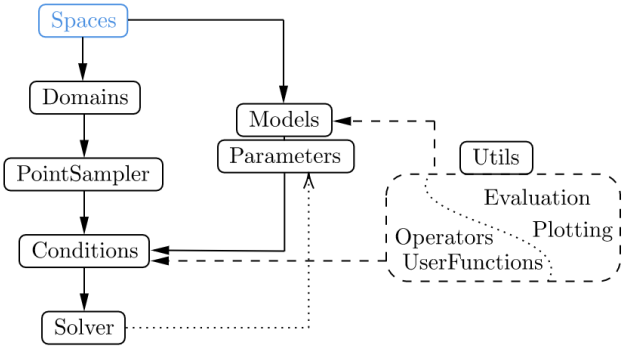
- **Goal:** User-friendly for everyone with basic mathematical knowledge
- TORCHPHYSICS provides:
  - Clean documentation, detailed tutorials
  - Modular and extendable structure
  - Intuitive transfer of maths into code

[1] Paszke et al., *PyTorch: An Imperative Style, High-Performance Deep Learning Library*, 2019

TorchPhysics - Structure

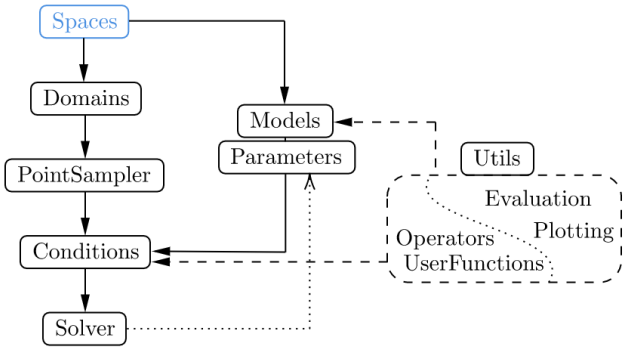
TorchPhysics

Spaces



TorchPhysics

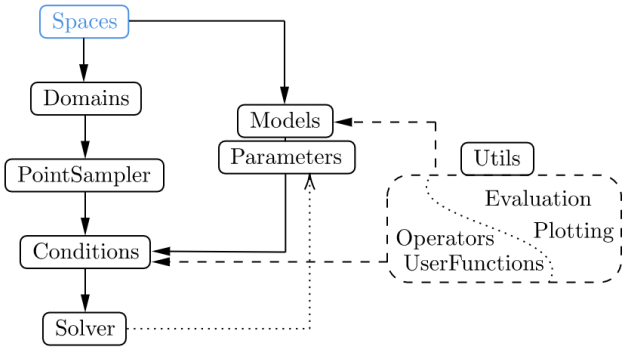
Spaces



Example: $\Omega = [0,1] \times [0,1]$

$$\begin{aligned} \Delta u(x) &= f(x), && \text{for } x \in \Omega, \\ u(x) &= u_0, && \text{for } x \in \partial\Omega. \end{aligned}$$

TorchPhysics
Spaces



Example: $\Omega = [0,1] \times [0,1]$

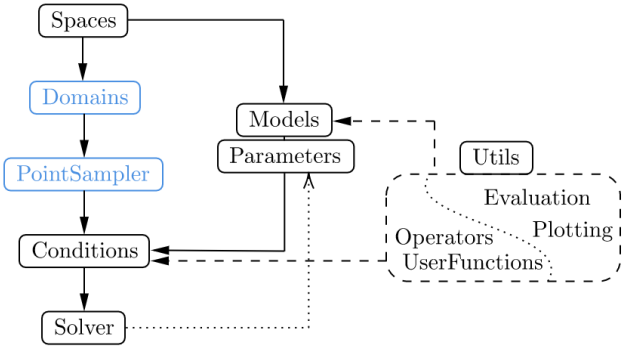
$$\Delta u(x) = f(x), \quad \text{for } x \in \Omega,$$
$$u(x) = u_0, \quad \text{for } x \in \partial\Omega.$$

TorchPhysics Spaces:

- Names for in-/output variables x, u
- Order of function inputs does not matter!

TorchPhysics

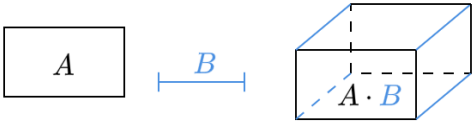
Domains



Domains

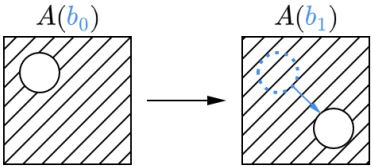
- Basic geometries implemented:
  - `Point, Interval, Parallelogram, Circle, ...`
- Complex domains via logical operators:

- Higher dimensional domains via Cartesian product:



Domains

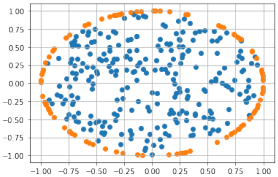
- Domain $A(b)$ may depend on point b from domain B (e.g. time-dependent)

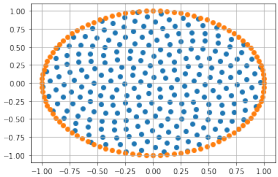


- Property `.boundary` returns the boundary as a new `Domain` object

PointSampler

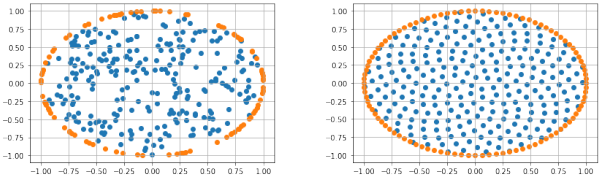
- Creation of training/validation points inside of the domains
- Different types of sampling:
  - `RandomUniformSampler, GridSampler, GaussianSampler, AdaptiveRejectionSampler, ...`





PointSampler

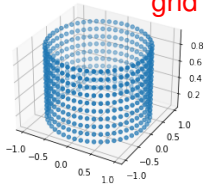
- Creation of training/validation points inside of the domains
- Different types of sampling:
  - `RandomUniformSampler, GridSampler, GaussianSampler, AdaptiveRejectionSampler, ...`
- Can be combined as desired



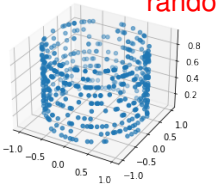
PointSampler

Flexible domain sampling

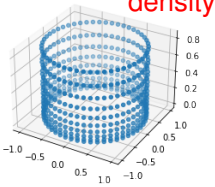
grid



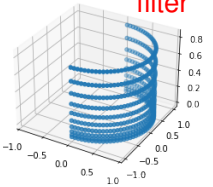
random

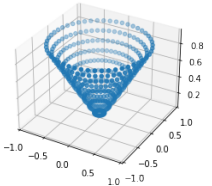


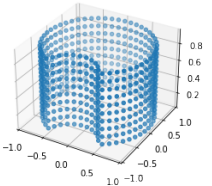
density

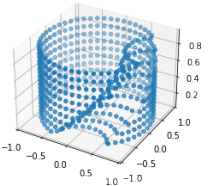


filter





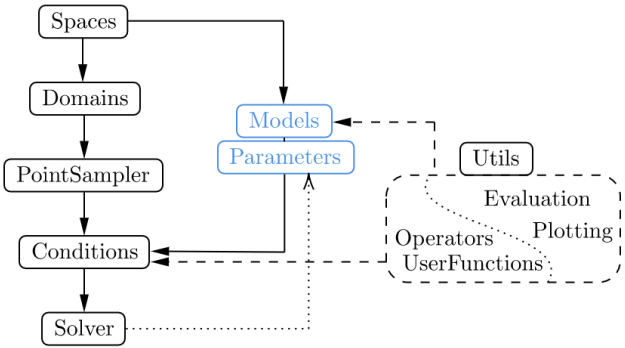




time-dependent domain

TorchPhysics

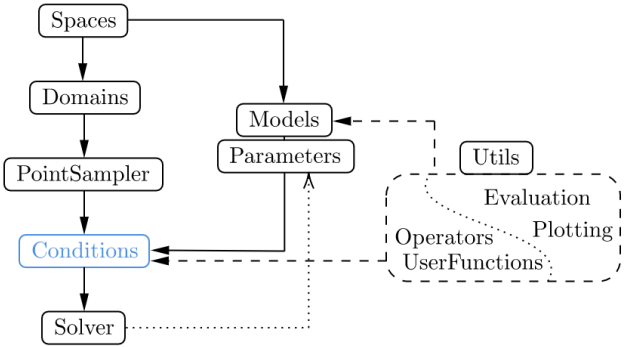
Neural Networks



- Load any PyTorch model
- Pre-implemented architectures: FCN, ResNet, DeepONet,...
- Trainable parameters for parameter identification

TorchPhysics

Conditions



Conditions

Core of TorchPhysics

- Different types, e.g. PINNCondition
- Represents one mathematical condition, e.g.

$$\Delta u(x) = f(x) \text{ in } \Omega \qquad \text{or} \qquad u(x) = u_0 \text{ at } \partial\Omega$$

Conditions

Core of TorchPhysics

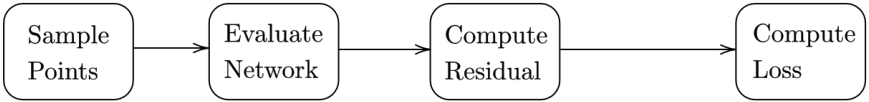
- Different types, e.g. PINNCondition
- Represents one mathematical condition, e.g.

$$\Delta u(x) = f(x) \text{ in } \Omega \qquad \text{or} \qquad u(x) = u_0 \text{ at } \partial\Omega$$

- Pre-implemented `DifferentialOperators` available, e.g. Laplacian Δ

Conditions

Workflow



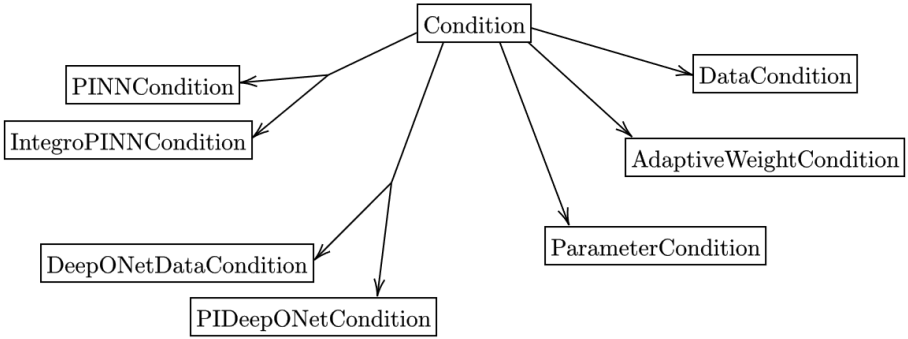
PINNCondition

$x_i \in \Omega$ $\quad u_\theta(x_i)$ $\quad \Delta u_\theta(x_i) - f(x_i)$ $\quad \sum_i |\Delta u_\theta(x_i) - f(x_i)|^2$

DataCondition

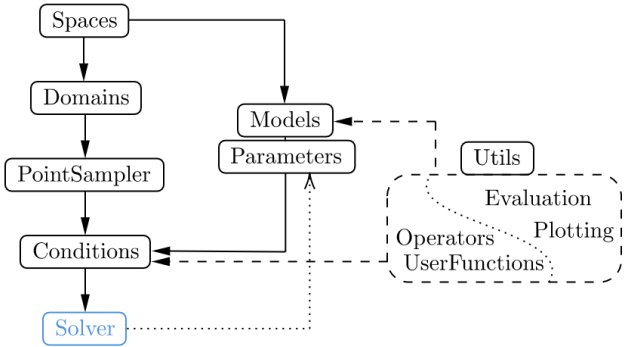
$(x_i, u_i) \in \Omega \times \mathbb{R}$ $\quad u_\theta(x_i)$ $\quad u_\theta(x_i) - u_i$ $\quad \sum_i |u_\theta(x_i) - u_i|^2$

Some More Conditions



TorchPhysics

Solver

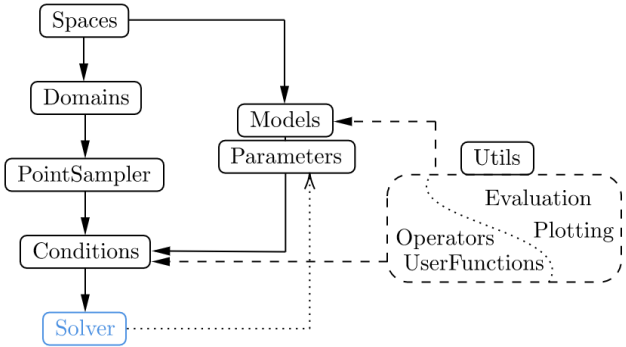


- Collects all conditions $\rightarrow$ overall loss
- Flexible choice optimization algorithm
- PyTorch Lightning[2] e.g. monitor individual conditions

[2] Falcon et al., *PyTorch Lightning*, 2019

TorchPhysics

Solver

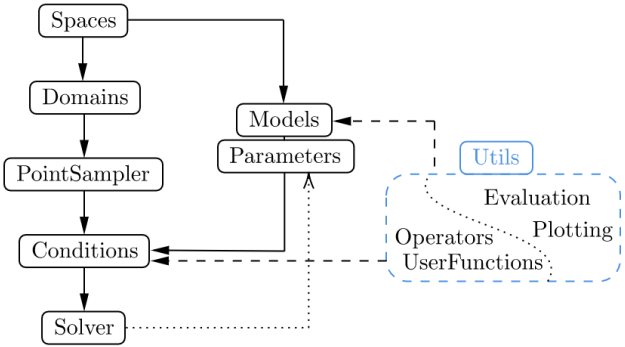


- Collects all conditions $\rightarrow$ overall loss
- Flexible choice optimization algorithm
- PyTorch Lightning[2] e.g. monitor individual conditions

[2] Falcon et al., *PyTorch Lightning*, 2019

TorchPhysics

Utils

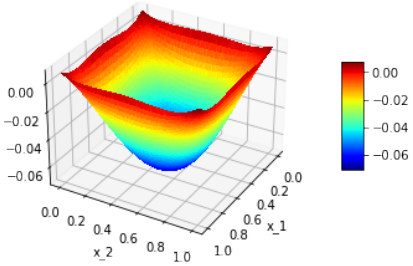


- Provides differential operators Δ, ∇, div,...
- Simplifies visualization

Examples

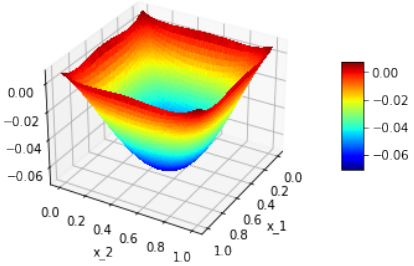
Poisson Equation with PINNs

$$\Delta u(x) = f(x) := 1, \quad \text{for } x \in \Omega$$
$$u(x) = u_0 := 0, \quad \text{for } x \in \partial\Omega$$



Poisson Equation with PINNs

$$\Delta u(x) = f(x) := 1, \quad \text{for } x \in \Omega$$
$$u(x) = u_0 := 0, \quad \text{for } x \in \partial\Omega$$

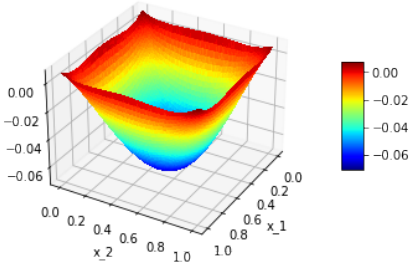


Only $\approx$ 25 lines of code!

- Imports
- Formulation of PDE
- Training
- Visualization

Poisson Equation with PINNs

$$\Delta u(x) = f(x) := 1, \quad \text{for } x \in \Omega$$
$$u(x) = u_0 := 0, \quad \text{for } x \in \partial\Omega$$



Only ≈ 25 lines of code!

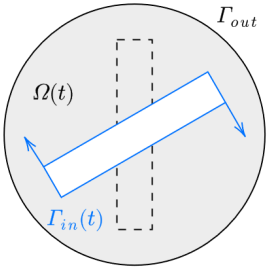
- Imports
- Formulation of PDE
- Training
- Visualization

→ **hands-on session later**

Time-Dependent Domain

PINNs for Navier-Stokes Example[3]

Bar heats up and rotates within some fluid:



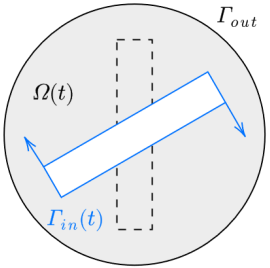
[3] Nganyu et al., *Deep Learning Methods for Partial Differential Equations [...]*, 2023

Time-Dependent Domain

PINNs for Navier-Stokes Example[3]

Bar heats up and rotates within some fluid:

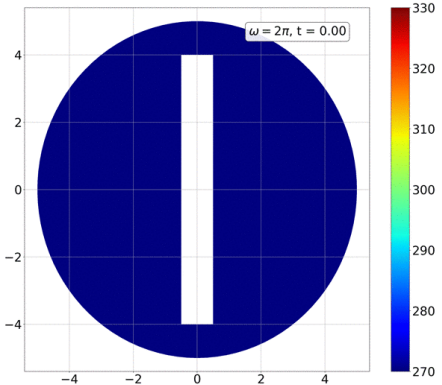
$$
\begin{aligned}
\partial_t u + (u \cdot \nabla) u &= \nu \Delta u - \nabla p, && \text{in } \Omega(t),\\
\nabla \cdot u &= 0, && \text{in } \Omega(t),\\
\partial_t T + u \cdot \nabla T &= \lambda \Delta T, && \text{in } \Omega(t),\\
u(0,\cdot), p(0,\cdot), T(0,\cdot) &= 0, 0, 270 && \text{in } \Omega(0),\\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



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

Time-Dependent Domain

Learned Temperature Field



Comparison to Other Python Libraries

Other Open-Source Python Libraries

...implementing physics-informed learning

1) DEEPXDE[4]
   - Developed by L. Lu, supervised by G. Karniadakis, Brown University, USA
   - Available on GitHub

[4] Lu et al., *DeepXDE: A Deep Learning Library for Solving Differential Equations*, 2021

Other Open-Source Python Libraries

...implementing physics-informed learning

1) DEEPXDE[4]
   - Developed by L. Lu, supervised by G. Karniadakis, Brown University, USA
   - Available on GitHub

2) NVIDIA MODULUS
   - ©2021-2022, NVIDIA Corporation
   - Available on GitHub

[4] Lu et al., *DeepXDE: A Deep Learning Library for Solving Differential Equations*, 2021

Comparison

General information

- TorchPhysics, Modulus based on PyTorch
- DeepXDE mainly on TensorFlow, also supports PyTorch, JAX, PaddlePaddle
- pip-installable
- Share similar structure/building blocks, like Domain, Conditions, etc. (different names)

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

Comparison

Pre-Implemented Methods

	TorchPhysics	DeepXDE	Modulus
PINN	✓	✓ Extensions	✓
DeepRitz	✓	Manually	Manually
(PI)DeepONet	✓	✓ Extensions	✓
FNO	✗	Fourier-DeepONet	✓
PINO	✗	✗	✓

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
```