Universität Bremen

Center for Industrial Mathematics (ZeTeM)

Faculty 03

Mathematics / Computer science

Introduction to TorchPhysics

Working with DeepONets

Tom Freudenberg, Nick Heilenkötter, Janek Gödeke

Heidelberg, 08.11.2023

But first...

Setting up Google Colab

DeepONet Example in TorchPhysics

Goal: Learn for many $f : I_x \to \mathbb{R}$

Wave equation:

$$
\begin{aligned}
\partial_t^2 u &= c^2 \partial_x^2 u && \text{on } I_x \times I_t \subset \mathbb{R}^2,\\
u &= 0 && \text{on } \partial I_x \times I_t,\\
\partial_t u(\cdot, 0) &= 0 && \text{on } I_x,\\
u(x, 0) &= f(x) && \text{for } x \in I_x.
\end{aligned}
$$

DeepONet Example in TorchPhysics

Goal: Learn for many $f : I_x \to \mathbb{R}$

- Train on parameterized functions, e.g.

$$f_2(x, k_1, k_2, k_3) = \frac{1}{3}\sum_{n=1}^{3} k_n \sin(nx)$$

Wave equation:

$$\begin{aligned}
\partial_t^2 u &= c^2 \partial_x^2 u && \text{on } I_x \times I_t \subset \mathbb{R}^2,\\
u &= 0 && \text{on } \partial I_x \times I_t,\\
\partial_t u(\cdot, 0) &= 0 && \text{on } I_x,\\
u(x, 0) &= f(x) && \text{for } x \in I_x.
\end{aligned}$$

DeepONet Example in TorchPhysics

Goal: Learn for many $f : I_x \to \mathbb{R}$

- Train on parameterized functions, e.g.

$$f_2(x, k_1, k_2, k_3) = \frac{1}{3}\sum_{n=1}^{3} k_n \sin(nx)$$

- Paired with analytical solutions u_f

Wave equation:

$$\begin{aligned}
\partial_t^2 u &= c^2 \partial_x^2 u && \text{on } I_x \times I_t \subset \mathbb{R}^2, \\
u &= 0 && \text{on } \partial I_x \times I_t, \\
\partial_t u(\cdot, 0) &= 0 && \text{on } I_x, \\
u(x, 0) &= f(x) && \text{for } x \in I_x.
\end{aligned}$$

DeepONet Example in TorchPhysics

Goal: Learn for many $\color{red}{f}$ $: I_x \to \mathbb{R}$

- Train on parameterized functions, e.g.

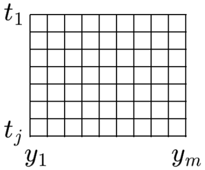
$$f_2(x, k_1, k_2, k_3) = \frac{1}{3}\sum_{n=1}^{3} k_n \sin(nx)$$

- Paired with analytical solutions u_f

- **Dataset:** f sampled at points $y_1, ..., y_m \in I_x$
  u sampled on grid

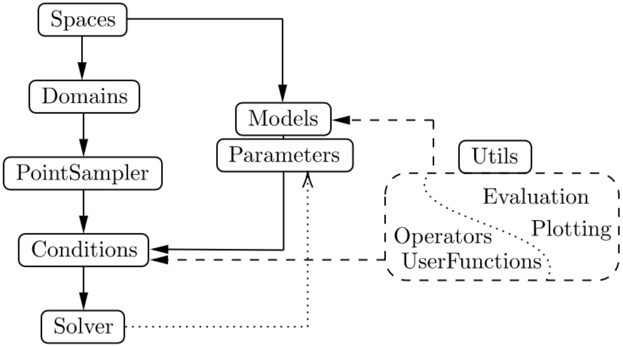
Wave equation:

$$\begin{aligned}
\partial_t^2 u &= c^2 \partial_x^2 u && \text{on } I_x \times I_t \subset \mathbb{R}^2,\\
u &= 0 && \text{on } \partial I_x \times I_t,\\
\partial_t u(\cdot, 0) &= 0 && \text{on } I_x,\\
u(x, 0) &= \color{red}{f}(x) && \text{for } x \in I_x.
\end{aligned}$$



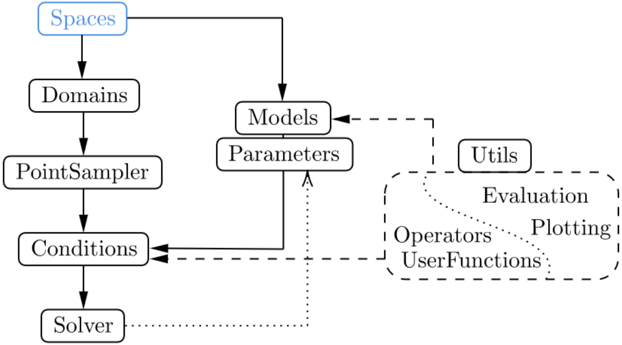
TorchPhysics

Structure



TorchPhysics

Spaces



As before: Spaces for

- Variables x, t
- Output u

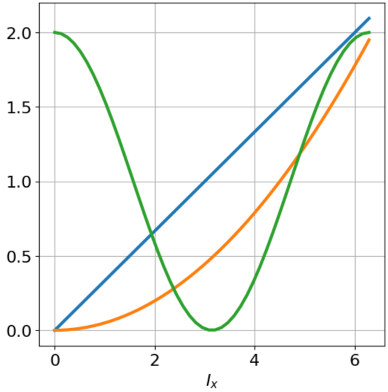
New: Space for f

Function Spaces

- Equivalent to the mathematical definition

$$\{f \mid f : I_x \to \mathbb{R}\}$$

- Function space for $f : I_x \to \mathbb{R}$ determined by
  - Domain I_x
  - Output space $\mathbb{R}$



Tom Freudenberg, Nick Heilenkötter, Janek Gödeke
Faculty 03 Mathematics / Computer science

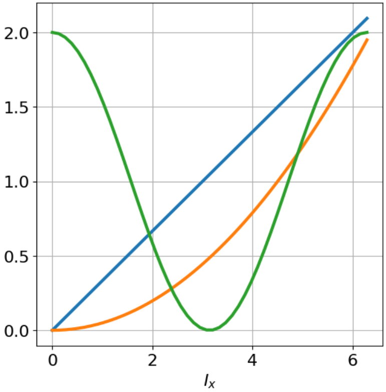
Function Spaces

- Equivalent to the mathematical definition

$$\{f \mid f : I_x \to \mathbb{R}\}$$

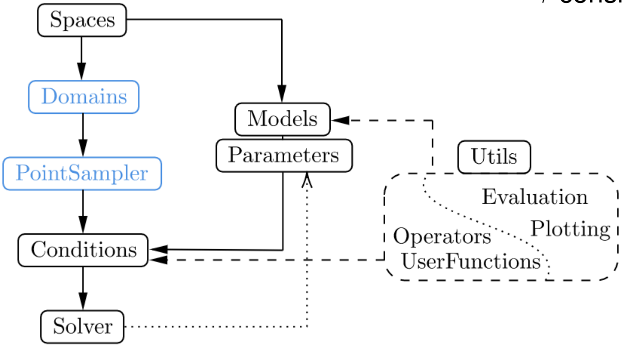
- Function space for $f : I_x \to \mathbb{R}$ determined by
  - Domain I_x
  - Output space $\mathbb{R}$

```
F        = tp.spaces.R1('f')

I_x      = tp.domains.Interval(X, x_min, x_max)

Fn_space = tp.spaces.FunctionSpace(I_x, F)
```



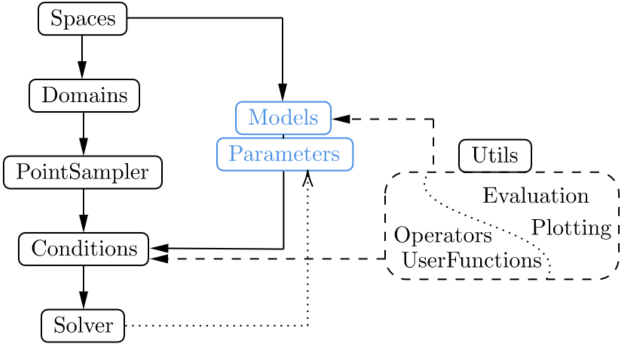
TorchPhysics
Domains

- Functionality only needed for PI-DeepONet
$\rightarrow$ considered later



TorchPhysics

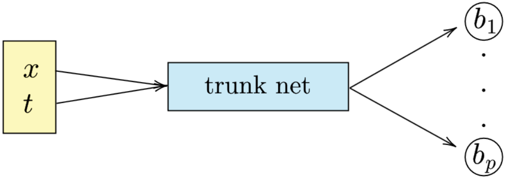
Neural Networks



Construct DeepONet

Trunk net:

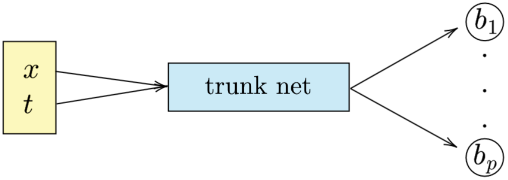
- Inputs similar to previous examples
- Arbitrary NN architecture



Construct DeepONet

Trunk net:

- Inputs similar to previous examples
- Arbitrary NN architecture

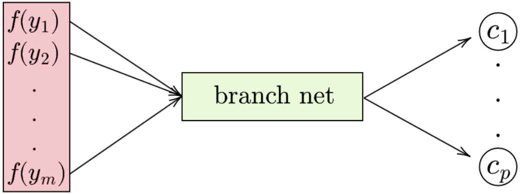



```
trunk_net   = tp.models.FCTrunkNet(input_space=T*X,
                                   hidden=(20, 30, 30, 30, 30, 40))
```

Construct DeepONet

Branch Net:

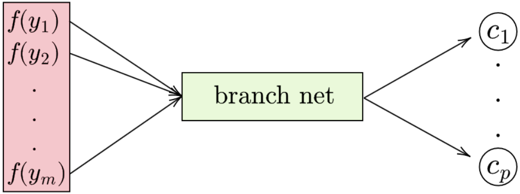
- Arbitrary NN architectures
- Branch inputs:
  - Discrete $(f(y_1), \dots, f(y_m))$
  - Continuous $f : I_x \to \mathbb{R}$
- Discretization sampler



Construct DeepONet

Branch Net:

- Arbitrary NN architectures
- Branch inputs:
  - Discrete $(f(y_1), \dots, f(y_m))$
  - Continuous $f : I_x \to \mathbb{R}$
- Discretization sampler

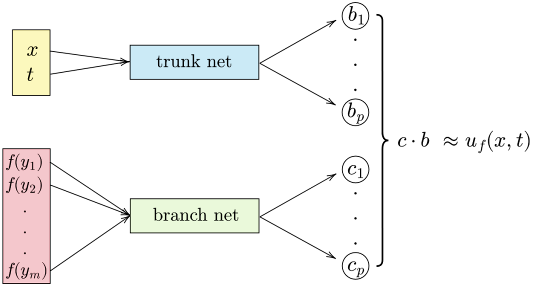



```
branch_net = tp.models.FCBranchNet(function_space=Fn_space,
                                   discretization_sampler=discretization_sampler,
                                   hidden=(50, 20, 20, 20, 50))
```

Construct DeepONet

DeepONet:

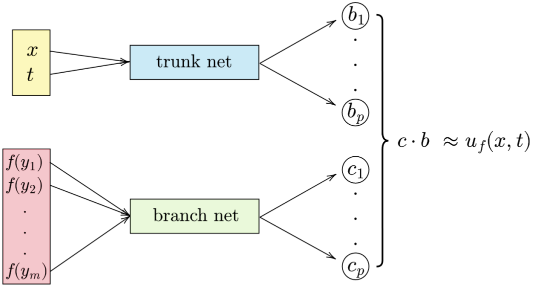
- Combines Trunk and Branch
- Set number of brunch/trunk outputs



Construct DeepONet

DeepONet:

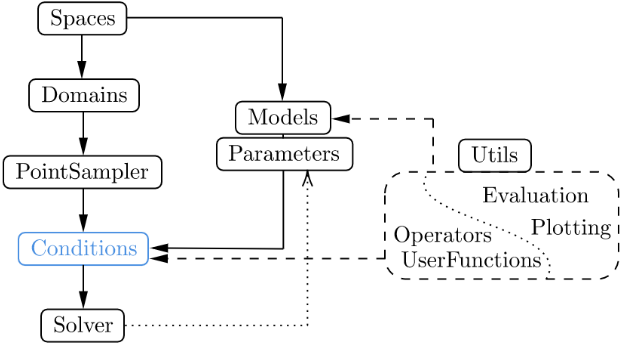
- Combines Trunk and Branch
- Set number of brunch/trunk outputs




```
deep_O_net = tp.models.DeepONet(trunk_net, branch_net,
                                output_space=U,
                                output_neurons=100)
```

TorchPhysics

Conditions



Conditions

- Different types, e.g `DeepONetDataCondition` or `PIDeepONetCondition`
- Still represents one mathematical equation, here

$$u_\theta\left(x_i, t_i, \begin{pmatrix} f(y_1) \\ \vdots \\ f(y_m) \end{pmatrix}\right) = u_f(x_i, t_i), \qquad u_\theta \text{ DeepONet}$$

Conditions

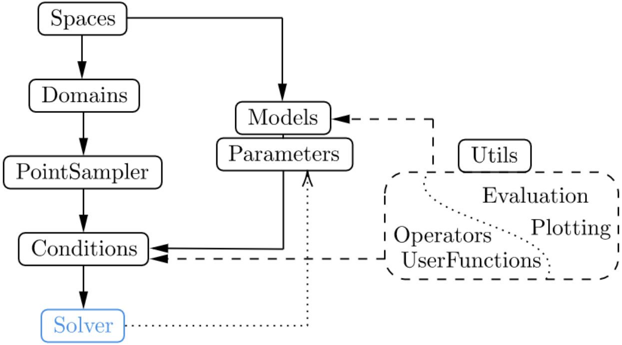
- Different types, e.g `DeepONetDataCondition` or `PIDeepONetCondition`
- Still represents one mathematical equation, here

$$u_\theta\left(x_i, t_i, \begin{pmatrix} f(y_1) \\ \vdots \\ f(y_m) \end{pmatrix}\right) = u_f(x_i, t_i), \qquad u_\theta \text{ DeepONet}$$

- `DeepONetDataCondition` implements the above condition given a corresponding dataloader

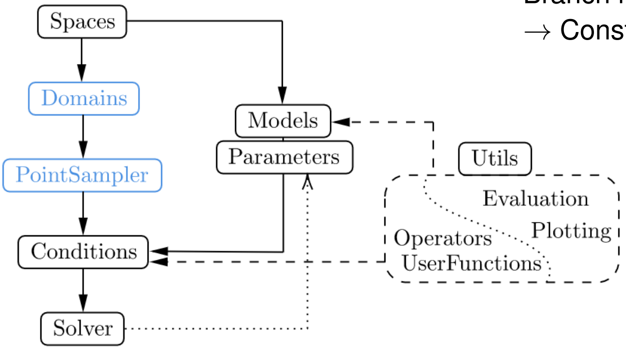
```
data_condition = tp.conditions.DeepONetDataCondition(module=deep_O_net,
                                                     dataloader=dataloader,
                                                     norm=2, root=2)
```

TorchPhysics
Solver



PI-DeepONet

TORCHPHYSICS
Function Sets



- Branch input not necessarily discrete data
  $\rightarrow$ Construct numerous different inputs

Tom Freudenberg, Nick Heilenkötter, Janek Gödeke

Faculty 03
Mathematics / Computer science

Function Sets

- Set of functions used as the Branch input
- Belongs to a given function space

Function Sets

- Set of functions used as the Branch input
- Belongs to a given function space

$$f_1(x; k) = kx$$

```
def f1(k, x):
    return k*x
```

Function Sets

- Set of functions used as the Branch input
- Belongs to a given function space

$$f_1(x; k) = kx$$

```python
def f1(k, x):
    return k*x

K_int         = tp.domains.Interval(K, 0, 6)
param_sampler = tp.samplers.RandomUniformSampler(K_int, n_points=80)
Fn_set_1      = tp.domains.CustomFunctionSet(Fn_space, param_sampler, f1)
```

Function Sets

- Set of functions used as the Branch input
- Belongs to a given function space

$$f_2(x;k) = k\cos(kx)$$

```
def f2(k, x):
    return k*cos(kx)

K_int         = tp.domains.Interval(K, 0, 6)
param_sampler = tp.samplers.RandomUniformSampler(K_int, n_points=80)
Fn_set_2      = tp.domains.CustomFunctionSet(Fn_space, param_sampler, f2)
```

Function Sets

- Set of functions used as the Branch input
- Belongs to a given function space

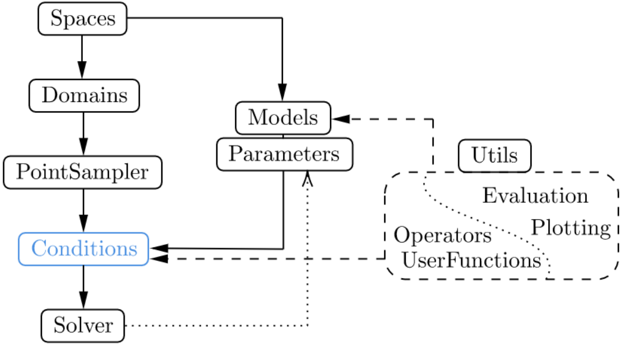
$$f_2(x; k) = k\cos(kx)$$

```
def f2(k, x):
    return k*cos(kx)

K_int         = tp.domains.Interval(K, 0, 6)
param_sampler = tp.samplers.RandomUniformSampler(K_int, n_points=80)
Fn_set_2      = tp.domains.CustomFunctionSet(Fn_space, param_sampler, f2)
```

- Multiple function sets can be combined (set union)

```
Fn_set = Fn_set_1 + Fn_set_2
```

TorchPhysics
Conditions



Conditions

- Seperate `PIDeepONetCondition` for mathematical equations, e.g

$$\partial_t^2 u = c^2 \partial_x^2 u \quad \text{or} \quad u(x,0) = f(x).$$

Conditions

- Seperate `PIDeepONetCondition` for mathematical equations, e.g

$$\partial_t^2 u = c^2 \partial_x^2 u \qquad \text{or} \qquad u(x,0) = f(x).$$

```python
def initial_residual(u, f):
    return u - f

initial_cond = tp.conditions.PIDeepONetCondition(deeponet_model = model,
                                                 function_set   = Fn_set,
                                                 input_sampler  = initial_sampler,
                                                 residual_fn    = initial_residual)
```

- In residual: f has already been evaluated at x

Exercises

Data driven (`Example_8.ipynb`)

- Add noise to dataset
- Study the influence of the dataset size on the generalization
- Implement different initial conditions

Physics informed (`Example_9.ipynb`)

- Implement a simple ODE problem
- Add different right hand sides to the training