Bauhaus-Universität Weimar

4 Introduction to Physics-Informed Neural Networks

Leon Herrmann

Stefan Kollmannsberger

Chair of Data Engineering in Construction

Bauhaus-Universität Weimar

Deep Learning in Computational Mechanics – an introductory course, Herrmann et al. 2025

website

book

Contents

4.1 Overview

- Data-driven solvers (such as the surrogate for strain distributions in Chapter 3) require data
  - Limited data availability
  - Breakeven threshold limits applicability

$$\tau = \frac{T_{\text{data}} + T_{\text{train}}}{T_{\text{simulation}} + T_{\text{surrogate}}}$$

- Incorporation of governing laws into neural networks to improve generalization with less data:
  - **Physics-informed learning**
    - Via the neural network architecture (by constraining the learnable space, i.e., strong enforcement; see Chapter 3 & 7)
    - Through training (via penalty terms in the cost function, i.e., weak enforcement/regularization; see Chapters **4** & 5)
      - **Physics-informed neural networks (PINNs):**

Solving differential equations through the minimization of the their residual (using gradient-based optimization)

Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, Raissi et al. 2019

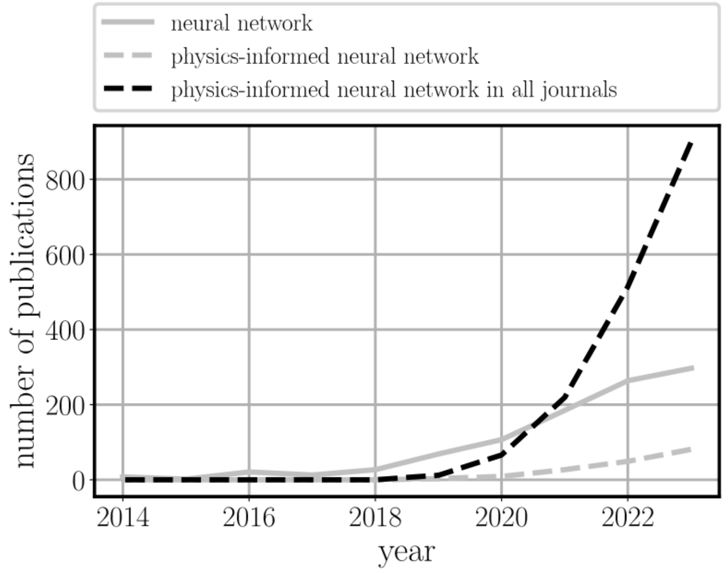
Bauhaus-Universität Weimar

4.1 Overview – Motivation

Spoiler:
currently **not competitive to classical methods** in practically all engineering scenarios
- possibly advantages in combination with other techniques
- maybe: Future potential (given breakthroughs in fundamental research in machine learning)

~ 27% of neural network-related computational mechanics publications cover physics-informed neural networks

Why do we cover physics-informed neural networks (PINNs) at all?
- One of the first deep learning techniques beyond pure supervised learning
- Immense popularity despite significant (and not widely known) shortcomings



Importantly:
- Introduction of various concepts with value beyond physics-informed neural networks

Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, Raissi et al. 2019

4.1 Overview

Physics-informed neural networks (PINNs) are useful, when only limited data is available.

To compensate for data scarcity, the neural networks are “enriched“ with physical laws

These laws can be formulated by differential equations of the form

$$\mathcal{N}[u;\lambda] = 0, x \in \Omega, t \in \mathcal{T}$$

Where

- $u(x,t)$ is the latent solution depending on space x and time t (defined on their respective domains Ω and $\mathcal{T}$)
- $\mathcal{N}[u;\lambda]$ is a nonlinear operator with coefficients λ

4.1 Overview

$$\mathcal{N}[u;\lambda] = 0, x \in \Omega, t \in \mathcal{T}$$

The two main use cases of PINNs are:

1. Data driven inference
   (the forward problem)

- The coefficients λ are known
- The (nonlinear) operator $\mathcal{N}$ is known
- The solution $u(x,t)$ is unknown

2. Data driven identification
   (the inverse problem)

- The coefficients λ are unknown
- **Or** the (nonlinear) operator $\mathcal{N}$ is unknown
- The solution $u(x,t)$ is at least partially known

For simplicity, let us now assume that the changes of u with respect to time are zero, then

$$\mathcal{N}[u;\lambda] = 0, x \in \Omega$$

Where

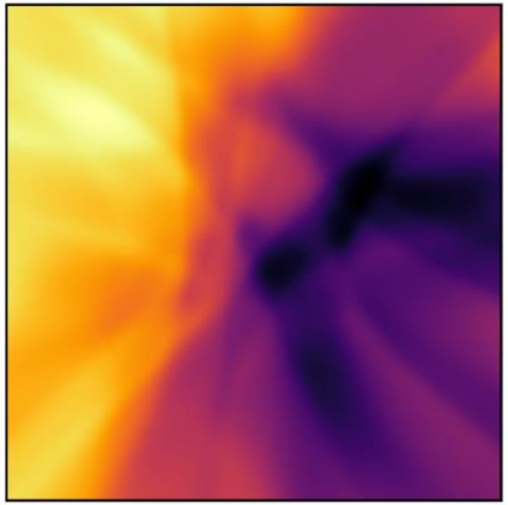
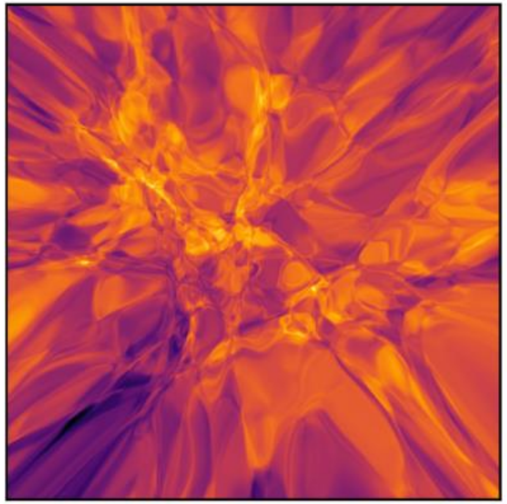
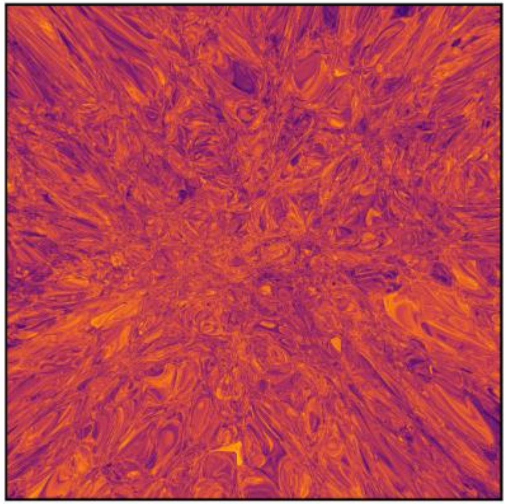
$$u(x)$$

4.1 Overview

We now remember the following essentials from the previous lectures

Chapter 3.2: Fully connected neural networks

can represent any (continuous) function $\hat{y} = f_{NN}(x)$

Bauhaus-Universität Weimar

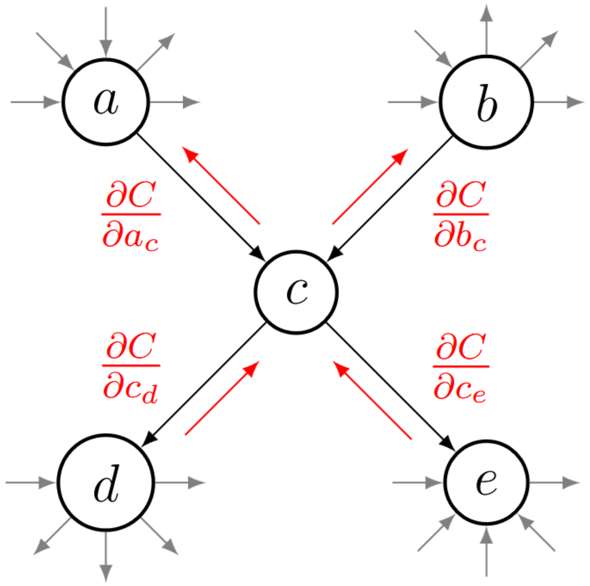
4.1 Overview

We now remember the following essentials from the previous lectures

Chapter 3.3: Differentiation

Given an output $\hat{y} = f_{NN}(x)$, the derivatives of the output $\hat{y}$ (or cost C) can be computed with respect to both

- The neural network parameters $\mathbf{\Theta}$
- The inputs x (see 3.4.2)



a
b
c
d
e
$\frac{\partial C}{\partial a_c}$
$\frac{\partial C}{\partial b_c}$
$\frac{\partial C}{\partial c_d}$
$\frac{\partial C}{\partial c_e}$

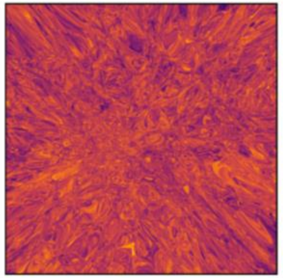
4.1 Overview

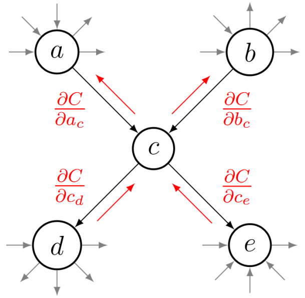
Therefore, we can

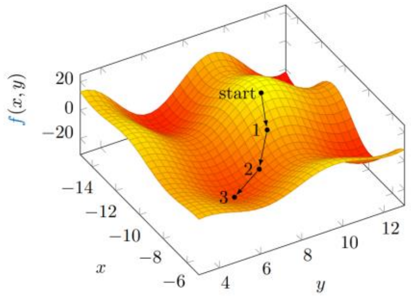
- Set up a network to approximate the solution $u(x)$
- Compute the derivative of the solution $u(x)$ with respect to its input x for

$$\mathcal{N}[u;\lambda] = 0, x \in \Omega, t \in \mathcal{T}$$

- Compute a loss function (residual of the differential equation)
  - And derive it with respect to $\boldsymbol{\Theta}$
- Use a gradient-based optimizer to minimize that residual

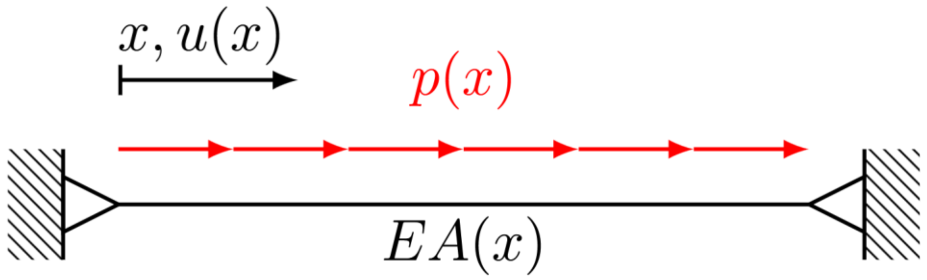



a
b
c
d
e
$\frac{\partial C}{\partial a_c}$
$\frac{\partial C}{\partial b_c}$
$\frac{\partial C}{\partial c_d}$
$\frac{\partial C}{\partial c_e}$



4.2 Data-Driven Inference

4.2.1 A one-dimensional static model problem



$$\mathcal{N}[u;\lambda] = 0, x \in \Omega, t \in \mathcal{T}$$

Where

$$\mathcal{N}[u] = \frac{d}{dx}\left(EA\frac{du}{dx}\right) + p = 0 \quad x \in \Omega$$

$$EA\frac{du}{dx} = F \quad x \in \Gamma_N$$

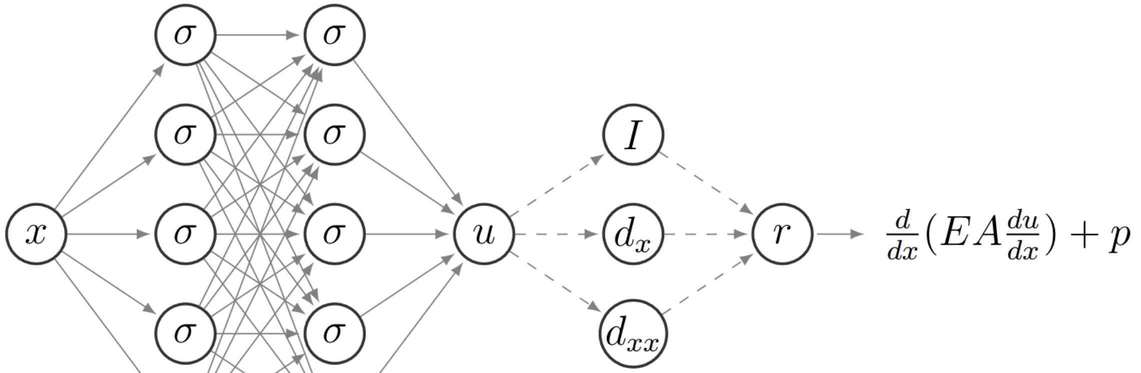
$$u = g \quad x \in \Gamma_D$$

Neumann boundary conditions

Dirichlet boundary conditions

4.2 Data-Driven Inference

Conceptual point of view



Fully connected neural network $u_{NN}(x)$:
Approximates the solution

Physics-informed part, i.e., $r(x)$:

Computation of the residual r based on $\mathcal{N}[u] = \frac{d}{dx}\left(EA\frac{du}{dx}\right) + p$

4.2 Data-Driven Inference – Manufactured Solution

The static model problem: **a manufactured solution** (even more specific)

$$\boxed{\mathcal{N}[u] = \frac{d}{dx}\left(EA\frac{du}{dx}\right) + p = 0}$$

Choose zero-Dirichlet boundary conditions, i.e.,

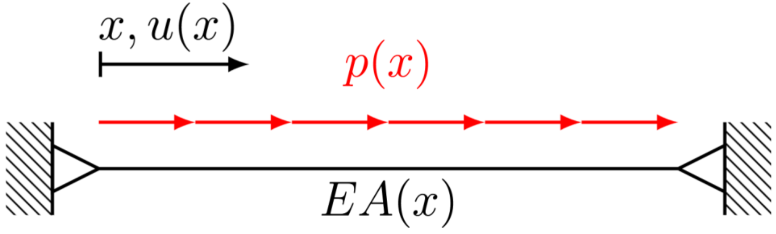
$$\boxed{u(0) = u(1) = 0}$$

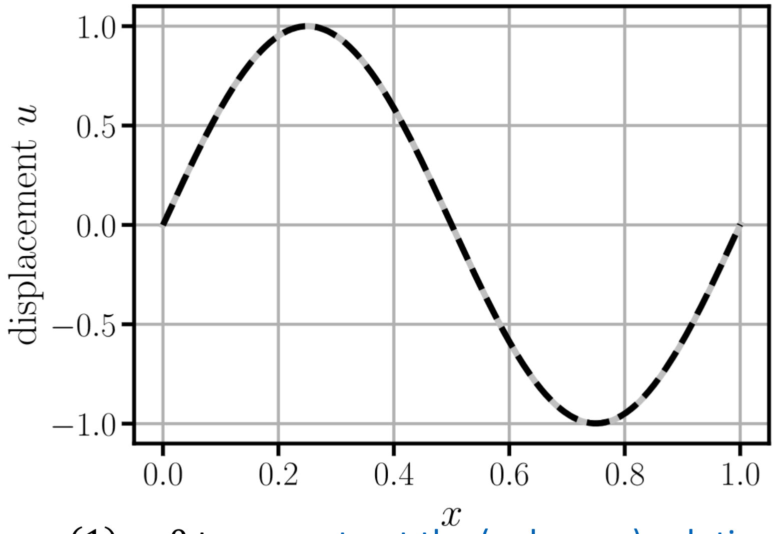
Assume the following solution with the domain

$\boxed{u(x) = \sin(2\pi x)}$ obeys the Dirichlet boundary conditions

Insert into differential equation

$\boxed{p(x) = 4\pi^2 \sin(2\pi x)}$ is the corresponding load





Problem to be solved:

Use the distributed load $p(x)$ and the boundary conditions $u(0) = u(1) = 0$ to reconstruct the (unknown) solution u

4.2 Data-Driven Inference – Cost

This is carried out by minimizing the cost function

$$C = \mathcal{L}_B + \mathcal{L}_R$$

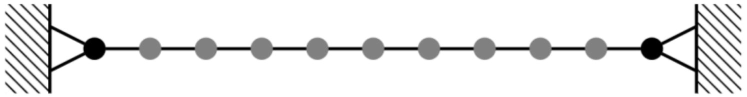
$$\hat{r}(u(x)) = \frac{d}{dx}\left(EA\frac{du}{dx}\right)\bigg|_x + p(x)$$

Where

Boundary loss:

$$\mathcal{L}_B = \frac{1}{m_B}\sum_{i=1}^{m_B}\left(u_{NN}(\tilde{x}_B^i) - \tilde{u}_B^i\right)^2$$

m_B: labeled data points at the boundary of the domain $\{\tilde{x}_B^i, \tilde{u}_B^i\}_{i=1}^{m_B}$

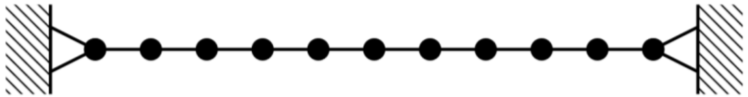


Approximation error at known data points (could also be $\mathcal{L}_u$ without boundary data)

Residual of PDE loss:

$$\mathcal{L}_R = \frac{1}{m_R}\sum_{i=1}^{m_R}\left(\hat{r}(u_{NN}(x_R^i))\right)^2$$

m_R: set of collocation points $\{x_R^i\}_{i=1}^{m_R}$



Approximation error at domain/collocation points

4.2 Data-Driven Inference – Algorithm

General algorithm

Algorithm 9 Training a physics-informed neural network for the static solution of the problem described in Equation (4.4)

```
Require: training data for boundary condition $\{\tilde{x}^i_{\mathcal{B}}, \tilde{u}^i_{\mathcal{B}}\}_{i=1}^{m_{\mathcal{B}}}$
  generate $m_{\mathcal{R}}$ collocation points with a uniform distribution $\{x^i_{\mathcal{R}}\}_{i=1}^{m_{\mathcal{R}}}$
  define network architecture (input, output, hidden layers, hidden neurons)
  initialize network parameters $\boldsymbol{\Theta}$: weights $\{\boldsymbol{W}^l\}_{l=1}^{L}$ and biases $\{\boldsymbol{b}^l\}_{l=1}^{L}$ for all layers $L$
  set hyperparameters for optimizer (epochs, learning rate $\alpha$, ...)

  for all epochs do
      $\hat{\boldsymbol{u}}_{\mathcal{B}} \leftarrow u_{NN}(\tilde{\boldsymbol{x}}_{\mathcal{B}}; \boldsymbol{\Theta})$
      $\hat{\boldsymbol{r}} \leftarrow r_{NN}(\boldsymbol{x}_{\mathcal{R}}; \boldsymbol{\Theta})$
      compute $\mathcal{L}_{\mathcal{B}}$, $\mathcal{L}_{\mathcal{R}}$                    ▷ cf. Equations (4.12) and (4.13)
      compute cost function: $C \leftarrow \mathcal{L}_{\mathcal{B}} + \mathcal{L}_{\mathcal{R}}$
      update parameters: $\boldsymbol{\Theta} \leftarrow \boldsymbol{\Theta} - \alpha \frac{\partial C}{\partial \boldsymbol{\Theta}}$                    ▷ Adam or L-BFGS
  end for
```

4.2 Data-Driven Inference – PyTorch

- Neural network architecture

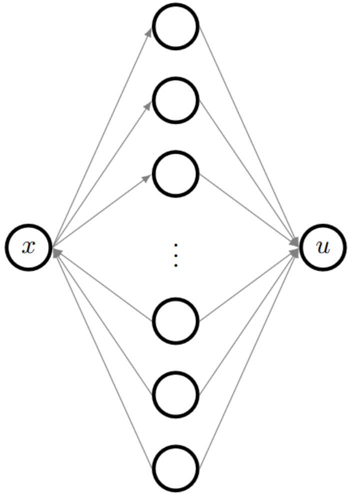
```
model = torch.nn.Sequential(torch.nn.Linear(inputDim, hiddenDim),
                            torch.nn.Tanh(),
                            torch.nn.Linear(hiddenDim, outputDim))
```

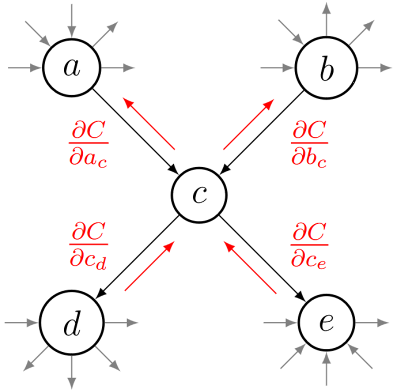
- Prediction

```
uPred = model(torch.tensor([0.5]))
```

- Helper function for gradient computation

```
def getDerivative(y, x):
    dydx = grad(y, x, torch.ones_like(y),
                create_graph=True,
                retain_graph=True)[0]
    return dydx
```

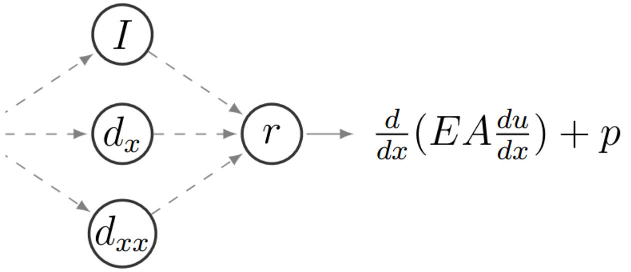



a
b
c
d
e
$\frac{\partial C}{\partial a_c}$
$\frac{\partial C}{\partial b_c}$
$\frac{\partial C}{\partial c_d}$
$\frac{\partial C}{\partial c_e}$

4.2 Data-Driven Inference – PyTorch

- Function for residual computation

```
def r(model, x, EA, p):
    u = model(x)
    dudx = getDerivative(u, x)
    dEAdudxx = getDerivative(EA(x) * dudx, x)
    r = dEAdudxx + p(x)
    return r
```

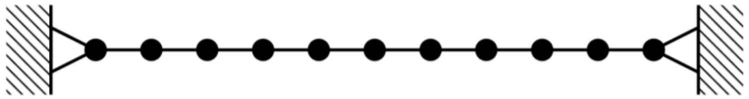


Remember the manufactured solution $u(x) = \sin(2\pi x)$ with the load $p(x) = 4\pi^2 \sin(2\pi x)$

- Residual loss $\mathcal{L}_R$ computation

```
x = torch.linspace(0, 1, 10, requires_grad=True).unsqueeze(1)
EA = lambda x: 1 + 0 * x
p = lambda x: 4 * torch.pi ** 2 * torch.sin(2 * torch.pi * x)
rPred = r(model, x, EA, p)
lossR = torch.sum(rPred ** 2)
```

$$\mathcal{L}_R = \frac{1}{m_R}\sum_{i=1}^{m_R}\left(\hat{r}(u_{NN}(x_R^i))\right)^2$$



Bauhaus-Universität Weimar

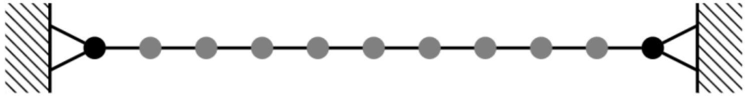
4.2 Data-Driven Inference – PyTorch

- Boundary loss $\mathcal{L}_B$ computation

```
u0 = 0
u1 = 0

u0Pred = model(torch.tensor([0.]))
u1Pred = model(torch.tensor([1.]))
lossB = (u0Pred – u0) ** 2 + (u1Pred – u1) ** 2
```

$$\mathcal{L}_B = \frac{1}{m_B} \sum_{i=1}^{m_B} \left(u_{NN}\left(\tilde{x}_B^i\right) - \tilde{u}_B^i \right)^2$$

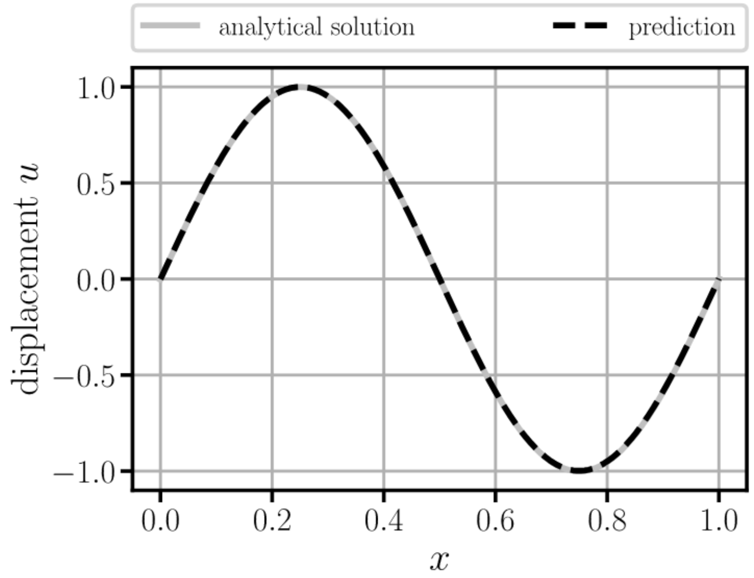


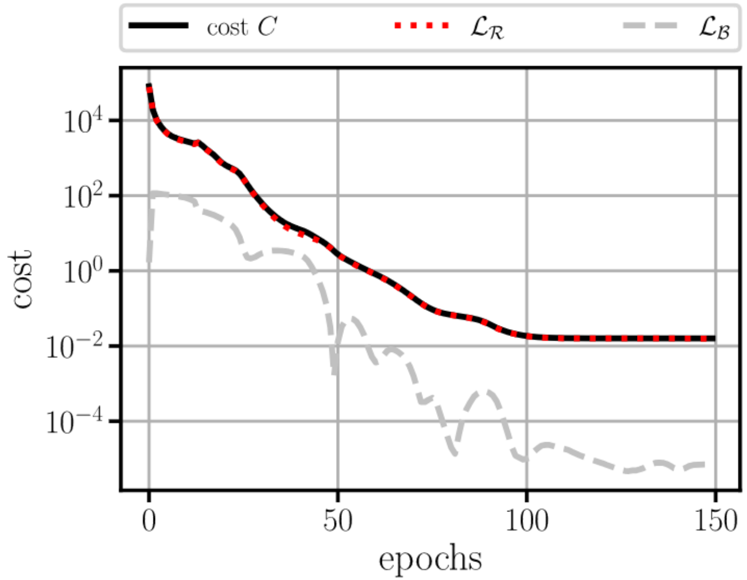
- Training according to gradient-based optimization from Chapter 3 using cost C

```
cost = lossR + lossB
```

4.2.1 One-Dimensional Static Model

Results after 50 epochs with Adam





Differences in magnitude between $\mathcal{L}_R$ and $\mathcal{L}_B$ $\rightarrow$ delay/prevention of convergence; possible remedies:

- Weighting factors between the terms
- Satisfaction of Dirichlet boundary conditions a priori

Will be covered in detail in Chapter 5

Exercises

E.14 Physics-Informed Neural Network for a Static Bar (P & C)

- Apply a physics-informed neural network to a static bar problem. Start with constructing a manufactured solution to verify your implementation.

E.16 Normalization and Weighting (C)

- Scale the manufactured solution of your physics-informed neural network from E.14 and observe how convergence is affected, i.e., the importance of normalization. In addition, introduce weighting factors between the loss terms.

4.2.2 Two-Dimensional Static Model

Plate in membrane action:

- Governed by differential equation

$$\nabla \cdot \boldsymbol{\sigma}(x_1, x_2) + \boldsymbol{p}(x_1, x_2) = \mathbf{0}, \qquad \boldsymbol{x} \in \Omega$$

- With the stresses $\boldsymbol{\sigma}$ computed from the strains $\boldsymbol{\varepsilon}$

$$\boldsymbol{\sigma} = \begin{pmatrix} \sigma_{11} \\ \sigma_{22} \\ \sigma_{12} \end{pmatrix} = \boldsymbol{C\varepsilon} = \frac{E}{1-\nu^2} \begin{pmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & 1-\nu \end{pmatrix} \boldsymbol{\varepsilon}$$

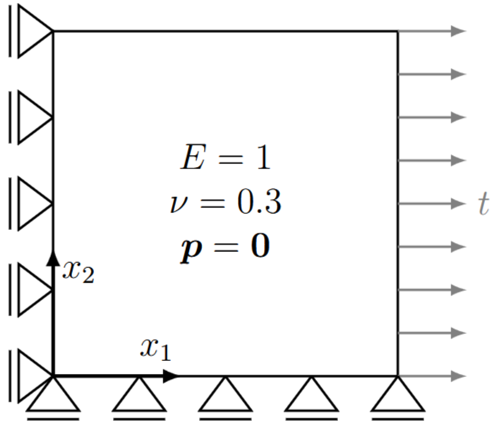
Assuming plane stress

- Which are computed from the displacements $\boldsymbol{u} = (u_1, u_2)^T$

$$\boldsymbol{\varepsilon} = \begin{pmatrix} \varepsilon_{11} \\ \varepsilon_{22} \\ \varepsilon_{12} \end{pmatrix} = \frac{1}{2}\left(\nabla \boldsymbol{u}(x_1, x_2) + \nabla \boldsymbol{u}(x_1, x_2)^T\right)$$

- Let's consider a plate under uni-axial tension with an analytical solution

$$\boldsymbol{u}^{\text{Analytical}} = \left(\frac{x_1 t}{E}, -\frac{x_2 \nu t}{E}\right)^T$$

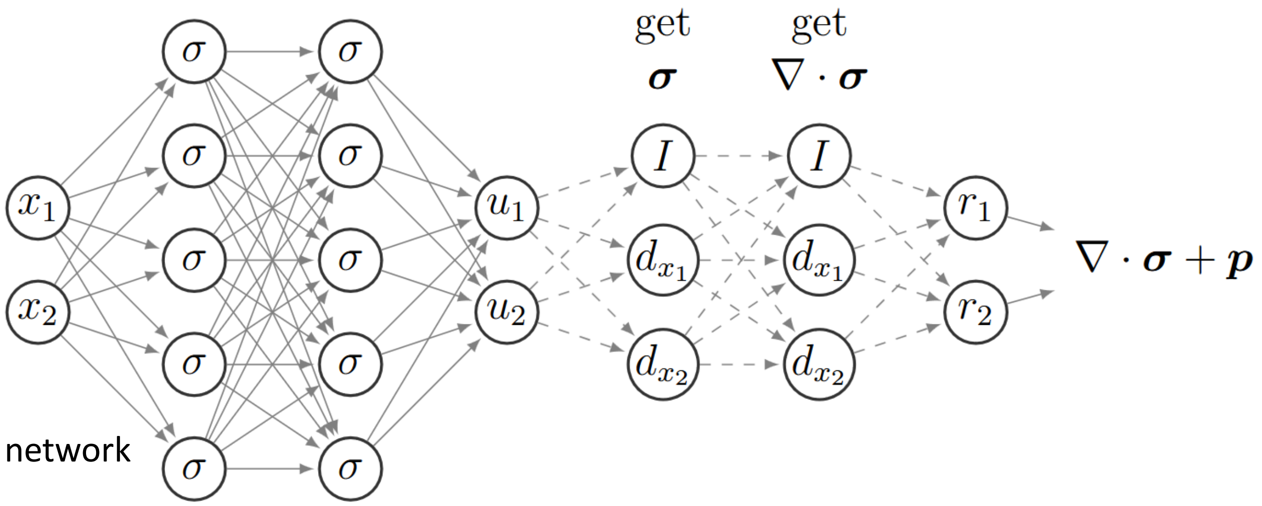


Boundary conditions

Bottom edge	$u_2(x_1, 0) = 0, \sigma_{12}(x_1, 0) = 0$
Right edge	$u_1(0, x_2) = 0, \sigma_{12}(0, x_2) = 0$
Top edge	$\sigma_{22}(x_1, L) = 0, \sigma_{12}(x_1, L) = 0$
Left edge	$\sigma_{11}(L, x_2) = t, \sigma_{12}(L, x_2) = 0$

4.2.2 Two-Dimensional Static Model

Conceptual point of view



Prediction of $\boldsymbol{u}$ with neural network

$\hat{\boldsymbol{u}} = u_{NN}(x_1, x_2)$

Residual (vector) is given as

$$\hat{\boldsymbol{r}} = \nabla \cdot \boldsymbol{\sigma}(\boldsymbol{u}) + \boldsymbol{p}$$

Residual loss

$$\mathcal{L}_R = \frac{1}{m_R} \sum_{i=1}^{m_R} \sum_{j=1}^{2} \left(\hat{\boldsymbol{r}}_j \left(x_{R_1}^i, x_{R_2}^i \right) \right)^2$$

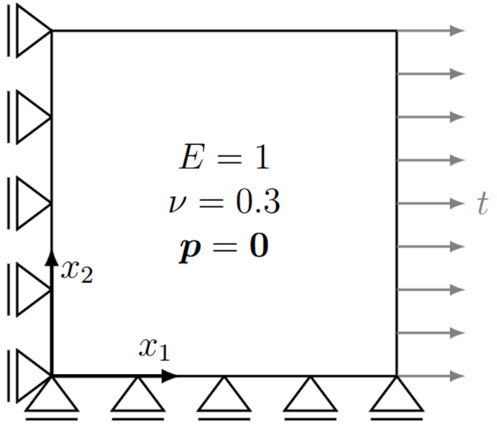
4.2.2 Two-Dimensional Static Model

Combination with boundary loss $\mathcal{L}_B$

$$C = \mathcal{L}_R + \mathcal{L}_B$$

But how can we avoid the competition between $\mathcal{L}_R$ and $\mathcal{L}_B$?

- Strong enforcement of Dirichlet boundary conditions:
  - Modification of neural network output $\boldsymbol{z} = u_{NN}(x_1, x_2)$
    $$\boldsymbol{u} = \begin{pmatrix} x_1 z_1 \\ x_2 z_2 \end{pmatrix}$$
  - If $\boldsymbol{u}$ is used in its modified formulation, the Dirichlet boundary conditions are satisfied a priori
  - $\mathcal{L}_B$ is however still needed for the Neumann boundary conditions (given via σ)



$$\mathcal{L}_B = \frac{1}{m_{B_{11}}} \sum_{i=1}^{m_{B_{11}}} \left(\sigma_{11}\left[\hat{\mathbf{u}}\left(\tilde{x}^i_{B_1}, \tilde{x}^i_{B_2}\right)\right] - \tilde{\sigma}^i_{B_{11}}\right)^2$$
$$+ \frac{1}{m_{B_{22}}} \sum_{i=1}^{m_{B_{22}}} \left(\sigma_{22}\left[\hat{\mathbf{u}}\left(\tilde{x}^i_{B_1}, \tilde{x}^i_{B_2}\right)\right] - \tilde{\sigma}^i_{B_{22}}\right)^2$$
$$+ \cdots$$

Boundary conditions

Bottom edge	$u_2(x_1, 0) = 0, \sigma_{12}(x_1, 0) = 0$
Right edge	$u_1(0, x_2) = 0, \sigma_{12}(0, x_2) = 0$
Top edge	$\sigma_{22}(x_1, L) = 0, \sigma_{12}(x_1, L) = 0$
Left edge	$\sigma_{11}(L, x_2) = t, \sigma_{12}(L, x_2) = 0$

4.2.2 Two-Dimensional Static Model

Due to the Neumann boundary conditions, the competition between $\mathcal{L}_B$ and $\mathcal{L}_R$ persists

Weighting terms $\boldsymbol{\kappa} = (\kappa_B, \kappa_R)$ as remedy

$$C = \kappa_B \mathcal{L}_B + \kappa_R \mathcal{L}_R$$

But how to choose weighting terms?

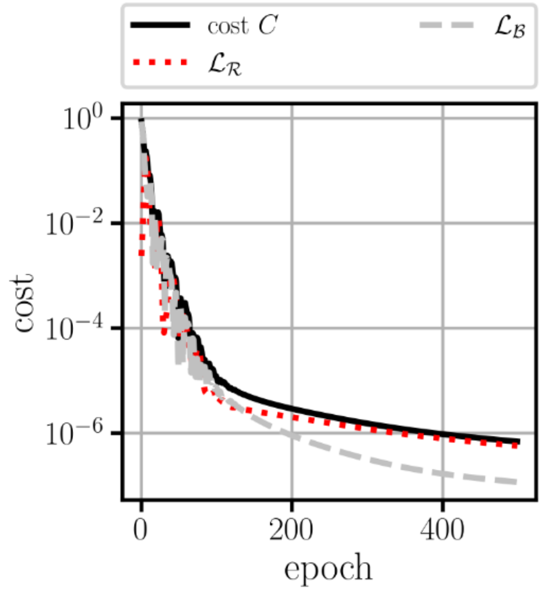
- Set value manually & tune (treatment as hyperparameter)
- Automation through optimization (idea is to focus on more critical term of C, i.e., use attention):
  - Maximize C with respect to the more critical loss (by increasing κ_B or κ_R)
  - Achieved through minimax optimization

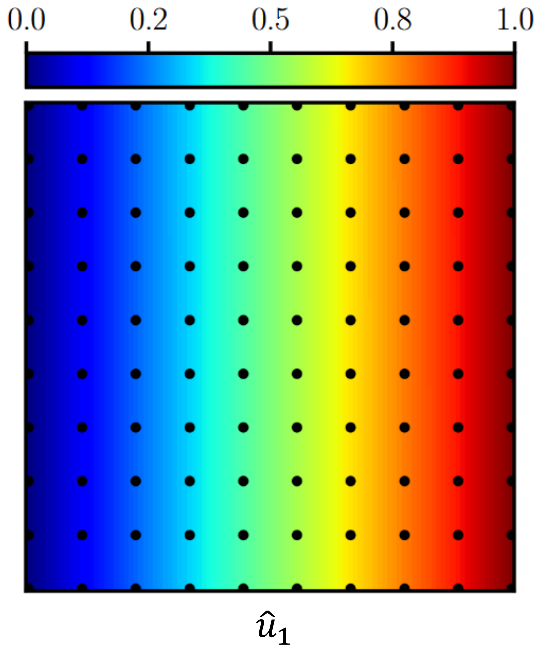
$$\min_{\boldsymbol{\Theta}} \max_{\boldsymbol{\kappa}} C$$

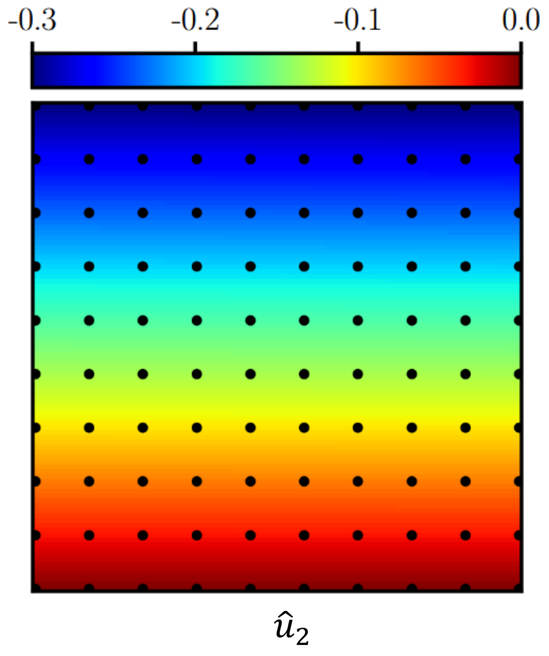
  - In practice $\boldsymbol{\kappa}$ is simply treated as $\boldsymbol{\Theta}$ during the optimization (but with an inversion of the gradient's sign)

4.2.2 Two-Dimensional Static Model

Results after 500 epochs with Adam





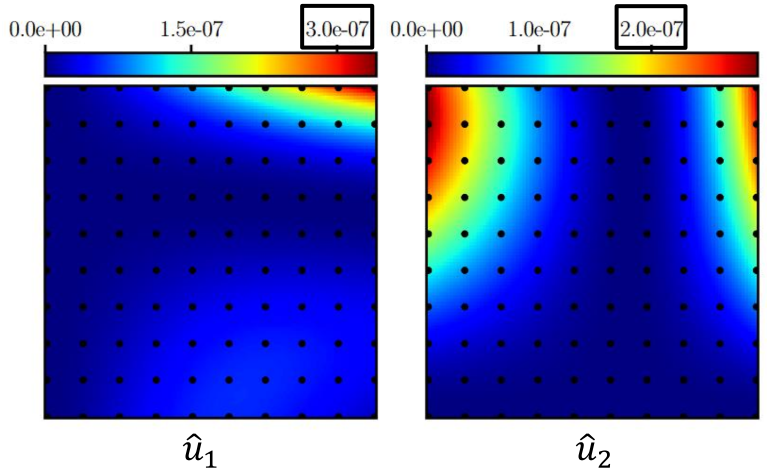


How does the prediction (& computational effort) hold up in comparison to the finite element method?

4.2.2 Two-Dimensional Static Model

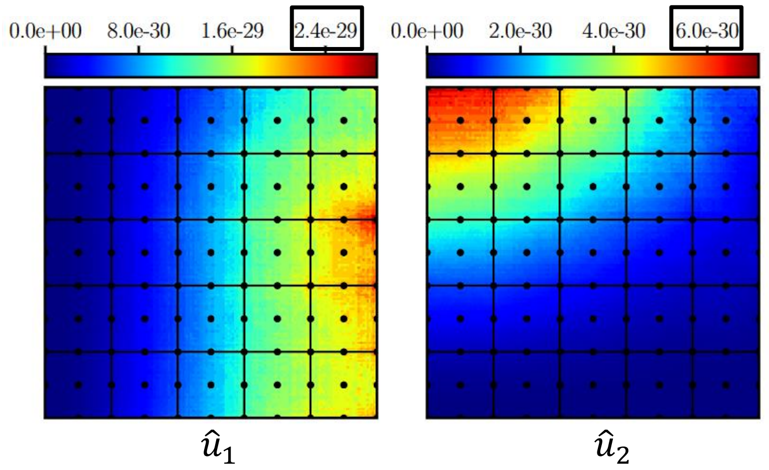
Physics-informed neural network

- 10×10 collocation points
- Total compute time: $500 \cdot\sim 1.2 \cdot 10^{-2}$ s (500 epochs)
- Mean squared error to analytical solution:



Finite element method

- 5×5 elements with $p = 2$ (242 degrees of freedom)
- Total compute time: $\sim 9.1 \cdot 10^{-3}$ s
- Mean squared error to analytical solution:



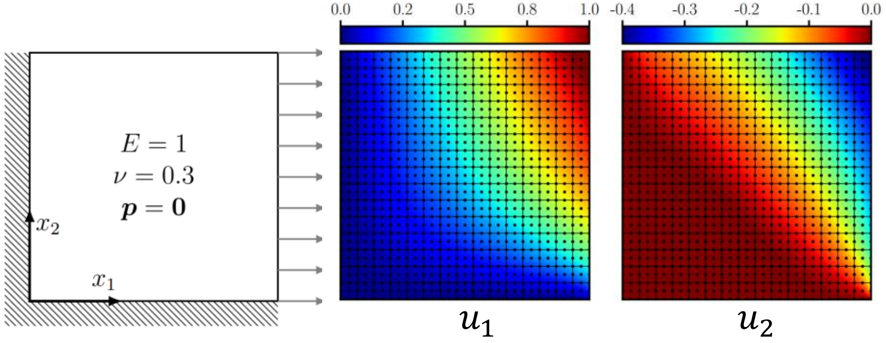
PINNs as forward solvers have no advantage over FEM! shown for linear elasticity (benefits might be possible for nonlinear or high-dimensional problems)

Bauhaus-Universität Weimar

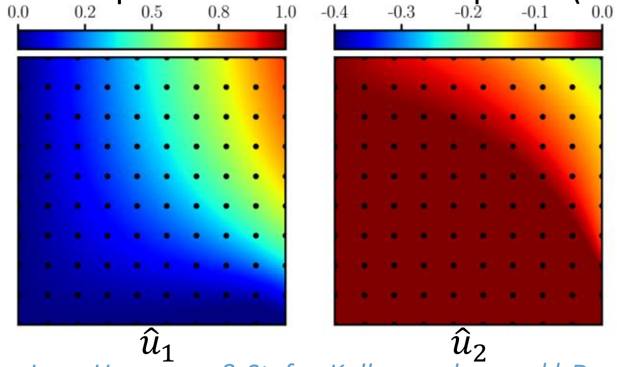
4.2 Data-Driven Inference

Two-dimensional static model with different loading

Finite element solution as reference ($\sim 8.3 \cdot 10^{-2}$ s)

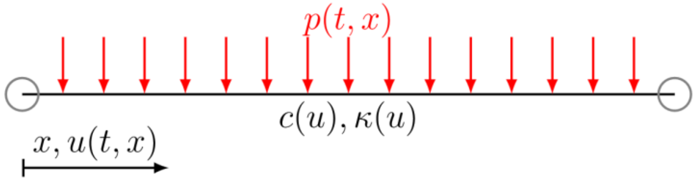


PINN prediction after 5‘000 epochs ($\sim 7 \cdot 10^{2}$ s)

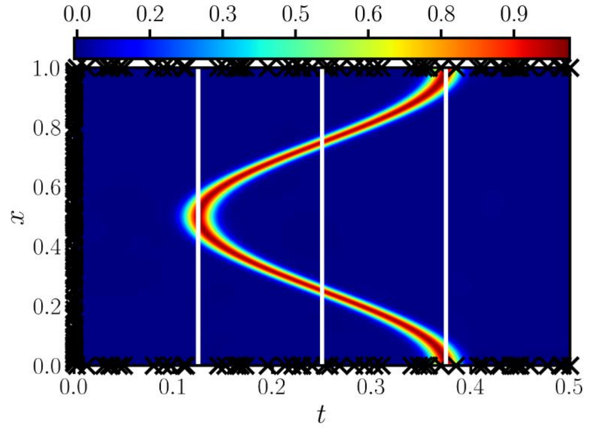


Stress concentration is not captured by the PINN

Continuous-Time Model (4.2.3): Nonlinear heat equation



PINN (temperature) prediction after 5’000 epochs ($\sim 6.6 \cdot 10^{2}$ s)



Required extensive tuning (with special loss weighting term)

Exercises

E.17 Physics-Informed Neural Network for a Plate in Membrane Action (C)

- Apply a physics-informed neural network to a plate in membrane action and compare the results with reference solutions obtained with the finite element method. Tune the neural network to improve convergence.

4.3 Data-Driven Identification

$$\mathcal{N}[u;\lambda] = 0, x \in \Omega, t \in \mathcal{T}$$

The two main use cases of PINNs are:

1. Data driven inference
   (the forward problem)
   - The coefficients λ are known
   - The (nonlinear) operator $\mathcal{N}$ is known
   - The solution $u(x,t)$ is unknown

2. Data driven identification
   (the inverse problem)
   - The coefficients λ are unknown
   - **Or** the (nonlinear) operator $\mathcal{N}$ is unknown
   - The solution $u(x,t)$ is at least partially known

For simplicity, let us now assume that the changes of u with respect to time are zero, then

$$\mathcal{N}[u;\lambda] = 0, x \in \Omega$$

Where

$$u(x)$$

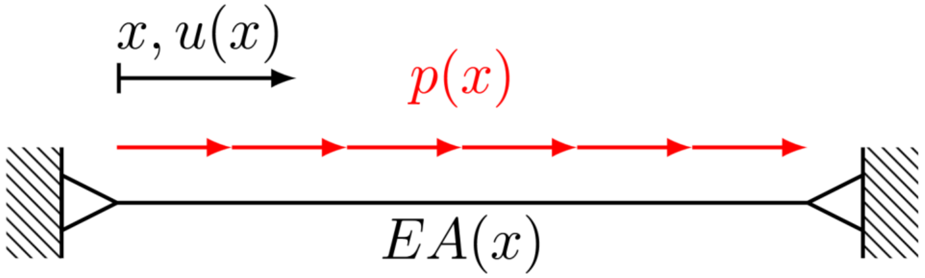
4.3 Data-Driven Identification

Reconsider the one-dimensional static model problem (from 4.2.1)

$$\mathcal{N}[u;EA] = \frac{d}{dx}\left(EA\frac{du}{dx}\right) + p = 0 \quad x \in \Omega$$
$$EA\frac{du}{dx} = F \qquad x \in \Gamma_N$$
$$u = g \qquad x \in \Gamma_D$$

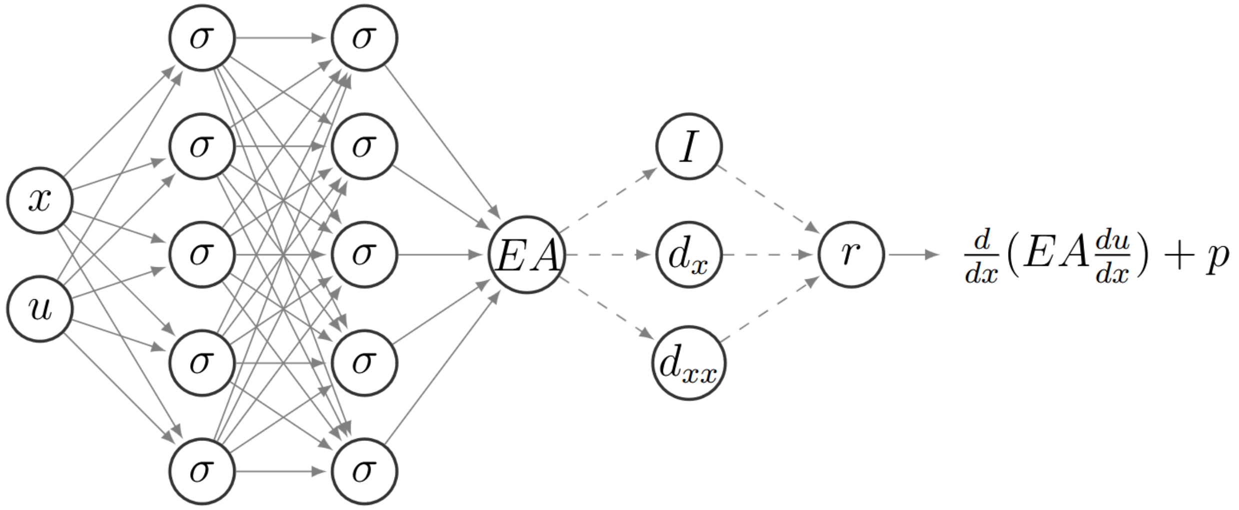
For the data-driven identification

- The displacement $u(x)$ is known along the domain Ω
- The **cross-sectional property** $\lambda = EA(x)$ is **unknown** along the domain Ω
- Thus: the nonlinear operator $\mathcal{N}[u;\lambda]$ is known, while the coefficients λ are unknown



4.3 Data-Driven Identification

Conceptual point of view



Prediction of the inverse quantity (the coefficients λ) via a neural network

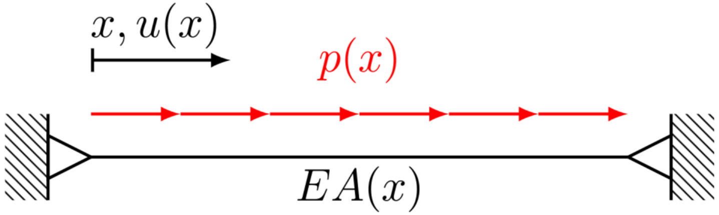
$$\widehat{EA}(x) = EA_{NN}(x)$$

Computation of the residual r based on $\mathcal{N}[u]$

$$r = \frac{d}{dx}\left(\widehat{EA}\frac{du}{dx}\right) + p$$

Bauhaus-Universität Weimar

4.3 Data-Driven Identification



Example with a manufactured solution

$$u(x) = \sin(2\pi x)$$
$$EA(x) = x^3 - x^2 + 1$$

After insertion into the differential equation

$$p(x) = -2(3x^2 - 2x)\pi\cos(2\pi x) + 4(x^3 - x^2 + 1)\pi^2\sin(2\pi x)$$

With two homogeneous Dirichlet boundary conditions

$$u(0) = u(1) = 0$$

Prediction of EA with a neural network $\widehat{EA} = EA_{NN}(x)$ from coordinates x

Optimization with the cost function C

$$C = \mathcal{L}_R$$

No boundary loss $\mathcal{L}_B$ as the Dirichlet boundary condition is already satisfied by the "measurement" u

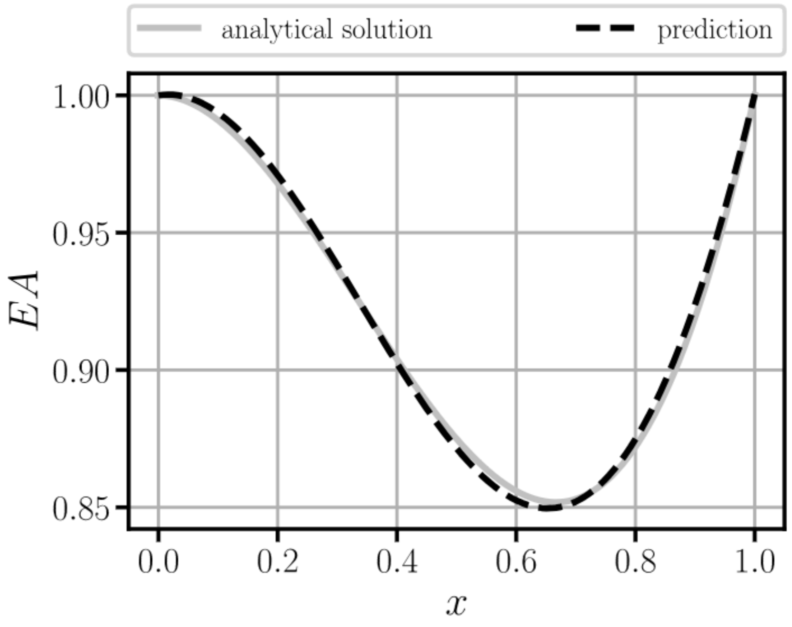
Where the residual of the differential equation is

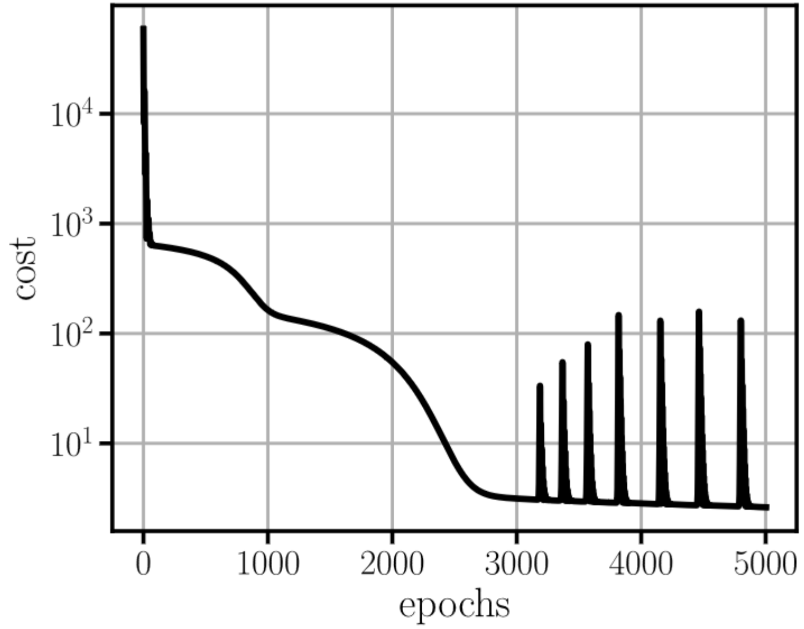
$$\mathcal{L}_R = \frac{1}{m_R}\sum_{i=1}^{m_R}\left(r\left(\widehat{EA}(x_R^i)\right)\right)^2$$

Bauhaus-Universität Weimar

4.3 Data-Driven Identification

Results after 5‘000 epochs with Adam





Physics-informed neural networks can be useful for data-driven identification (inverse problems) if full-field solution data is available (i.e., $u(x)$ in the entire domain Ω) → More details will follow in Chapter 9

Exercises

E.15 Data-Driven Identification using Physics-Informed Neural Networks for a Static Bar (P & C)

- Apply a physics-informed neural network to a static bar problem. Specifically, the aim is to identify the cross-sectional properties $EA(x)$ given the solution $u(x)$. Start with constructing a manufactured solution to verify your implementation.

Bauhaus-Universität Weimar

Contents

Bauhaus-Universität Weimar

4 Introduction to Physics-Informed Neural Networks

Leon Herrmann

Stefan Kollmannsberger

Chair of Data Engineering in Construction

Bauhaus-Universität Weimar

Deep Learning in Computational Mechanics – an introductory course, Herrmann et al. 2025

website

book