Universität Bremen

Center for Industrial Mathematics (ZeTeM)

Faculty 03

Mathematics / Computer science

Deep Learning for Differential Equations and TorchPhysics

Workshop

Janek Gödeke, Tom Freudenberg

Bremen, 19.07.2023

Goals of This Workshop

- Data-based and physics-informed DL for ODEs/PDEs
- Learn usage of TorchPhysics library
- Learning solution functions of PDEs
  + parameter identification
- Learning solution operators - for multiple parameter functions

Today’s Programme

Lecture:

- Introduction to Differential Equations
- Learning solution functions from data or physics-informed
- Physics-Infomed Neural Networks (PINNs)

Exercises:

- Manual implementation, without TorchPhysics

Introduction to Differential Equations

Ordinary Differential Equations (ODEs)

- Consider functions of a **single** variable

$$
\begin{aligned}
u : \Omega \subseteq \mathbb{R} &\longrightarrow \mathbb{R}^m \\
t &\longmapsto \big(u_1(t), \ldots, u_m(t)\big)^T
\end{aligned}
$$

- Derivatives denoted by $u^{(n)}(t) = \big(u_1^{(n)}(t), \ldots, u_m^{(n)}(t)\big)^T$
- Alternative notation $u^{(1)} = u' = \partial_t u, \;\; u^{(2)} = u'' = \partial_t^2 u, \quad \ldots$

Ordinary Differential Equations (ODEs)

- Consider functions of a **single** variable

$$u : \Omega \subseteq \mathbb{R} \longrightarrow \mathbb{R}^m$$
$$t \longmapsto \big(u_1(t), ..., u_m(t)\big)^T$$

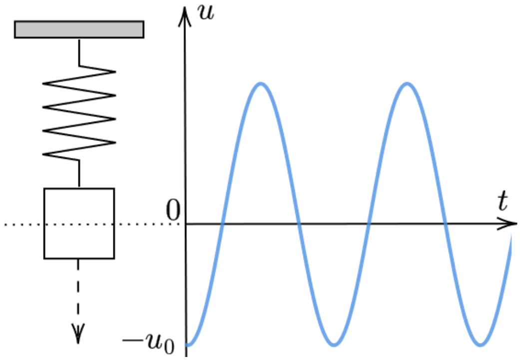
- Derivatives denoted by $u^{(n)}(t) = \big(u_1^{(n)}(t), ..., u_m^{(n)}(t)\big)^T$
- Alternative notation $u^{(1)} = u' = \partial_t u, \;\; u^{(2)} = u'' = \partial_t^2 u, \quad \ldots$
- **Goal:** Find u that solves the ODE

$$u^{(n)}(t) = f\left(t, u^{(1)}(t), u^{(2)}(t), ..., u^{(n-1)}(t)\right) \quad \text{for} \quad t \in \Omega$$

and satisfies some initial conditions, e.g. $u(t_0) = u_0$

ODE Example - Simple Harmonic Oscillator

- Consider mass on a spring
- Equilibrium position at $u = 0$ (no movement)
- Gets displaced at time $t = 0$ by u_0

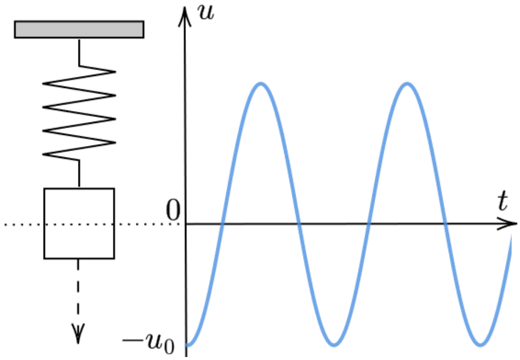


ODE Example - Simple Harmonic Oscillator

- Consider mass on a spring
- Equilibrium position at $u = 0$ (no movement)
- Gets displaced at time $t = 0$ by u_0
- Hooke's law:

$$u^{(2)}(t) = -ku(t)$$

- Velocity $u^{(1)}(0) = 0$
- Solution $u(t) = -u_0 \cos\left(\sqrt{k}t\right)$



Notation for Partial Differential Equations (PDEs)

- Consider functions of **multiple** variables

$$\begin{aligned} u : D \subseteq \mathbb{R}^l &\longrightarrow \mathbb{R}^m \\ x &\longmapsto \big(u_1(x), \ldots, u_m(x)\big)^T \end{aligned}$$

- Partial derivatives

kth standard basis vector in $\mathbb{R}^l$

$$\partial_{x_k} u(x) := \frac{\partial}{\partial x_k} u(x) := \lim_{h \to 0} \frac{u(x + h e_k) - u(x)}{h}$$

- Further conventions

$$\partial_{x_k}^2 u := \partial_{x_k}\left(\partial_{x_k} u\right) \neq \left(\partial_{x_k} u\right)^2$$

PDEs - Some Basic Differential Operators

- For scalar fields $u : D \subseteq \mathbb{R}^l \to \mathbb{R}$

Gradient $$\nabla u(x) = \left(\partial_{x_1} u(x), \dots, \partial_{x_l} u(x)\right)^T$$

Laplacian $$\Delta u(x) = \sum_k \partial_{x_k}^2 u(x)$$

- For vector fields $u : D \subseteq \mathbb{R}^l \to \mathbb{R}^m$

Divergence $$\operatorname{div} u(x) = \sum_k \partial_{x_k} u_k(x) = \nabla \cdot u$$

Rotation $$\operatorname{rot} u = \nabla \times u$$

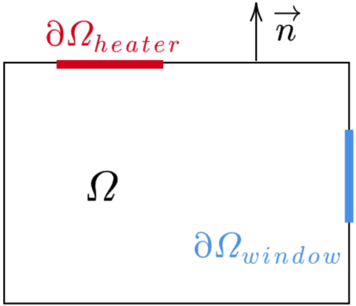
PDE Example - Stationary Heat Equation

- Room $\Omega \subset \mathbb{R}^2$ with window ($16^\circ C$) and heater ($40^\circ C$)

$$u = 16 \text{ at } \partial\Omega_{window}$$

$$u = 40 \text{ at } \partial\Omega_{heater}$$

(Dirichlet boundary conditions)



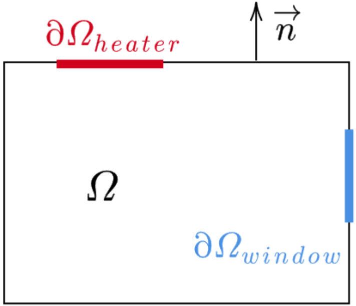
PDE Example - Stationary Heat Equation

- Room $\Omega \subset \mathbb{R}^2$ with window ($16^\circ C$) and heater ($40^\circ C$)

$$u = 16 \text{ at } \partial\Omega_{window}$$
$$u = 40 \text{ at } \partial\Omega_{heater}$$
(Dirichlet boundary conditions)

- Insulated walls $\partial\Omega_{wall} = \partial\Omega \setminus (\partial\Omega_{window} \cup \partial\Omega_{heater})$

$$\nabla u \cdot \vec{n} = 0 \text{ at } \partial\Omega_{wall}$$
(Neumann boundary condition)



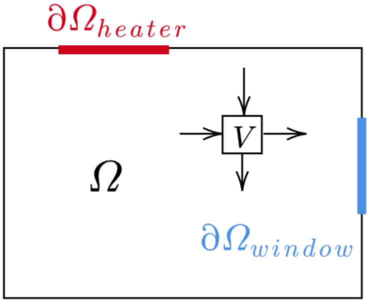
Normal vector $\vec{n}(x)$ is illustrated in the figure

PDE Example - Stationary Heat Equation

- Assumption: Temperature distribution reached static state $u : \Omega \to \mathbb{R}$:

$$0 = \int_{\partial V} \nabla u \cdot n \, dS$$
$$= \int_V \Delta u \, dV,$$

according to Gaussian Integration Theorem

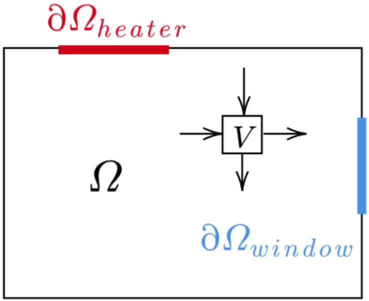


PDE Example - Stationary Heat Equation

- Assumption: Temperature distribution reached static state $u : \Omega \to \mathbb{R}$:

$$0 = \int_{\partial V} \nabla u \cdot n \, dS$$
$$= \int_V \Delta u \, dV,$$

according to Gaussian Integration Theorem



- Hence,

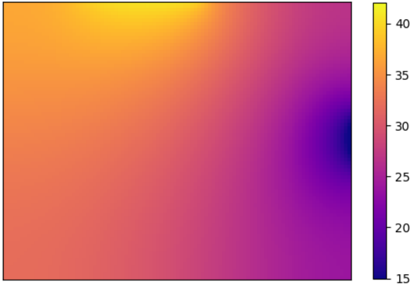
$$\Delta u = 0 \quad \text{on} \quad \Omega$$

PDE Example - Stationary Heat Equation

- Stationary Heat Equation with boundary conditions for the room:

$$\begin{cases} \Delta u & = 0 \text{ on } \Omega \\ u & = 16 \text{ at } \partial\Omega_{window} \\ u & = 40 \text{ at } \partial\Omega_{heater} \\ \nabla u \cdot \vec{n} & = 0 \text{ at } \partial\Omega_{wall} \quad \text{(insulated wall)} \end{cases}$$

- Solution approximated in TorchPhysics with PINNs



PDE Example - Time-Dependent Heat Equation

- Heater follows heat curve $h : [t_0, t_{end}] \to \mathbb{R}$

$$u(x,t) = h(t) \quad \text{for} \quad x \in \partial\Omega_{heater}, t \in [t_0, t_{end}]$$

- Time-dependent temperature distribution $u : \Omega \times [t_0, t_{end}] \to \mathbb{R}$
- Heat equation with constant diffusion $D \in \mathbb{R}$ becomes

$$\partial_t u = D \cdot \Delta_x u$$

- Will be considered in the exercise session tomorrow

Classical Numerical Methods for PDEs

- Finite Difference Methods (FDM)
  $\rightarrow$ classical formulation of PDE
- Finite Element Method (FEM)
  $\rightarrow$ weak formulation of PDE
- etc.

Finite Difference Method

Main Idea

- For $f : [0,1] \to \mathbb{R}$ find u

$$\begin{cases} u^{(1)}(x) &= f(x) \quad \text{on } (0,1) \\ u(0) &= 0 \end{cases}$$

- Taylor approximation at $x \in (0,1)$

$$u^{(1)}(x) \approx \frac{u(x+h) - u(x-h)}{2h}$$

Finite Difference Method

Main Idea

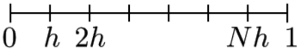
- For $f : [0,1] \to \mathbb{R}$ find u

$$\begin{cases} u^{(1)}(x) & = f(x) \quad \text{on } (0,1) \\ u(0) & = 0 \end{cases}$$

- Taylor approximation at $x \in (0,1)$

$$u^{(1)}(x) \approx \frac{u(x+h) - u(x-h)}{2h}$$

- Equidistant grid, stepsize $h = 1/(N+1)$



- Find $\big(u(0), u(h), \ldots, u(Nh), u(1)\big)^T$ by

$$\frac{1}{2h}\begin{pmatrix} 2h & & & & & \\ -1 & 0 & 1 & & & \\ & \ddots & \ddots & \ddots & & \\ & & & -1 & 0 & 1 \\ & & & & -2 & 2 \end{pmatrix} \begin{pmatrix} u(0) \\ u(h) \\ \vdots \\ u(Nh) \\ u(1) \end{pmatrix} = \begin{pmatrix} 0 \\ f(h) \\ \vdots \\ f(Nh) \\ f(1) \end{pmatrix}$$

Parameter Identification - Inverse Problem

- Have talked about finding solution of ODE/PDE for given parameters
  - Given diffusion coefficient $D \in \mathbb{R}$ solve

$$\partial_t u = D \Delta u$$

  - Given function $f : [0, 1] \to \mathbb{R}$ solve $u^{(1)} = f$

Parameter Identification - Inverse Problem

- Have talked about finding solution of ODE/PDE for given parameters
  - Given diffusion coefficient $D \in \mathbb{R}$ solve

$$\partial_t u = D \Delta u$$

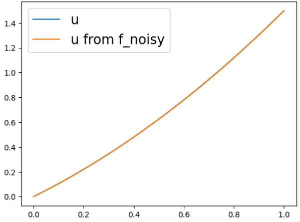
  - Given function $f : [0, 1] \to \mathbb{R}$ solve $u^{(1)} = f$
- Other-way-round: Given solution u, find parameter D or parameter function f
- Typically ill-posed $\to$ next slide

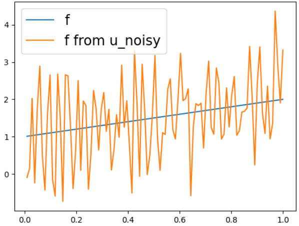
Parameter Identification - Ill-Posedness

- Parameter-solution pair
  $f(x) = x + 1$ and $u(x) = 0.5x^2 + x$ of

$$\begin{cases} u^{(1)}(x) = f(x) \\ u(0) = 0 \end{cases}$$

- Add noise (1.7%) to both of them
- Reconstruct u from f_{noisy}
  (solving ODE)
- Reconstruct f from u_{noisy} (parameter-identification)
  $\rightarrow$ Inverse integral operator





Deep Learning for PDEs

Motivation: Why Deep Learning for PDEs?

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

Deep Learning (DL) for PDEs

- Harmonic Oscillator
$$\begin{cases} u^{(2)}(t) = -ku(t) \\ u(0) = u_0,\ u^{(1)}(0) = 0 \end{cases}$$

- Forward problem: For different parameters $k > 0$ find solutions u_k
  1) Fix points $t_0 = 0, t_1, ..., t_N$ and learn (continuous) function
$$\begin{aligned} A_1 : \mathbb{R}_+ &\longrightarrow \mathbb{R}^{N+1} \\ k &\longmapsto \big(u_k(t_0), ..., u_k(t_N)\big)^T \end{aligned}$$
  2) **Mesh-independent:** Learn (continuous) function
$$\begin{aligned} A : \mathbb{R}_+ \times \mathbb{R} &\longrightarrow \mathbb{R} \\ (k, t) &\longrightarrow u_k(t) \end{aligned}$$

Deep Learning (DL) for PDEs

- Harmonic Oscillator
$$\begin{cases} u^{(2)}(t) = -ku(t) \\ u(0) = u_0,\ u^{(1)}(0) = 0 \end{cases}$$

- Forward problem: For different parameters $k > 0$ find solutions u_k
  1) Fix points $t_0 = 0, t_1, \dots, t_N$ and learn (continuous) function
$$\begin{aligned} A_1 : \mathbb{R}_+ &\longrightarrow \mathbb{R}^{N+1} \\ k &\longmapsto \left(u_k(t_0), \dots, u_k(t_N)\right)^T \end{aligned}$$

Inverse Problem:
$$A_1 k = \left(u_k(t_0), \dots, u_k(t_N)\right)^T$$

  2) **Mesh-independent:** Learn (continuous) function
$$\begin{aligned} A : \mathbb{R}_+ \times \mathbb{R} &\longrightarrow \mathbb{R} \\ (k, t) &\longrightarrow u_k(t) \end{aligned}$$

Why Do NNs Even Have a Chance?

Some Universal Approximation Theorem

Theorem (Hornik, 1989)

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

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

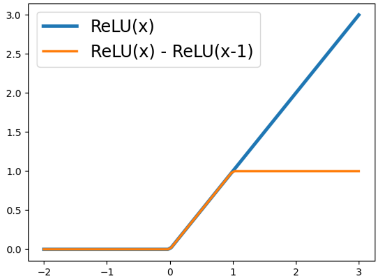
For each error ε there is φ_θ with p hidden layers and sigmoidal activations that uniformly approximates A, i.e.

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

- Sigmoidal $\sigma : \mathbb{R} \to \mathbb{R}$ means $\lim_{x\to-\infty} \sigma(x) = 0$ and $\lim_{x\to\infty} \sigma(x) = 1$

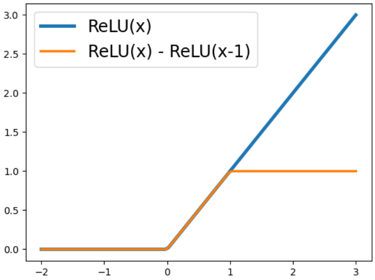
Many Universal Approximation Theorems...

- Also for *ReLU* activation, e.g.
  $ReLU(x) - ReLU(x-1)$ is sigmoidal

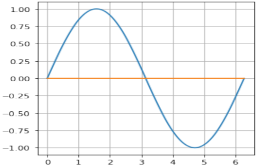


Many Universal Approximation Theorems...

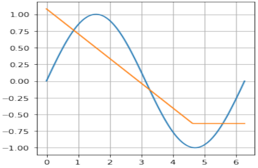
- Also for *ReLU* activation, e.g. $ReLU(x) - ReLU(x-1)$ is sigmoidal
- Theorems with explicit bounds on required network size, e.g. (Yarotsky 2017), (Barron 1993)
- Approximation in different norms, e.g. L^p-norms or Sobolev norms, e.g. (Petersen 2019)
- Overview e.g. (DeVore et al. 2020)



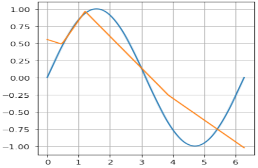
Example of Approximation Properties



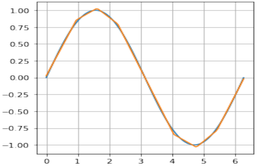
(a) 1 layer with 1 neuron



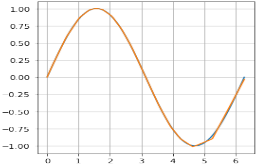
(b) 1 layer with 2 neurons



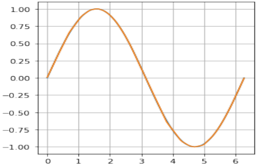
(c) 1 layer with 5 neurons



(d) 1 layer with 10 neurons



(e) 1 layer with 50 neurons

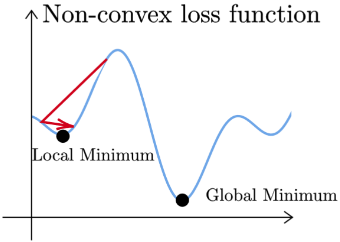


(f) 1 layer with 100 neurons

Nothing More to Do?

- Neural Networks typically non-convex
  $\rightarrow$ Loss minimization is difficult
  $\rightarrow$ Convergence guarantees to global minimum?

- Incomplete or noisy data
  $\rightarrow$ "Bad" network architecture gets side-tracked



DL for PDEs - Data-Driven Approach

- Harmonic Oscillator
$$\begin{cases} u^{(2)}(t) = -ku(t) \\ u(0) = u_0,\ u^{(1)}(0) = 0 \end{cases}$$

- Forward problem: For different parameters $k > 0$ find solutions u_k
  1) Fix points $t_0 = 0, t_1, \ldots, t_N$ and learn (continuous) function
$$\begin{aligned} A_1 : \mathbb{R}_+ &\longrightarrow \mathbb{R}^{N+1} \\ k &\longmapsto \big(u_k(t_0), \ldots, u_k(t_N)\big)^T \end{aligned}$$
  2) **Mesh-independent:** Learn (continuous) function
$$\begin{aligned} A : \mathbb{R}_+ \times \mathbb{R} &\longrightarrow \mathbb{R} \\ (k, t) &\longrightarrow u_k(t) \end{aligned}$$

DL for PDEs - Data-Driven Approach

1) Fix points $t_0 = 0, t_1, ..., t_N$ and learn

$$\begin{aligned} A_1 : \mathbb{R}_+ &\longrightarrow \mathbb{R}^{N+1} \\ k &\longmapsto \left(u_k(t_0), ..., u_k(t_N)\right)^T \end{aligned}$$

2) **Mesh-independent:** Learn

$$\begin{aligned} A : \mathbb{R}_+ \times \mathbb{R} &\longrightarrow \mathbb{R} \\ (k, t) &\longrightarrow u_k(t) \end{aligned}$$

DL for PDEs - Data-Driven Approach

1) Fix points $t_0 = 0, t_1, ..., t_N$ and learn

$$A_1 : \mathbb{R}_+ \longrightarrow \mathbb{R}^{N+1}$$
$$k \longmapsto \left(u_k(t_0), ..., u_k(t_N)\right)^T$$

- Data-tuples $k_j \in \mathbb{R}_+$ and $\tilde{u}_j \in \mathbb{R}^{N+1}$
- Train u_θ for minimizing e.g. MSE-loss

$$\frac{1}{|J|} \sum_{j \in J} \left\| u_\theta(k_j) - \tilde{u}_j \right\|^2$$

2) **Mesh-independent:** Learn

$$A : \mathbb{R}_+ \times \mathbb{R} \longrightarrow \mathbb{R}$$
$$(k, t) \longrightarrow u_k(t)$$

- Data $(k_j, t_j) \in \mathbb{R}_+ \times \mathbb{R}$ and $\tilde{u}_j \in \mathbb{R}$
- Train u_θ for minimizing

$$\frac{1}{|J|} \sum_{j \in J} \left| u_\theta((k_j, t_j)) - \tilde{u}_j \right|^2$$

Problems of Data-Driven Approaches for PDEs

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

  $\rightarrow$ Expensive and time consuming

Problems of Data-Driven Approaches for PDEs

- Deep Learning generally needs lot of data
- Obtaining data of solution u is complicated
  - Through multiple experiments
  - Solving the equation with classical methods
  - $\rightarrow$ Expensive and time consuming

Encode physical laws/PDEs into DL approaches?

$\rightarrow$ **Physics-informed neural networks (PINNs)**

$\rightarrow$ Plug neural network into the differential equation

Physics-Informed Neural Networks (PINNs)

PINNs - Original Publication

- **Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations**
- Authors: Raissi, Perdikaris, Karniadakis
- Journal of Computational Physics, 2019

PINNs - Main Idea

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

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

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

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

PINNs - Main Idea

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

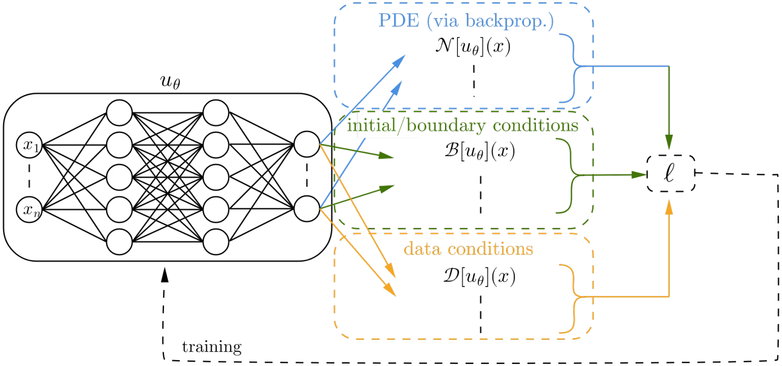
$$\mathcal{N}[u](x) = 0, \text{ for } x \in \Omega,$$
$$\mathcal{B}[u](x) = 0, \text{ for } x \in \partial\Omega.$$

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

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

PINN-Approach

Schematic depiction



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

Physics-Informed Loss - We need to ...

- Compute differential operator $\mathcal{N}$ of NN u_θ, e.g. Laplacian Δu_θ
  1) Autograd/Backpropagation of PyTorch
  2) Finite Differences

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

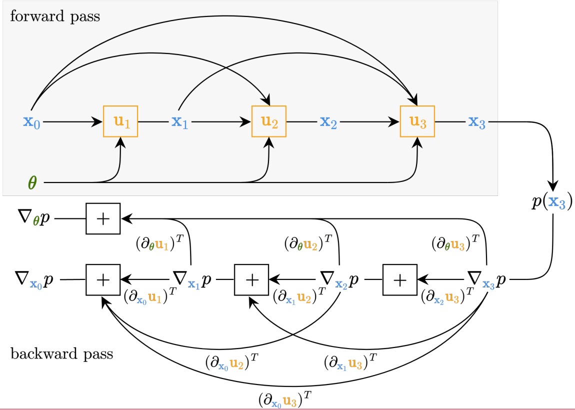
$$\text{PDE-loss:} \quad \frac{1}{N_\mathcal{N}} \sum_{i=1}^{N_\mathcal{N}} \left\| \mathcal{N}[u_\theta](x_i^\mathcal{N}) \right\|^2 + \frac{1}{N_\mathcal{B}} \sum_{j=1}^{N_\mathcal{B}} \left\| \mathcal{B}[u_\theta](x_j^\mathcal{B}) \right\|^2$$

Backpropagation

- Gradients of **scalar-valued** functions

$$p \circ u_\theta : \mathbb{R}^n \to \mathbb{R}^m \to \mathbb{R},$$

p is projection or loss function



Backpropagation

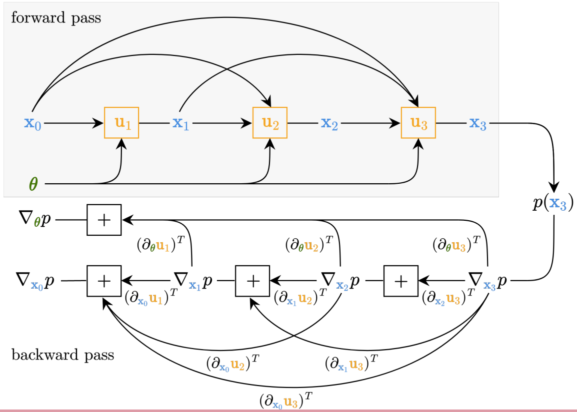
- Gradients of **scalar-valued** functions

$$p \circ u_\theta : \mathbb{R}^n \to \mathbb{R}^m \to \mathbb{R},$$

  p is projection or loss function

- Chain-rule for

$$\nabla_\theta (p \circ u_\theta)(x_0), \ \nabla_x (p \circ u_\theta)(x_0)$$



Backpropagation

- Gradients of **scalar-valued** functions

$$p \circ u_\theta : \mathbb{R}^n \to \mathbb{R}^m \to \mathbb{R},$$

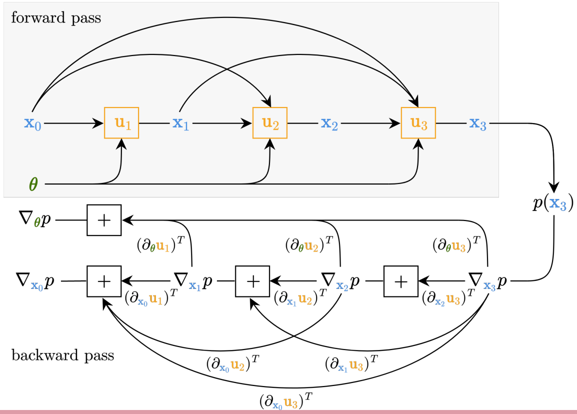
p is projection or loss function

- Chain-rule for

$$\nabla_\theta (p \circ u_\theta)(x_0), \;\; \nabla_x (p \circ u_\theta)(x_0)$$

- Gradient $\nabla_{x_0} (p \circ u_\theta)(x_0)$ obtained by

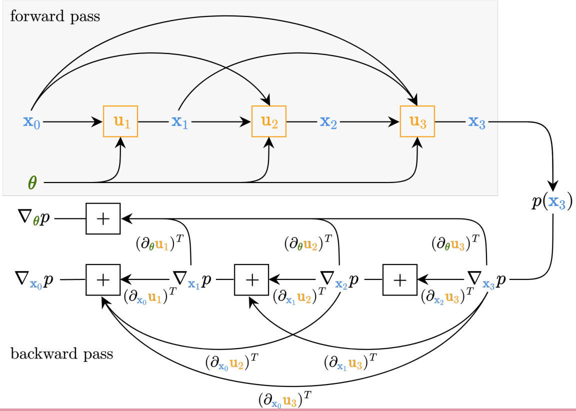
```python
import torch.autograd as auto
pu     = p(model(x_0))
grad_x = auto.grad(pu, x_0)[0]
```



Backpropagation - Higher Derivatives

- Gradient $\nabla_{x_0}(p \circ u_\theta)(x_0)$ obtained by

```python
import torch.autograd as auto
pu     = p(model(x_0))
grad_x = auto.grad(pu, x_0)[0]
```



Backpropagation - Higher Derivatives

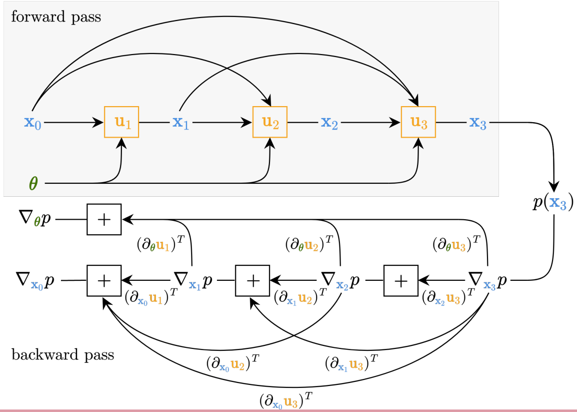
- Gradient $\nabla_{x_0}(p \circ u_\theta)(x_0)$ obtained by

```python
import torch.autograd as auto
pu     = p(model(x_0)
grad_x = auto.grad(pu, x_0)[0]
```

- Computational graph for gradient computation:

```python
grad_x = auto.grad(pu, x_0,
                   create_graph=True)[0]
```

- Allows for computation of higher derivatives



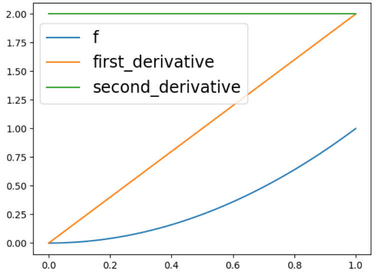
Example - Second Derivative of $f(t) = t^2$

```python
import torch.autograd as auto

t    = torch.linspace(0, 1, 100, requires_grad=True)
ft   = t**2

first_derivative  = auto.grad(ft.sum(), t,
                              create_graph=True)[0]

second_derivative = auto.grad(first_derivative.sum(), t,
                              create_graph=True)[0]
```



- Why is "ft.sum()" correct?

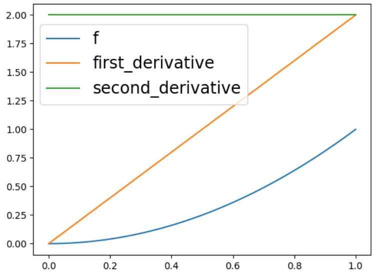
Example - Second Derivative of $f(t) = t^2$

```
import torch.autograd as auto

t   = torch.linspace(0, 1, 100, requires_grad=True)
ft  = t**2

first_derivative  = auto.grad(ft.sum(), t,
                              create_graph=True)[0]

second_derivative = auto.grad(first_derivative.sum(), t,
                              create_graph=True)[0]
```



- Second "create_graph=True"
  $\rightarrow$ Further derivatives computable, e.g.

$$\partial_\theta\left(\partial_t^2 f(t,\theta)\right) \quad \text{if } f \text{ was } \theta\text{-dependent}$$

Alternative Physics-Informed Loss

- Previous slides: PyTorch's backpropagation for **analytic** derivatives of NN
- Instead: Approximate derivatives, e.g. by finite differences
- Example: Again $u^{(1)} = f$. Approximate (part of) PDE-loss

$$|u_\theta^{(1)}(t) - f(t)|^2 \approx \left| \frac{u_\theta(t+h) - u_\theta(t-h)}{2h} - f(t) \right|^2$$

- Also higher derivatives, e.g.

$$u_\theta^{(2)}(t) \approx \frac{u_\theta(t+h) - 2u(t) + u_\theta(t-h)}{h^2}$$

PINN - Parameter-Identification

- Given solution $u(x_1, t_1), ...u(x_n, t_n)$ of

$$\partial_t u = D\Delta u$$

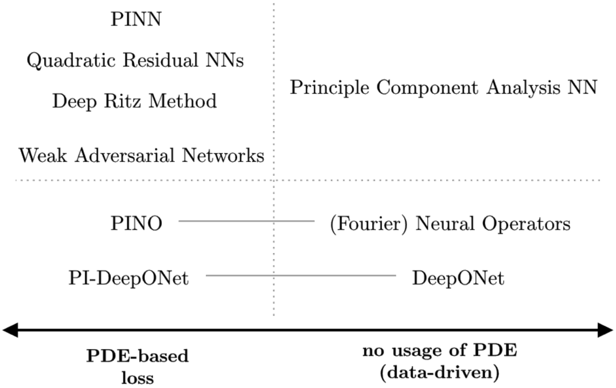
- First network $u_\theta \approx u$

$$\text{loss-term: } \frac{1}{n}\sum_{k=1}^{n} \left\| u_\theta(x_k, t_k) - u(x_k, t_k) \right\|^2$$

- Second network $D_\theta \in \mathbb{R}$ - trainable leaf-node

$$\text{loss-term: } \frac{1}{p}\sum_{k=1}^{p} \left\| \partial_t u_\theta(\tilde{x}_k, \tilde{t}_k) - D_\theta \Delta u_\theta(\tilde{x}_k, \tilde{t}_k) \right\|^2$$

Overview of other DL-Approaches



PINNs - Examples, Failures and Extensions

PINN - Heat Equation with Variable Diffusion

- Spatial domain $\Omega \subset \mathbb{R}^2$, time interval I_t,
- Parameter interval $I_D \subset \mathbb{R}$ for Diffusion
- For every $D \in I_D$ find solution $u(t, x; D)$ of

$$\begin{aligned} \partial_t u &= D\Delta u, \text{ in } \Omega \times I_t, \\ u(0, x; D) &= u_0(x), \text{ in } \Omega, \\ u(t, x; D) &= 0, \text{ on } \partial\Omega \times I_t. \end{aligned}$$

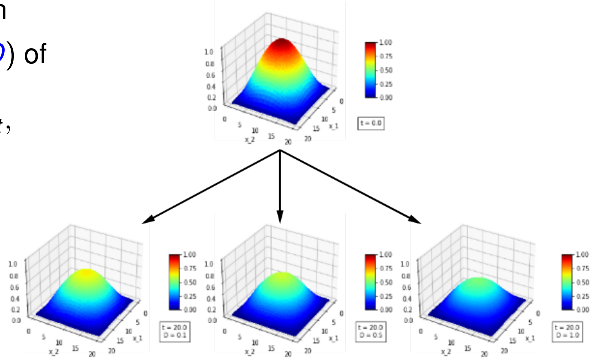
- Input of the NN is (t, x, D), so

$$u_\theta : \Omega \times I_t \times I_D \to \mathbb{R}$$

PINN - Heat Equation with Variable Diffusion

- Spatial domain $\Omega \subset \mathbb{R}^2$, time interval I_t,
- Parameter interval $I_D \subset \mathbb{R}$ for Diffusion
- For every $D \in I_D$ find solution $u(t, x; D)$ of

$$
\begin{aligned}
\partial_t u &= D\Delta u, \ \text{in } \Omega \times I_t, \\
u(0, x; D) &= u_0(x), \ \text{in } \Omega, \\
u(t, x; D) &= 0, \ \text{on } \partial\Omega \times I_t.
\end{aligned}
$$



- Input of the NN is (t, x, D), so

$$u_\theta : \Omega \times I_t \times I_D \to \mathbb{R}$$

PINN - Problem: High Frequencies

- Consider the problem $\Omega = [0,1] \times [0,1]$ and

$$\begin{aligned} \partial_y u &= y^{-1} u, && \text{in } \Omega, \\ u(x,0) &= 0, && \text{for } x \in [0,1], \\ u(x,1) &= \sin(20\pi x), && \text{for } x \in [0,1], \\ \vec{n} \nabla u(x,y) &= 0, && \text{for } x \in \{0,1\}, y \in [0,1]. \end{aligned}$$

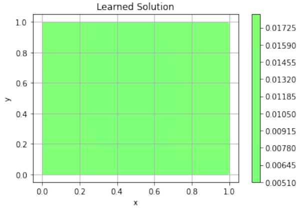
- Solution is $u(x,y) = y \sin(20\pi x)$

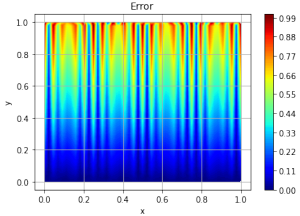
PINN - Problem: High Frequencies

- Consider the problem $\Omega = [0,1] \times [0,1]$ and

$$
\begin{aligned}
\partial_y u &= y^{-1} u, && \text{in } \Omega,\\
u(x,0) &= 0, && \text{for } x \in [0,1],\\
u(x,1) &= \sin(20\pi x), && \text{for } x \in [0,1],\\
\vec{n}\nabla u(x,y) &= 0, && \text{for } x \in \{0,1\}, y \in [0,1].
\end{aligned}
$$

- Solution is $u(x,y) = y\sin(20\pi x)$
- Default PINN-Approach does **not** work!



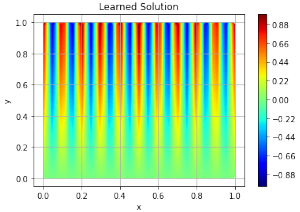


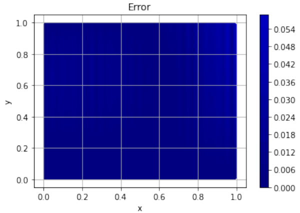
PINN - Problem: High Frequencies

- Consider the problem $\Omega = [0,1] \times [0,1]$ and

$$
\begin{aligned}
\partial_y u &= y^{-1} u, && \text{in } \Omega, \\
u(x,0) &= 0, && \text{for } x \in [0,1], \\
u(x,1) &= \sin(20\pi x), && \text{for } x \in [0,1], \\
\vec{n}\nabla u(x,y) &= 0, && \text{for } x \in \{0,1\}, y \in [0,1].
\end{aligned}
$$

- Solution is $u(x,y) = y \sin(20\pi x)$
- Ansatz: $\tilde{u}_\theta(x,y) = (1-y)u_\theta(x,y) + \sin(20\pi x)$



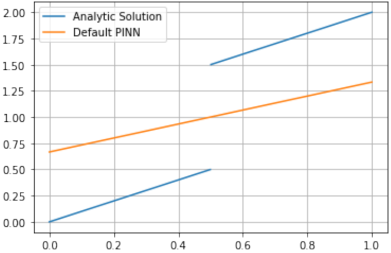


PINN - Approximation of Discontinuous Functions

- Find $u : [0,1] \to \mathbb{R}$ solving the coupled ODEs:

$$(1)\begin{cases} \partial_x^2 u_1(x) &= 0 \text{ for } x \in (0, 0.5) \\ u_1(0) &= 0 \\ \partial_x u_1(0.5) &= u_2(0.5) - u_1(0.5) \end{cases}$$

$$(2)\begin{cases} \partial_x^2 u_2(x) &= 0 \text{ for } x \in (0.5, 1) \\ u_2(0.5) &= 1.5 \\ \partial_x u_2(0.5) &= u_2(0.5) - u_1(0.5) \end{cases}$$



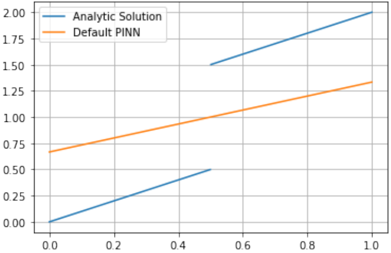
- Solution is $u(x) = x$, if $x <= 0.5$ and $= 1 + x$, if $x > 0.5$

PINN - Approximation of Discontinuous Functions

- Find $u : [0, 1] \to \mathbb{R}$ solving the coupled ODEs:

$$(1) \begin{cases} \partial_x^2 u_1(x) &= 0 \text{ for } x \in (0, 0.5) \\ u_1(0) &= 0 \\ \partial_x u_1(0.5) &= u_2(0.5) - u_1(0.5) \end{cases}$$

$$(2) \begin{cases} \partial_x^2 u_2(x) &= 0 \text{ for } x \in (0.5, 1) \\ u_2(0.5) &= 1.5 \\ \partial_x u_2(0.5) &= u_2(0.5) - u_1(0.5) \end{cases}$$



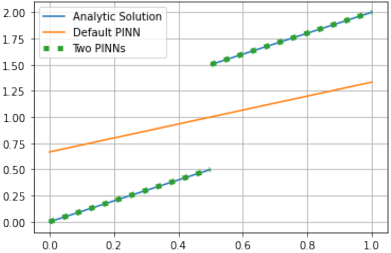
- Solution is $u(x) = x$, if $x <= 0.5$ and $= 1 + x$, if $x > 0.5$
- Default PINN-Approach does **not** work!

PINN - Approximation of Discontinuous Functions

- Find $u : [0,1] \to \mathbb{R}$ solving the coupled ODEs:

$$(1) \begin{cases} \partial_x^2 u_1(x) & = 0 \text{ for } x \in (0, 0.5) \\ u_1(0) & = 0 \\ \partial_x u_1(0.5) & = u_2(0.5) - u_1(0.5) \end{cases}$$

$$(2) \begin{cases} \partial_x^2 u_2(x) & = 0 \text{ for } x \in (0.5, 1) \\ u_2(0.5) & = 1.5 \\ \partial_x u_2(0.5) & = u_2(0.5) - u_1(0.5) \end{cases}$$



- Solution is $u(x) = x$, if $x <= 0.5$ and $= 1 + x$, if $x > 0.5$
- Default PINN-Approach does **not** work!
- Ansatz: Two individual networks for both subdomains

PINN - Advantages

Compared to classical methods

- 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
- Extension to optimization- & inverse problems easy to implement
- Parallelizable on multiple GPUs

PINN - 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

Exercise Session

- ODEs/PDEs can be lerned data-driven or physics-informed (or combi)
- Today's exercises:
  1) Recall basic PyTorch tensor syntax
  2) Manually implement data-driven and
  3) physics-informed learning, e.g. PINNs
- Tomorrow: More comfort with TorchPhysics + Parameter Identification
- Day after tomorrow: Introduction to Operator Learning

Appendix

PINN - Parameter-Identification

Parameter Function

- Look for spatially dependent diffusion $D(x)$ where $x \in \Omega$
- Given solution $u(x_1, t_1), ...u(x_n, t_n)$ of

$$\partial_t u = \operatorname{div}\left(D(x)\nabla u\right)$$

- First network $u_\theta \approx u$ as before
- Second network $D_\theta : \Omega \to \mathbb{R}$

$$\text{loss-term: } \frac{1}{p}\sum_{k=1}^{p}\left\|\partial_t u_\theta(\tilde{x}_k, \tilde{t}_k) - \operatorname{div}\left(D_\theta(\tilde{x}_k)\nabla u_\theta(\tilde{x}_k, \tilde{t}_k)\right)\right\|^2$$