Universität Bremen

Center for Industrial Mathematics (ZeTeM)

Operator Learning

Track 2

Janek Gödeke, Nick Heilenkötter, Tom Freudenberg

Renningen, 20.11.2025

So far: PINNs for Single PDE

Poisson equation:

$$\begin{aligned} \Delta u(x) &= f(x) && \text{for } x \in \Omega, \\ u(x) &= 0 && \text{for } x \in \partial\Omega. \end{aligned}$$

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

So far: PINNs for Single PDE

Poisson equation:

$$\begin{aligned} \Delta u(x) &= f(x) \quad \text{for } x \in \Omega, \\ u(x) &= 0 \quad \text{for } x \in \partial\Omega. \end{aligned}$$

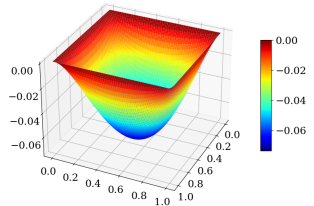


Fig.: Learned solution for $f = 1$

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

So far: PINNs for Single PDE

Poisson equation:

$$\begin{aligned} \Delta u(x) &= f(x) \quad && \text{for } x \in \Omega, \\ u(x) &= 0 && \text{for } x \in \partial\Omega. \end{aligned}$$

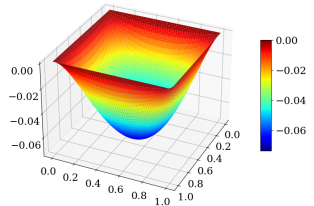


Fig.: Learned solution for $f = 1$

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

Neural network $u_\theta(x) \approx u(x)$

Drawback: New f —> retrain/new u_θ ...

So far: PINNs for Single PDE

Poisson equation:

$$\begin{aligned} \Delta u(x) &= f(x) \quad \text{for } x \in \Omega, \\ u(x) &= 0 \quad \text{for } x \in \partial\Omega. \end{aligned}$$

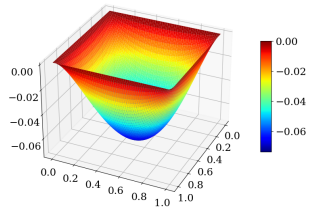


Fig.: Learned solution for $f = 1$

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

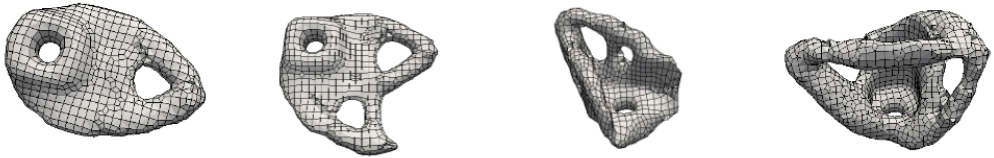
Neural network $u_\theta(x) \approx u(x)$

Drawback: New f —> retrain/new u_θ ...

Common applications: desirable to solve for u_θ for many different f

Application: Topology Optimization[1]

Setting



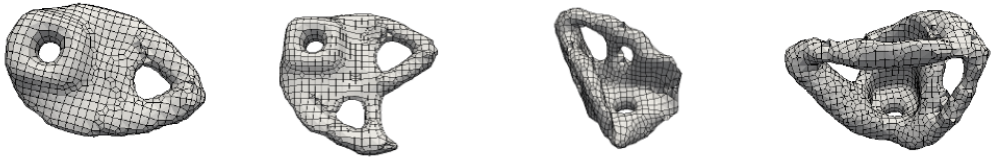
Goal: Finding lightweight but force-resilient geometries.

$$\begin{aligned} \min \quad & \text{compliance} \\ \text{s.t.} \quad & \text{deformation by given forces} \\ & \text{available volume} \end{aligned}$$

[1] Dittmer, Erzmann et al.: *SELTO: Sample-Efficient Learned Topology Optimization*

Application: Topology Optimization[1]

Setting



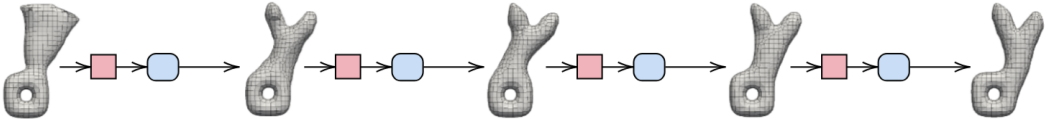
Goal: Finding lightweight but force-resilient geometries.

$$\begin{aligned} \min \quad & f_{\text{force}}^T u(f_{\text{density}}) \\ \text{s.t.} \quad & K(f_{\text{density}})u = f_{\text{force}} \\ & ||f_{\text{density}}||_1 \leq V \end{aligned}$$

[1] Dittmer, Erzmann et al.: *SELTO: Sample-Efficient Learned Topology Optimization*

Application: Topology Optimization

Iterative Optimization Process



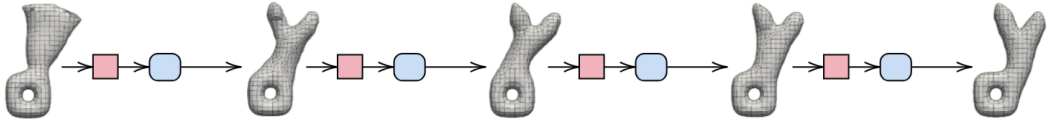
In each iteration:

Given f_{density}, f_{force}, solve PDE for u:

$$K(f_{\text{density}})u = f_{\text{force}}$$

Application: Topology Optimization

Iterative Optimization Process



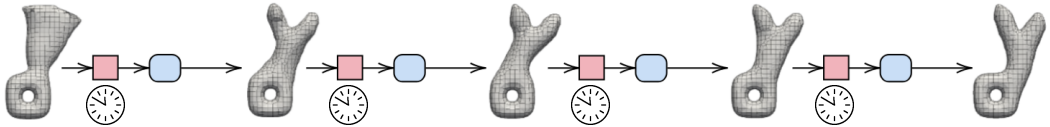
In each iteration:

Given $f_{\text{density}}, f_{\text{force}}$, solve PDE for u:

$$K(f_{\text{density}})u = f_{\text{force}}$$

Application: Topology Optimization

Iterative Optimization Process



In each iteration:

Given $f_{\text{density}}, f_{\text{force}}$, solve PDE for u:

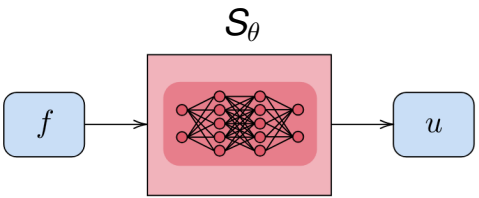
$$K(f_{\text{density}})u = f_{\text{force}}$$

Problem: PDE Solver via PINNs $\longrightarrow$ multiple trainings of a NN

Improving Time-Efficiency for Parameter Studies

Idea: A NN-design S_θ that is able to solve the PDE for diverse parameter functions f:

$$S_\theta(f) \approx u_f.$$

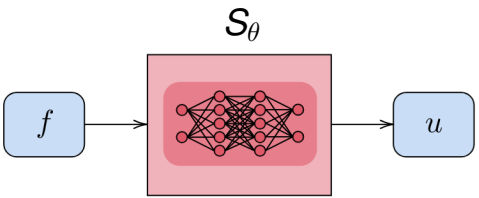


- **Train** S_θ on set of f_{train}
- **Hope:** S_θ also works for different f

Improving Time-Efficiency for Parameter Studies

Idea: A NN-design S_θ that is able to solve the PDE for diverse parameter functions f:

$$S_\theta(f) \approx u_f.$$



- **Train** S_θ on set of f_{train}
- **Hope:** S_θ also works for different f

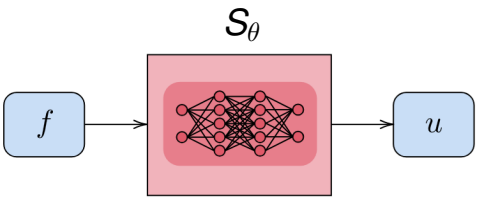
$\rightsquigarrow$ Trained S_θ replaces PDE solver

- Solving PDE $=$ Single forward pass

Improving Time-Efficiency for Parameter Studies

Idea: A NN-design S_θ that is able to solve the PDE for diverse parameter functions f:

$$S_\theta(f) \approx u_f.$$



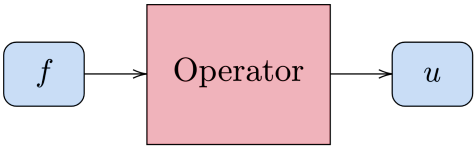
- **Train** S_θ on set of f_{train}
- **Hope:** S_θ also works for different f

$\rightsquigarrow$ Trained S_θ replaces PDE solver

- Solving PDE $=$ Single forward pass

Mathematical term for this approach:
"Operator Learning"

”Operator Learning”



Operator:

- Mapping *functions* to *functions*

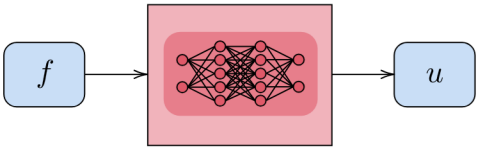
For PDEs:

- Mapping *parameter functions* to *solution functions*

Example: Poisson equation:

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

”Operator Learning”



Operator:

- Mapping *functions* to *functions*

For PDEs:

- Mapping *parameter functions* to *solution functions*

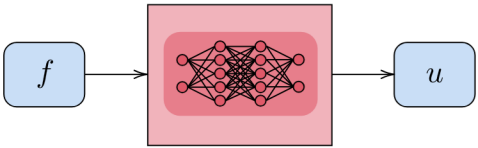
Example: Poisson equation:

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

NN $\approx$ Operator $\implies$ solution u for new f = single forward pass through NN

Operator Learning

Challenges

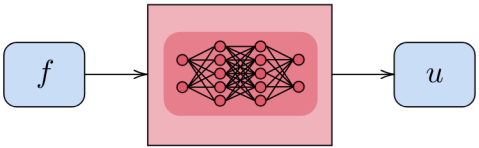


Architecture:

- How to plug in *functions* to a neural network?
- Generalization to unseen functions?

Operator Learning

Challenges



Architecture:

- How to plug in *functions* to a neural network?
- Generalization to unseen functions?

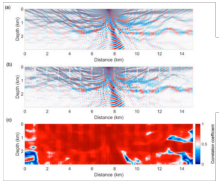
Training:

- Data-driven
- Physics-informed

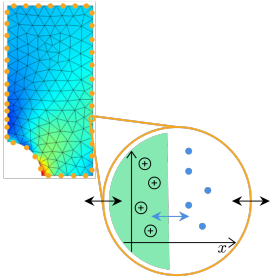
Operator Learning

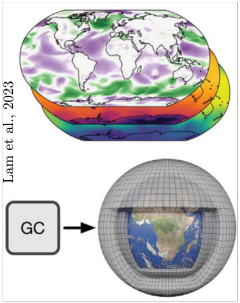
Use Cases

- Iterative algorithms:
  - Constrained optimization
  - Inverse Problems
  - ...
- Coupled micro-problems
- ...



Methaj et al., 2025

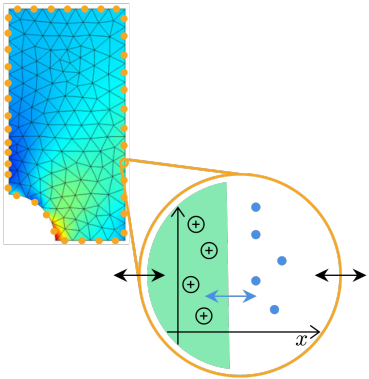




Lam et al., 2023

Application: Coupled Micro-Problems

Corrosion modeling

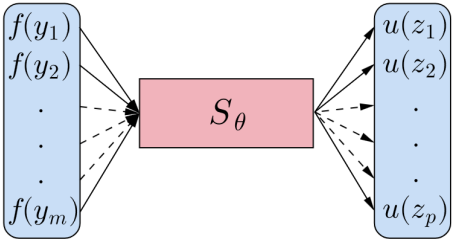


- In every FEM boundary cell: 1d system of electrochemical equations
- **Coupled** to macroscale simulation:
  - Input: $\underbrace{u(t_{i-1}), \text{boundary values, temperature, pH}}_{f}$
  - Output: fluxes (computed from solution $u(t_i)$)
- **Approach:** fast PCA-NN *learns the solution operator* and can be called in parallel on all cells

Getting Started: Naive Approach

Employ standard deep learning setting

1. **Discretize** functions f at fixed sampling points $y_1, ..., y_m$
2. **Discretize** functions u at fixed sampling points $z_1, ..., z_p$
3. Map discretized f to discretized u

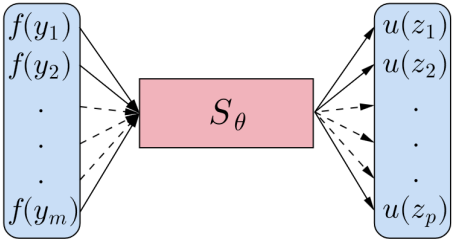


Getting Started: Naive Approach

Employ standard deep learning setting

1. **Discretize** functions f at fixed sampling points $y_1, ..., y_m$
2. **Discretize** functions u at fixed sampling points $z_1, ..., z_p$
3. Map discretized f to discretized u

How to train S_θ?



Training with Data - Workshop Focus

Set-up:

- Given f_k at fixed sampling points $y_1, ..., y_m$,
- Given u_k at fixed sampling points $z_1, ..., z_p$.

Poisson problem:

$$\begin{aligned} \Delta u(x) &= f(x) && \text{for } x \in \Omega, \\ u(x) &= 0 && \text{for } x \in \partial\Omega. \end{aligned}$$

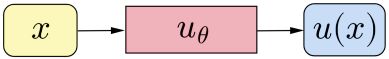
Data loss for K tuples (f_k, u_k):

$$\frac{1}{KL} \sum_{k=1}^{K} \sum_{l=1}^{p} \left| S_\theta \left(f_k(y_1), ..., f_k(y_m) \right)_l - u_k(z_l) \right|^2$$

Physics-Informed Training

PINNs u_θ: Trained with PDE-loss terms, e.g.,

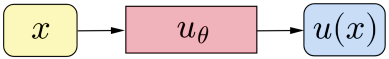
$$\frac{1}{N}\sum_{i=1}^{N}\left|\Delta u_\theta(x_i) - f(x_i)\right|^2.$$



Physics-Informed Training

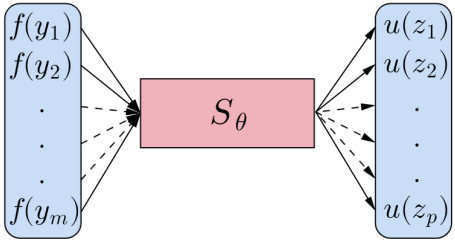
PINNs u_θ: Trained with PDE-loss terms, e.g.,

$$\frac{1}{N}\sum_{i=1}^{N}\left|\Delta u_\theta(x_i) - f(x_i)\right|^2.$$



Operator Learning S_θ:

- Finite differences of outputs $S_\theta(f)_i = u(z_i)$ (Famous method: PINO)



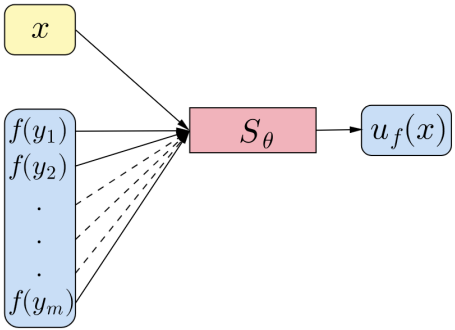
Physics-Informed Training

PINNs u_θ: Trained with PDE-loss terms, e.g.,

$$\frac{1}{N}\sum_{i=1}^{N}\left|\Delta u_\theta(x_i) - f(x_i)\right|^2.$$

Operator Learning S_θ:

- Finite differences of outputs $S_\theta(f)_i = u(z_i)$ (Famous method: PINO)
- Additional input $x \to$ similar to PINNs (Famous method: PI-DeepONet)



Hands on Session

Setting up the coding environment

Hands on: Operator Learning in TorchPhysics

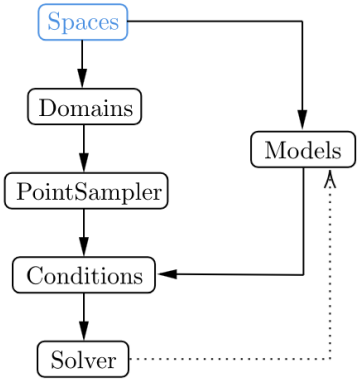
- We introduce operator learning with a simple ODE:

$$\begin{aligned} \partial_t u &= f \quad \text{in } (0,1), \\ u(0) &= 0. \end{aligned}$$

- **Goal**: find a neural network S_θ such that $S_\theta(f) = u$ for many f

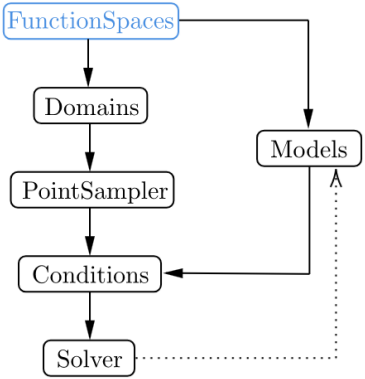
TorchPhysics

Spaces & FunctionSpaces



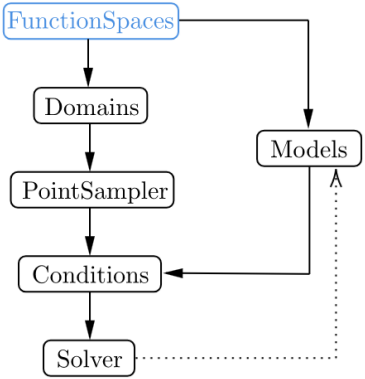
TorchPhysics

Spaces & FunctionSpaces



TorchPhysics

Spaces & FunctionSpaces

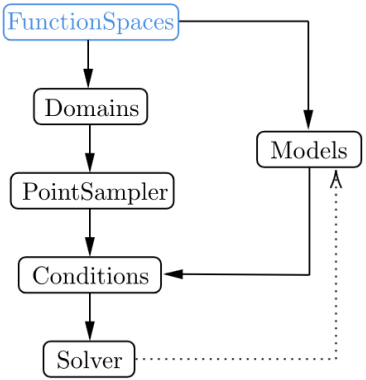


Example:

$$\partial_t u(t) = f(t) \qquad \text{for } t \in (0,1),$$
$$u(0) = 0$$

TorchPhysics

Spaces & FunctionSpaces



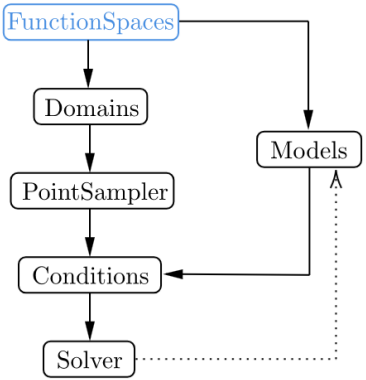
Example:

$$\partial_t u(t) = f(t) \qquad \text{for } t \in (0,1),$$
$$u(0) = 0.$$

- Functions with input t and output u
- Functions with input t and output f

TorchPhysics

Spaces & FunctionSpaces



Example:

$$\partial_t u(t) = f(t) \qquad \text{for } t \in (0,1),$$
$$u(0) = 0.$$

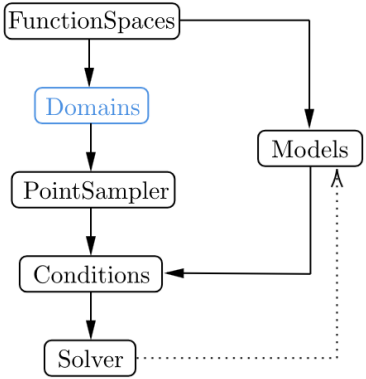
- Functions with input t and output u
- Functions with input t and output f

```
T = tp.spaces.R1("t")
F = tp.spaces.R1("f")
U = tp.spaces.R1("u")

fn_space_F = tp.spaces.FunctionSpace(T, F)
fn_space_U = tp.spaces.FunctionSpace(T, U)
```

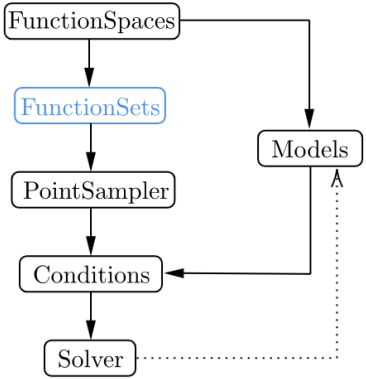
TorchPhysics

Domains & FunctionSets



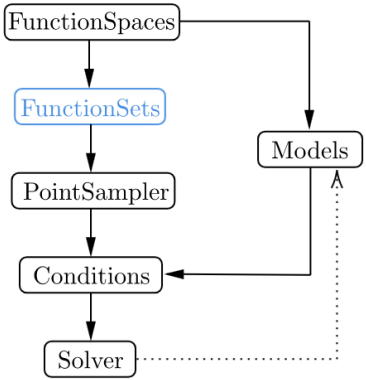
TorchPhysics

Domains & FunctionSets



TorchPhysics

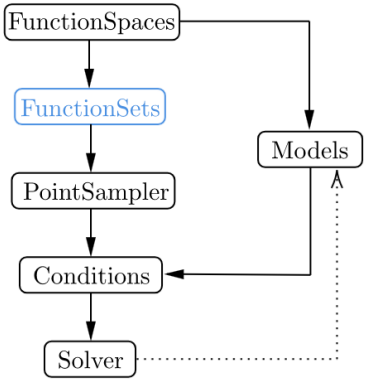
Domains & FunctionSets



- Data-driven approach: need to load and store data
- *FunctionSets*: connect data to *FunctionSpaces*

TorchPhysics

Domains & FunctionSets



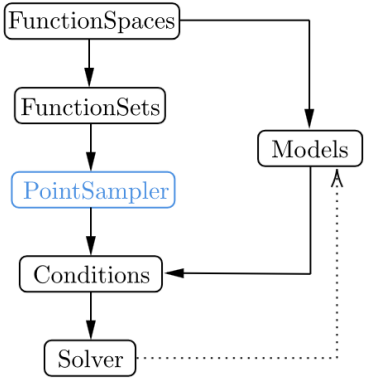
- Data-driven approach: need to load and store data
- *FunctionSets*: connect data to *FunctionSpaces*

```
data_functionset_input = \
    tp.domains.DataFunctionSet(fn_space_F, train_f)

data_functionset_output = \
    tp.domains.DataFunctionSet(fn_space_U, train_u)
```

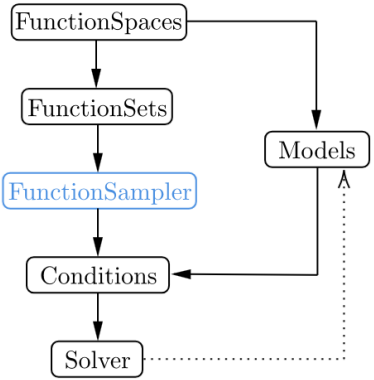
TorchPhysics

PointSampler & FunctionSampler



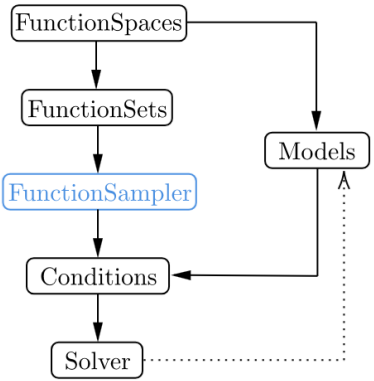
TorchPhysics

PointSampler & FunctionSampler



TorchPhysics

PointSampler & FunctionSampler



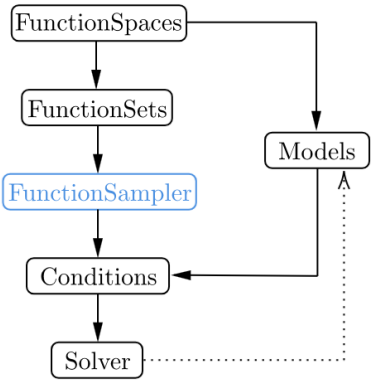
- Data-driven approach: many different combinations (f_i, u_i)

→ Too expensive to evaluate all at once

→ Sample (create batches) from the data

TorchPhysics

PointSampler & FunctionSampler



- Data-driven approach: many different combinations (f_i, u_i)

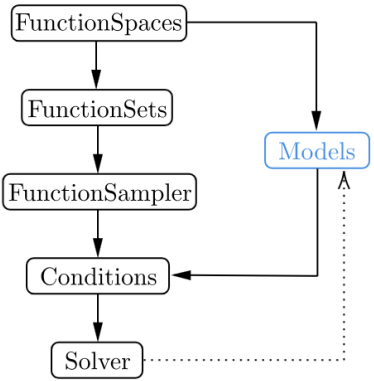
→ Too expensive to evaluate all at once

→ Sample (create batches) from the data

```
functionsampler_input = \
    tp.samplers.FunctionSamplerRandomUniform(
        6000, data_functionset_input
    )
functionsampler_output = \
    tp.samplers.FunctionSamplerCoupled(
        data_functionset_output, functionsampler_input
    )
```

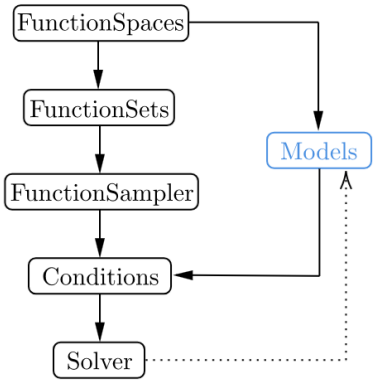
TorchPhysics

Neural Networks

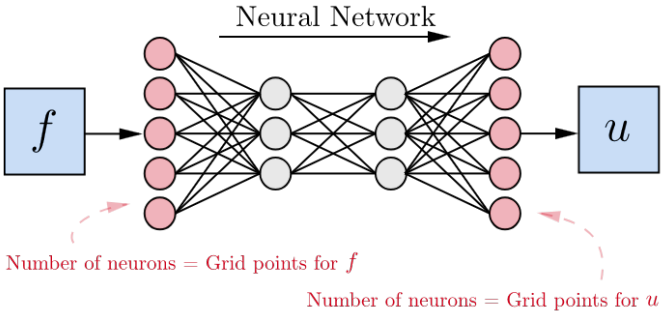


TorchPhysics

Neural Networks

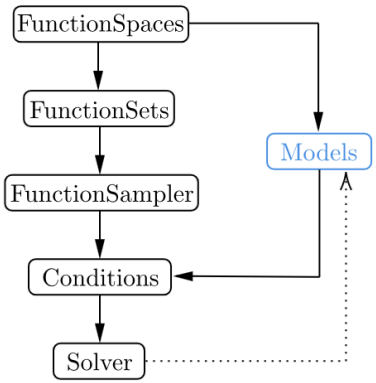


- Learn solution operator $S_\theta(f) = u$
- Here: f on grid $\rightarrow$ u on grid



TorchPhysics

Neural Networks

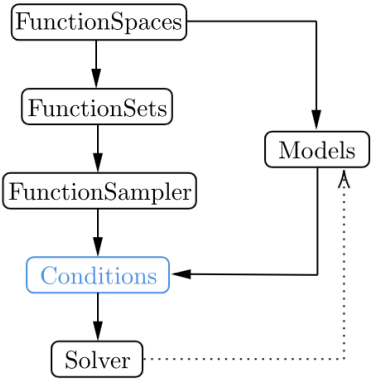


- Learn solution operator $S_\theta(f) = u$
- Here: f on grid $\rightarrow$ u on grid

```
model = tp.models.Operator_FCN(
            input_fn_space=fn_space_F,
            output_fn_space=fn_space_U,
            in_discretization=100,
            out_discretization=100,
            hidden=(50, 50)
        )
```

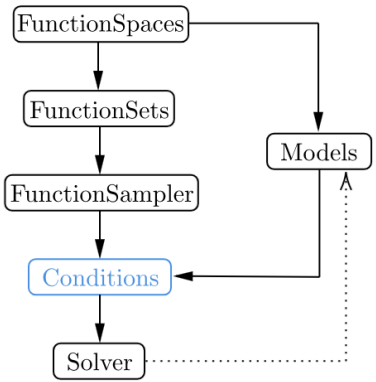
TorchPhysics

Conditions



TorchPhysics

Conditions



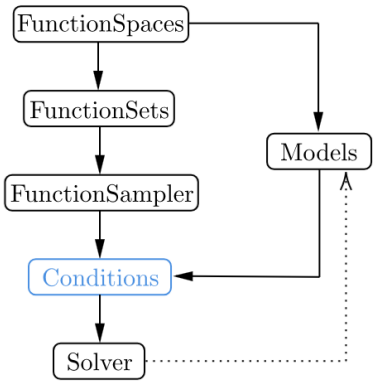
- Data-driven approach, **no** physics (yet)
- Minimize:

$$\frac{1}{N}\sum_{i=1}^{N} \|S_\theta(f_i) - u_i\|^2$$

- Default behavior of the `OperatorCondition`

TorchPhysics

Conditions



- Data-driven approach, **no** physics (yet)
- Minimize:

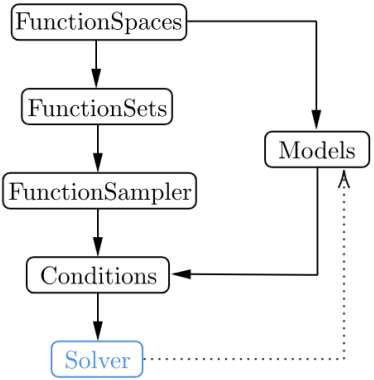
$$\frac{1}{N}\sum_{i=1}^{N} \|S_\theta(f_i) - u_i\|^2$$

- Default behavior of the `OperatorCondition`

```
data_condition = tp.conditions.OperatorCondition(
        module=model,
        input_function_sampler=functionsampler_input,
        output_function_sampler=functionsampler_output
        )
```

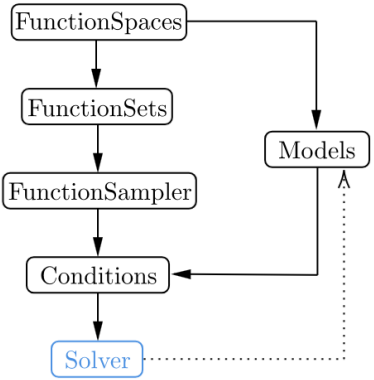
TorchPhysics

Solver



TorchPhysics

Solver



- Same syntax as before
- Now also include a *learning rate scheduler*
  - → Gradually reduces learning rate in training process

First exercises for operator learning

- Learning the solution operator of the **nonlinear Allen-Cahn equation**:

$$\begin{aligned} \epsilon \Delta u + (u^3 - u) &= f \quad \text{in } \Omega \\ \nabla u \cdot n &= 0 \quad \text{on } \partial\Omega \end{aligned}$$

Template: `Exercise_4.ipynb`

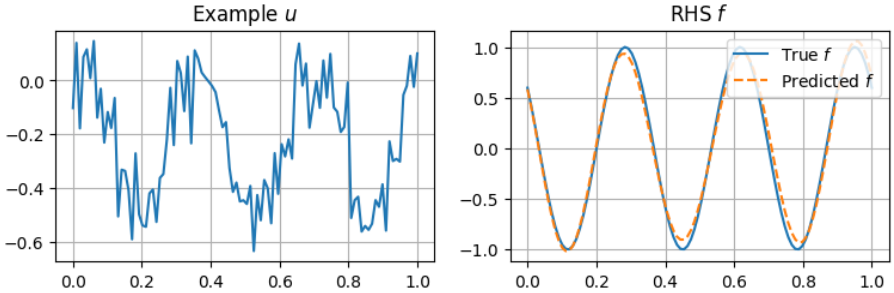
- **Bonus:** Learning the **inverse operator** $u \to f$ of the ODE:

$$\begin{aligned} \partial_t u &= f \quad \text{in } (0,1) \\ u(0) &= 0 \end{aligned}$$

Template: `Exercise_5.ipynb`

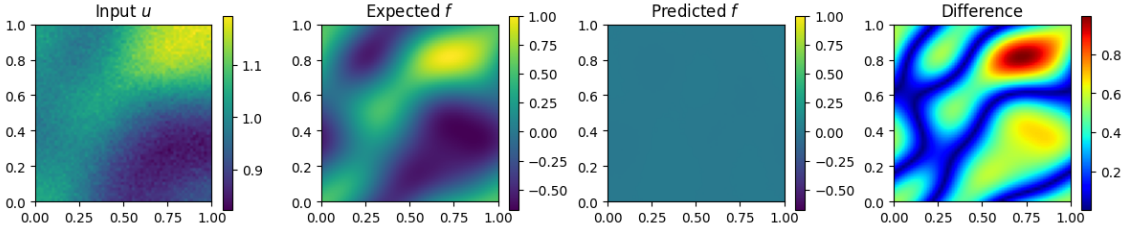
Naive Approach: Limitations

- Inverse mapping $u \mapsto f$ works in ODE case (even under **noise**)



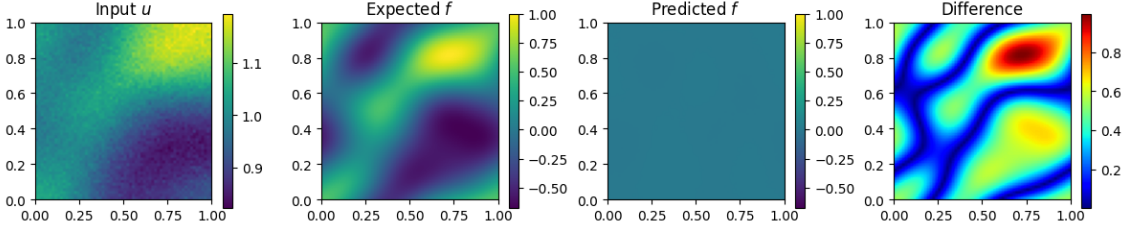
Naive Approach: Limitations

- Inverse mapping $u \mapsto f$ works in ODE case (even under **noise**)
- Naive inverse mapping $u \mapsto f$ **does not work** for **noisy** 2D Allen-Cahn equation!



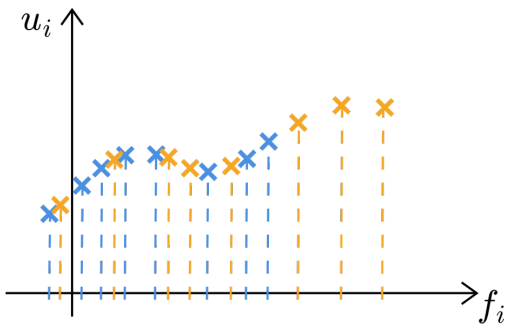
Naive Approach: Limitations

- Inverse mapping $u \mapsto f$ works in ODE case (even under **noise**)
- Naive inverse mapping $u \mapsto f$ **does not work** for **noisy** 2D Allen-Cahn equation!
- ***Tomorrow:*** *e.g. PCA-Nets*



Naive Approach: Limitations

we *train* on measured or sampled data,
use network on new data

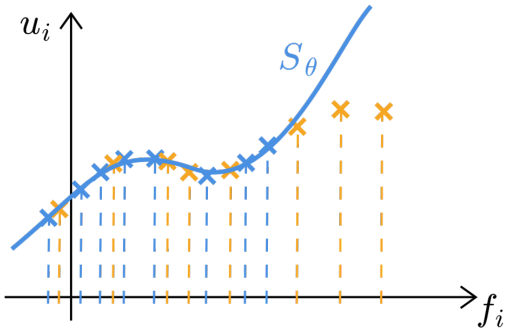


Naive Approach: Limitations

we *train* on measured or sampled data,
use network on new data

↓

Issue:
bad generalization to unseen functions



Naive Approach: Limitations

we *train* on measured or sampled data,
use network on new data

↓

Issue:
bad generalization to unseen functions

↓

Solution:
Enforce additional structure in architecture
(inductive bias)
Tomorrow: *e.g. FNO*