

Physics-informed machine learning – Hype or new trend in computational engineering?

Matthias Möller

Department of Applied Mathematics, TU Delft, The Netherlands

6th Short Course & Forum on Computational Fluid Dynamics in Rotary Positive Displacement Machines

City, University of London, 10-09-2023

ChatGPT

Can you explain to me the working principle of a twin screw compressor?

Can you explain to me the working principle of a twin screw compressor?

Certainly! A twin-screw compressor is a type of positive displacement compressor used to compress gases, most commonly air and refrigerant gases. It's called "twin-screw" because it consists of two interlocking helical rotors, usually referred to as the male rotor and the female rotor, that rotate in opposite directions within a tightly fitted casing. The working principle of a twin-screw compressor can be broken down into several key steps:

1. **Intake:** *The process begins with the rotation of the male and female rotors. As they turn, their helical lobes create expanding cavities between them and the compressor casing. This causes gas to be drawn into the compressor through the inlet port.*
2. **Compression:** *As the rotors continue to rotate, the interlocking lobes of the rotors reduce the volume of the cavities. This reduction in volume compresses the gas trapped in the cavities. The compression process occurs continuously as the gas is carried from the intake side towards the discharge side of the compressor.*
3. **Intermeshing:** ...
4. **Compression Ratio:** ...
5. **Discharge:** ...
6. **Oil Injection:** ...
7. **Cooling and Control:** ...

The design of a twin-screw compressor allows for continuous and relatively pulsation-free compression, making it well-suited for applications where a steady flow of compressed gas is required. Its efficiency, reliability, and ability to handle a wide range of operating conditions make it a popular choice in various industries.

Stable Diffusion (text-to-image generator)



“A sunny summer day in London”

Stable Diffusion (text-to-image generator)



“A sunny summer day in London”

“A twin screw compressor”

Stable Diffusion (text-to-image generator)



“A sunny summer day in London”



“A twin screw compressor”

① Introduction

Terminology

Deep learning 101

② Physics-informed machine learning

Physics-informed neural networks

Deep Operator neural networks

③ IgANets

Concept and preliminary applications

1 Introduction

Terminology

Deep learning 101

2 Physics-informed machine learning

Physics-informed neural networks

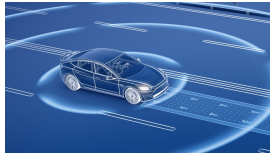
Deep Operator neural networks

3 IgANets

Concept and preliminary applications

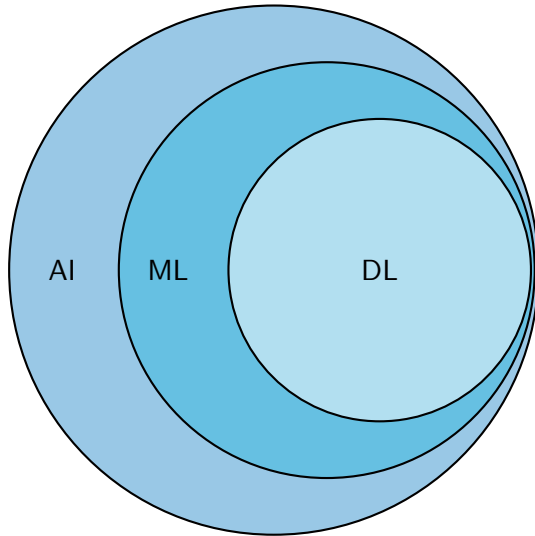
Terminology

- Artificial Intelligence (AI)
- Machine Learning (ML)
- Deep Learning (DL)
- Scientific Machine Learning (SciML)

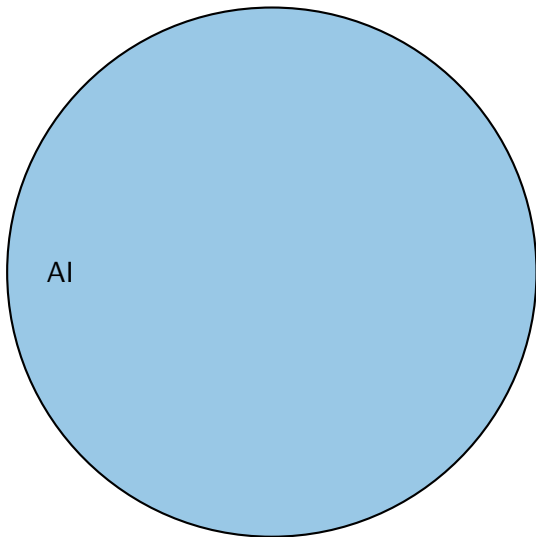


So what is what? And what is [physics-informed machine learning](#)?

Terminology



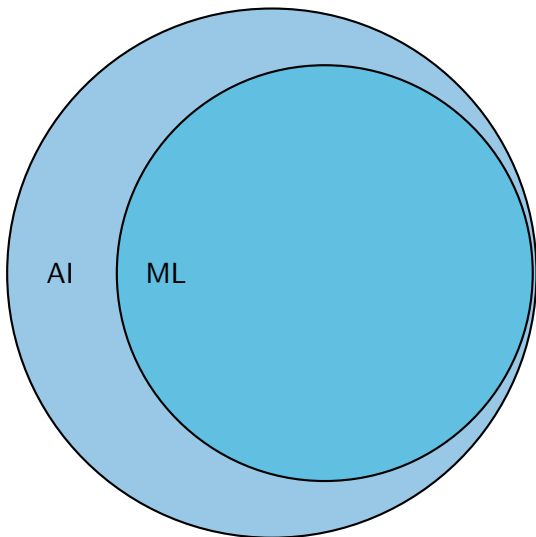
- **statistical modelling** often w/o using process-knowledge
- empirical discovery of new knowledge from (big) data
- Examples:
 - linear regression
 - PCA/SVD, POD
 - reduced-order models
 - self-driving cars
 - large language models (ChatGPT, Bard, Co-pilot)
 - text-to-image generators (Stable Diffusion)
 - ...



- “effort to *automate intellectual tasks* normally performed by humans” – François Chollet, creator of Keras
- Example: rule/label-based automatic packing machine



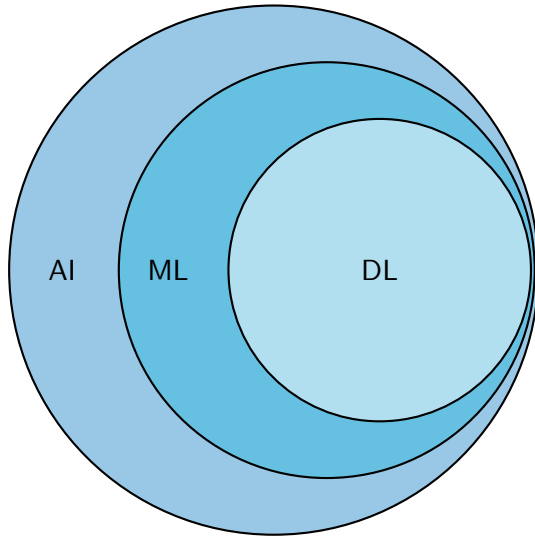
Terminology



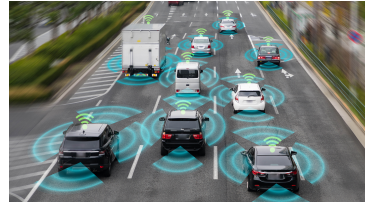
- “field of study that gives computers the ability to *learn without being explicitly programmed*” – Arthur Samuel, ML pioneer
- Example: classification-based waste sorting machine



Terminology

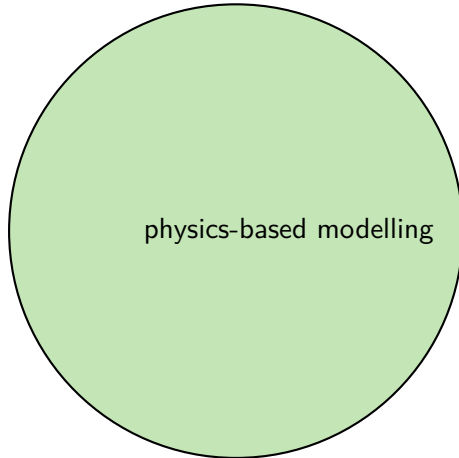


- next-level ML based on **deep artificial neural networks**
- Example: self-driving cars

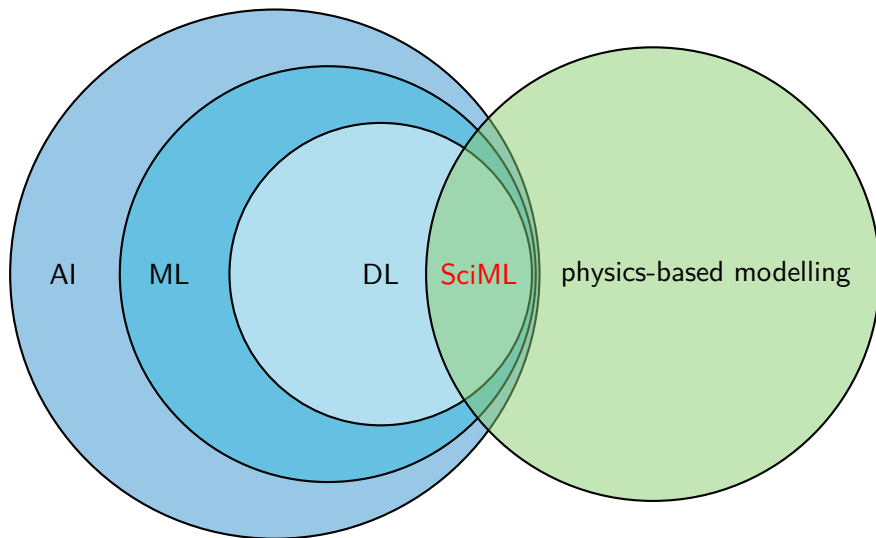


Terminology

- mathematical formulas based on **physics laws**
- derived from observational studies and experiments
- solved by numerical methods



Terminology



In a nutshell

Scientific Machine Learning [Baker et al., 2019] is *not* the replacement of physics-based modelling by statistical modelling but the clever combination of process-knowledge (i.e. mechanistic models) with the capability of ML/DL to '*learn*' from (big) data.

In a nutshell

Scientific Machine Learning [Baker et al., 2019] is *not* the replacement of physics-based modelling by statistical modelling but the clever combination of process-knowledge (i.e. mechanistic models) with the capability of ML/DL to '*learn*' from (big) data.

Synonyms

- *Theory-guided data science* [Karpatne et al., 2017]
- *Physically informed neural networks* [Raissi et al., 2019]
- *Universal differential equations* [Rackauckas et al., 2020]
- *Physics-informed machine learning* [Karniadakis et al., 2021a]
- *Physics-based deep learning* [Thuerey et al., 2021]

In a nutshell

Scientific Machine Learning [Baker et al., 2019] is *not* the replacement of physics-based modelling by statistical modelling but the clever combination of process-knowledge (i.e. mechanistic models) with the capability of ML/DL to '*learn*' from (big) data.

Synonyms

- *Theory-guided data science* [Karpatne et al., 2017]
- *Physically informed neural networks* [Raissi et al., 2019]
- *Universal differential equations* [Rackauckas et al., 2020]
- *Physics-informed machine learning* [Karniadakis et al., 2021a]
- *Physics-based deep learning* [Thuerey et al., 2021]

Note: Other AI/ML/DL approaches are not 'bad'. We just don't consider them here.

Timeline

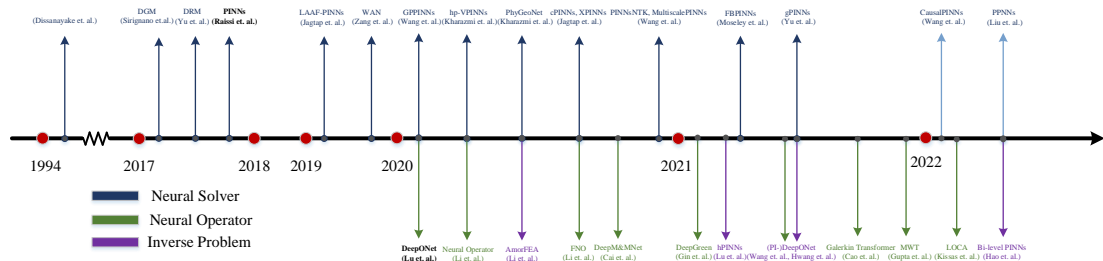


Figure from: *Physics-informed machine learning: A survey on problems, methods and applications* [Hao et al., 2022]

Notation

x scalar variable

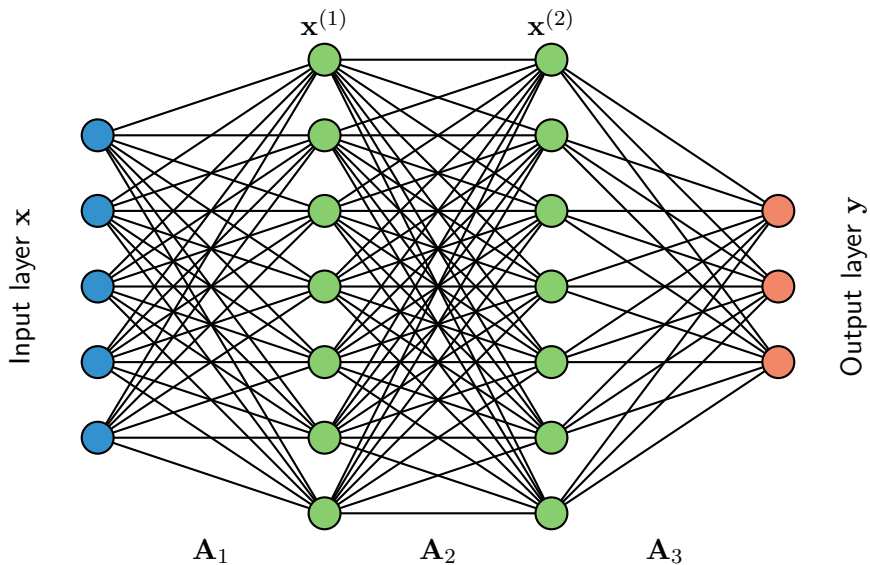
$f(x)$ function

$G(f)(x)$ operator

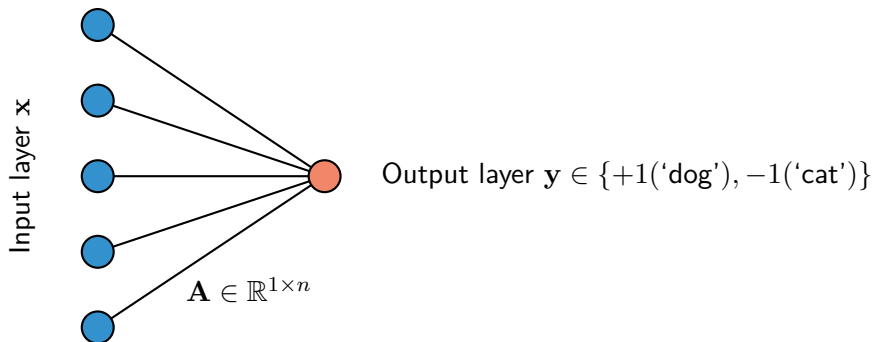
$\mathbf{x} = \begin{bmatrix} x_1 & \dots & x_n \end{bmatrix}$ vector

$\mathbf{A} = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mn} \end{bmatrix}$ matrix

Artificial neural network (ANN)



Single-layer network



Linear regression

$$\mathbf{AX} = \mathbf{Y} \quad \rightarrow \quad \begin{bmatrix} a_1 & a_2 & \dots & a_n \end{bmatrix} \begin{bmatrix} \begin{array}{c} | \\ \mathbf{x}_1 \\ | \end{array} & \begin{array}{c} | \\ \mathbf{x}_2 \\ | \end{array} & \dots & \begin{array}{c} | \\ \mathbf{x}_s \\ | \end{array} \end{bmatrix} = \begin{bmatrix} y_1 & y_1 & \dots & y_s \end{bmatrix}$$

Linear regression

$$\mathbf{A}\mathbf{X} = \mathbf{Y} \quad \rightarrow \quad \begin{bmatrix} a_1 & a_2 & \dots & a_n \end{bmatrix} \begin{bmatrix} \begin{bmatrix} | \\ \mathbf{x}_1 \\ | \end{bmatrix} & \begin{bmatrix} | \\ \mathbf{x}_2 \\ | \end{bmatrix} & \dots & \begin{bmatrix} | \\ \mathbf{x}_s \\ | \end{bmatrix} \end{bmatrix} = \begin{bmatrix} y_1 & y_1 & \dots & y_s \end{bmatrix}$$

Common solution approaches

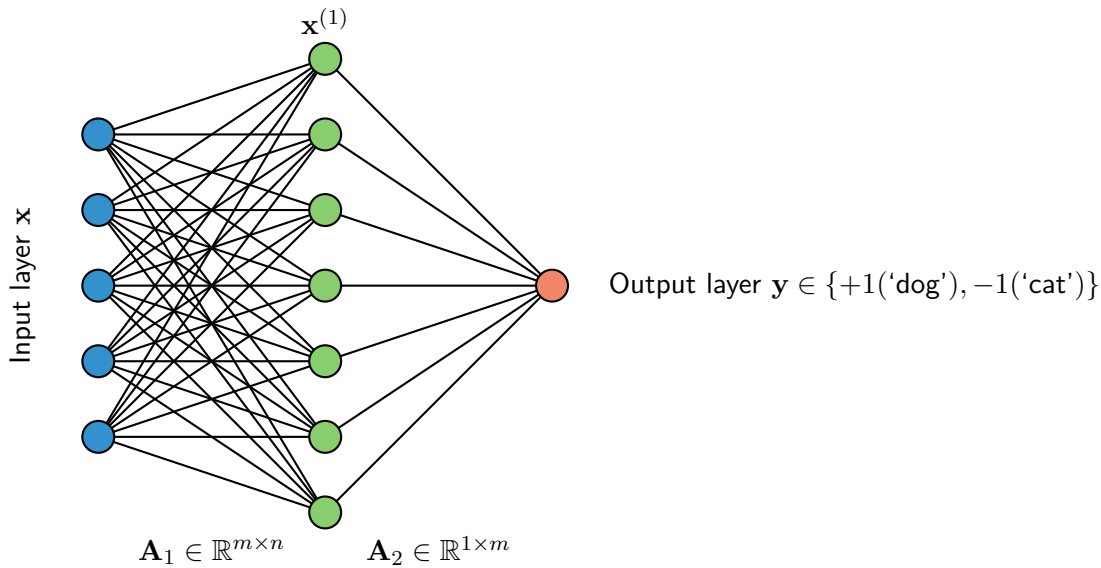
Pseudo-inverse

$$\mathbf{A} = \mathbf{Y}\mathbf{X}^\dagger$$

LASSO regression

$$\mathbf{A} = \operatorname{argmin}_{\mathbf{A}'} \|\mathbf{A}'\mathbf{X} - \mathbf{Y}\|_2 + \lambda \|\mathbf{A}'\|_1$$

Multi-layer network



Beyond linear regression

$$\left. \begin{array}{l} \mathbf{A}_1 \mathbf{X} = \mathbf{X}^{(1)} \\ \mathbf{A}_2 \mathbf{X}^{(1)} = \mathbf{Y} \end{array} \right\} \Leftrightarrow \mathbf{A}_2 \mathbf{A}_1 \mathbf{X} = \mathbf{Y} \Leftrightarrow \tilde{\mathbf{A}} \mathbf{X} = \mathbf{Y}$$

Beyond linear regression

$$\left. \begin{array}{l} \mathbf{A}_1 \mathbf{X} = \mathbf{X}^{(1)} \\ \mathbf{A}_2 \mathbf{X}^{(1)} = \mathbf{Y} \end{array} \right\} \Leftrightarrow \mathbf{A}_2 \mathbf{A}_1 \mathbf{X} = \mathbf{Y} \Leftrightarrow \tilde{\mathbf{A}} \mathbf{X} = \mathbf{Y}$$

Enrichment of functional responses by *nonlinear* activation functions

$$\left. \begin{array}{l} f_1(\mathbf{A}_1, \mathbf{X}) = \mathbf{X}^{(1)} \\ f_2(\mathbf{A}_2, \mathbf{X}^{(1)}) = \mathbf{Y} \end{array} \right\} \Leftrightarrow f_2(\mathbf{A}_2, f_1(\mathbf{A}_1, \mathbf{X})) = \mathbf{Y}$$

Activation functions

$$f(x) = x, \quad \text{linear}$$

$$f(x) = \begin{cases} 0 & \text{for } x \leq 0, \\ 1 & \text{for } x > 0, \end{cases} \quad \text{binary step}$$

$$f(x) = \frac{1}{1+\exp(-x)}, \quad \text{logistic (soft step)}$$

$$f(x) = \tanh(x), \quad \text{tanh}$$

$$f(x) = \begin{cases} 0 & \text{for } x \leq 0, \\ x & \text{for } x > 0, \end{cases} \quad \text{rectified linear unit (ReLU)}$$

Elementwise application

$$\mathbf{Y} = f(\mathbf{A}, \mathbf{X}) \quad :\Leftrightarrow \quad \mathbf{Y} = (y_{ij}), \quad y_{ij} = f(\tilde{y}_{ij}) \quad \forall i, j, \quad \tilde{\mathbf{Y}} = \mathbf{A}\mathbf{X}$$

Nonlinear optimization problem

$$\begin{bmatrix} \mathbf{A}_1 & \mathbf{A}_2 \end{bmatrix} = \underset{\mathbf{A}'_1, \mathbf{A}'_2}{\operatorname{argmin}} E(\mathbf{A}'_1, \mathbf{A}'_2)$$

In words: find **network weights** \mathbf{A}_1 and \mathbf{A}_2 that minimize the **mean-square error (MSE)**

$$E(\mathbf{A}_1, \mathbf{A}_2) = \frac{1}{s} \sum_{\ell=1}^s \left(f_2(\mathbf{A}_2, f_1(\mathbf{A}_1, \mathbf{x}^\ell)) - \mathbf{y}^\ell \right)^2$$

over the set of input-output pairs $(\mathbf{x}^\ell; \mathbf{y}^\ell)$ from the training data set

Nonlinear optimization problem

$$\begin{bmatrix} \mathbf{A}_1 & \mathbf{A}_2 \end{bmatrix} = \underset{\mathbf{A}'_1, \mathbf{A}'_2}{\operatorname{argmin}} E(\mathbf{A}'_1, \mathbf{A}'_2)$$

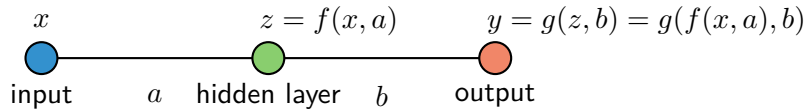
In words: find **network weights** \mathbf{A}_1 and \mathbf{A}_2 that minimize the **mean-square error (MSE)**

$$E(\mathbf{A}_1, \mathbf{A}_2) = \frac{1}{s} \sum_{\ell=1}^s \left(f_2(\mathbf{A}_2, f_1(\mathbf{A}_1, \mathbf{x}^\ell)) - \mathbf{y}^\ell \right)^2$$

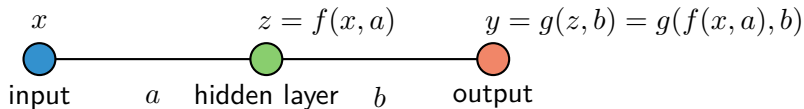
over the set of input-output pairs $(\mathbf{x}^\ell; \mathbf{y}^\ell)$ from the training data set

Solving nonlinear optimization problems is not new. A common approach in SciML is based on **backpropagation** and the **stochastic gradient descent algorithm**.

Backpropagation



Backpropagation

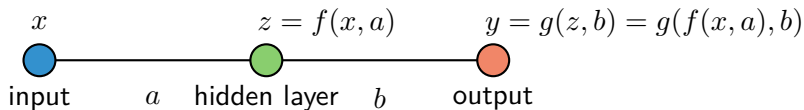


MSE $E = (y - \hat{y})^2$ between target output y and network output \hat{y} is minimal if

$$\frac{\partial E}{\partial a} = 0$$

$$\wedge \frac{\partial E}{\partial b} = 0$$

Backpropagation

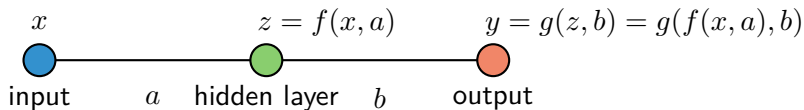


MSE $E = (y - \hat{y})^2$ between target output y and network output \hat{y} is minimal if

$$\frac{\partial E}{\partial a} = 2(y - \hat{y}) \left[\frac{dy}{da} - \frac{d\hat{y}}{da} \right] = 0$$

$$\wedge \frac{\partial E}{\partial b} = 2(y - \hat{y}) \left[\frac{dy}{db} - \frac{d\hat{y}}{db} \right] = 0$$

Backpropagation

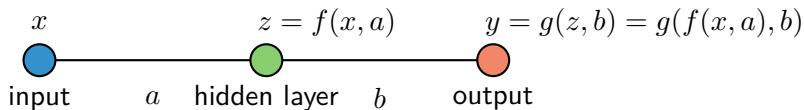


MSE $E = (y - \hat{y})^2$ between target output y and network output \hat{y} is minimal if

$$\frac{\partial E}{\partial a} = 2(y - \hat{y}) \left[\cancel{\frac{d\hat{y}}{da}} - \frac{d\hat{y}}{da} \right] = -2(y - \hat{y}) \left[\frac{d\hat{y}}{dz} \frac{dz}{da} + \frac{d\hat{y}}{db} \cancel{\frac{db}{da}} \right] = 0$$

$$\wedge \frac{\partial E}{\partial b} = 2(y - \hat{y}) \left[\cancel{\frac{d\hat{y}}{db}} - \frac{d\hat{y}}{db} \right] = -2(y - \hat{y}) \left[\frac{d\hat{y}}{dz} \cancel{\frac{dz}{db}} + \frac{d\hat{y}}{db} \frac{db}{db} \right] = 0$$

Backpropagation



MSE $E = (y - \hat{y})^2$ between target output y and network output \hat{y} is minimal if

$$\frac{\partial E}{\partial a} = 2(y - \hat{y}) \left[\frac{d\hat{y}}{da} - \frac{d\hat{y}}{da} \right] = -2(y - \hat{y}) \left[\frac{d\hat{y}}{dz} \frac{dz}{da} + \frac{d\hat{y}}{db} \frac{db}{da} \right] = -2(y - \hat{y}) \frac{d\hat{y}}{dz} \frac{dz}{da} = 0$$

$$\wedge \frac{\partial E}{\partial b} = 2(y - \hat{y}) \left[\frac{d\hat{y}}{db} - \frac{d\hat{y}}{db} \right] = -2(y - \hat{y}) \left[\frac{d\hat{y}}{dz} \frac{dz}{db} + \frac{d\hat{y}}{db} \frac{db}{db} \right] = -2(y - \hat{y}) \frac{d\hat{y}}{db} = 0$$

Iterative update of the network weights

$$a_{k+1} = a_k - \delta_k \frac{\partial E}{\partial a_k} \quad \wedge \quad b_{k+1} = b_k - \delta_k \frac{\partial E}{\partial b_k} \quad \text{with learning rate } \delta_k > 0$$

Example: $z = \tanh(ax)$ and $y = bz$

$$\frac{\partial E}{\partial a_k} = -2(y - \hat{y}) \frac{d\hat{y}}{dz} \frac{dz}{da_k} = -2(y - \hat{y})b_k [1 - \tanh^2(a_k x)] x$$

$$\frac{\partial E}{\partial b_k} = -2(y - \hat{y}) \frac{d\hat{y}}{db_k} = -2(y - \hat{y}) \tanh(a_k x)$$

Iterative update of the network weights

$$a_{k+1} = a_k - \delta_k \frac{\partial E}{\partial a_k} \quad \wedge \quad b_{k+1} = b_k - \delta_k \frac{\partial E}{\partial b_k} \quad \text{with learning rate } \delta_k > 0$$

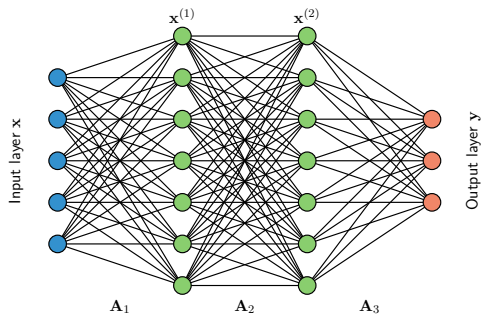
Example: $z = \tanh(ax)$ and $y = bz$

$$\frac{\partial E}{\partial a_k} = -2(y - \hat{y}) \frac{d\hat{y}}{dz} \frac{dz}{da_k} = -2(y - \hat{y})b_k \left[1 - \tanh^2(a_k x)\right] x$$

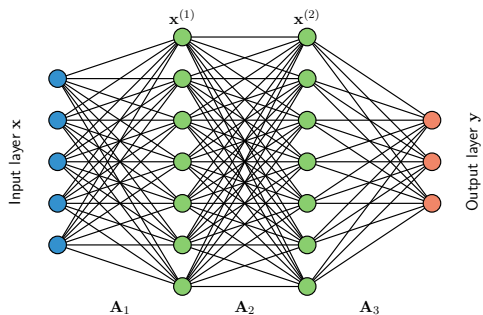
$$\frac{\partial E}{\partial b_k} = -2(y - \hat{y}) \frac{d\hat{y}}{db_k} = -2(y - \hat{y}) \tanh(a_k x)$$

We are not there yet. This is for *scalar* input x and output y and for a *single* input-output pair $(x; y)$. In practice we have multiple possibly vector-valued input-output pairs $(\mathbf{X}; \mathbf{Y})$.

ANN revisited



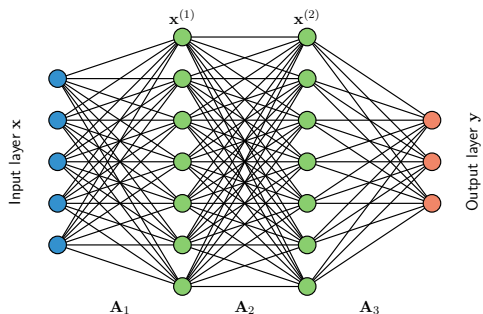
ANN revisited



Optimization problem

$$\operatorname{argmin}_{\mathbf{A}'_j} E(\mathbf{A}'_1, \mathbf{A}'_2, \mathbf{A}'_3),$$

ANN revisited



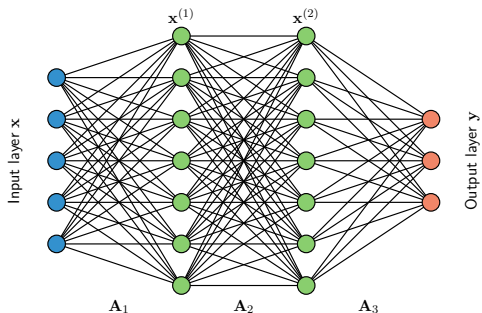
Optimization problem

$$\operatorname{argmin}_{\mathbf{A}'_j} E(\mathbf{A}'_1, \mathbf{A}'_2, \mathbf{A}'_3),$$

Minimum requirement

$$\frac{\partial E}{\partial (a_{ij})_l} = 0, \quad \forall (\mathbf{x}; \mathbf{y}) \text{ pairs}$$

ANN revisited



Optimization problem

$$\operatorname{argmin}_{\mathbf{A}'_j} E(\mathbf{A}'_1, \mathbf{A}'_2, \mathbf{A}'_3),$$

Minimum requirement

$$\frac{\partial E}{\partial (a_{ij})_l} = 0, \quad \forall (\mathbf{x}; \mathbf{y}) \text{ pairs}$$

Stochastic gradient descent algorithm with *a single randomly chosen* $(\mathbf{x}^\ell; \mathbf{y}^\ell)$ pair

$$\begin{bmatrix} \mathbf{A}_1 & \mathbf{A}_2 & \mathbf{A}_3 \end{bmatrix}_{k+1} = \begin{bmatrix} \mathbf{A}_1 & \mathbf{A}_2 & \mathbf{A}_3 \end{bmatrix}_k - \delta_k \nabla \left(f_3(\mathbf{A}_3, f_2(\mathbf{A}_2, f_1(\mathbf{A}_1, \mathbf{x}^\ell))) - \mathbf{y}^\ell \right)^2$$

Intermediate summary

- **Activation functions** enable nonlinear network responses
- **Backpropagation** yields the gradient of the error w.r.t. the network weights
- **Stochastic gradient descent algorithm** enables the efficient update of network weights

Intermediate summary

- **Activation functions** enable nonlinear network responses
- **Backpropagation** yields the gradient of the error w.r.t. the network weights
- **Stochastic gradient descent algorithm** enables the efficient update of network weights

Notes

- Backpropagation uses **algorithmic differentiation** (automated application of chain rule)
- More powerful optimization algorithms like **Adam** or **BFGS** are often used in practice

Intermediate summary

- **Activation functions** enable nonlinear network responses
- **Backpropagation** yields the gradient of the error w.r.t. the network weights
- **Stochastic gradient descent algorithm** enables the efficient update of network weights

Notes

- Backpropagation uses **algorithmic differentiation** (automated application of chain rule)
- More powerful optimization algorithms like **Adam** or **BFGS** are often used in practice
- Extra additive weights, **biases**, are often used in practice to offset responses, i.e.

$$\mathbf{y} = f(\mathbf{A}, \mathbf{b}, \mathbf{x}) = f(\mathbf{Ax} + \mathbf{b})$$

Intermediate summary

- **Activation functions** enable nonlinear network responses
- **Backpropagation** yields the gradient of the error w.r.t. the network weights
- **Stochastic gradient descent algorithm** enables the efficient update of network weights

Notes

- Backpropagation uses **algorithmic differentiation** (automated application of chain rule)
- More powerful optimization algorithms like **Adam** or **BFGS** are often used in practice
- Extra additive weights, **biases**, are often used in practice to offset responses, i.e.

$$\mathbf{y} = f(\mathbf{A}, \mathbf{b}, \mathbf{x}) = f(\mathbf{Ax} + \mathbf{b})$$

- Short hand notations

$$\mathbf{W} = (\mathbf{A}_1, \dots, \mathbf{A}_L, \mathbf{b}_1, \dots, \mathbf{b}_L), \quad \mathcal{N}(\mathbf{x}) = f_L(\mathbf{A}_L, \mathbf{b}_L, \dots, f_1(\mathbf{A}_1, \mathbf{b}_1, \mathbf{x}))$$

1 Introduction

Terminology

Deep learning 101

2 Physics-informed machine learning

Physics-informed neural networks

Deep Operator neural networks

3 IgANets

Concept and preliminary applications

Universal function approximator

Learning from

data

physics

given

$$(\mathbf{x}; \mathbf{y})$$

$$(\mathbf{x}; L(u(\mathbf{x})) = f)$$

predict

$$\hat{\mathbf{y}} = \mathcal{N}(\mathbf{x})$$

$$\hat{u} = \mathcal{N}(\mathbf{x})$$

compare

$$E = \|\mathbf{y} - \hat{\mathbf{y}}\|^2$$

$$E = \|L(\hat{u}) - f\|^2$$

update

$$\mathbf{W}_{k+1} = \mathbf{W}_k - \delta_k \nabla E$$

$$\mathbf{W}_{k+1} = \mathbf{W}_k - \delta_k \nabla E$$

Universal function approximator

Learning from

data

physics

given

$$(\mathbf{x}; \mathbf{y})$$

$$(\mathbf{x}; L(u(\mathbf{x})) = f)$$

predict

$$\hat{\mathbf{y}} = \mathcal{N}(\mathbf{x})$$

$$\hat{u} = \mathcal{N}(\mathbf{x})$$

compare

$$E = \|\mathbf{y} - \hat{\mathbf{y}}\|^2$$

$$E = \|L(\hat{u}) - f\|^2$$

update

$$\mathbf{W}_{k+1} = \mathbf{W}_k - \delta_k \nabla E$$

$$\mathbf{W}_{k+1} = \mathbf{W}_k - \delta_k \nabla E$$

Universal approximation theorem [Cybenko, 1989, Hornik et al., 1989]: For every continuous function $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ there exists an ANN that approximates it with given accuracy. *[This does not say that it is easy to find this ANN in practice.]*

Universal function approximator

Example: Poisson's equation in 1d

$$\begin{array}{l} u_{xx}(x) = f(x) \quad \text{in } (0, 1) \\ u(x) = g(x) \quad \text{at } 0, 1 \end{array} \quad \rightarrow \quad \begin{array}{l} E = \|\mathcal{N}_{xx}(x) - f(x)\|^2 \\ + \|\mathcal{N}(x) - g(x)\|^2 \end{array}$$

Compute derivatives w.r.t. to x by the same backpropagation algorithm as for the weights

Universal function approximator

Example: Poisson's equation in 1d

$$\begin{array}{l} u_{xx}(x) = f(x) \quad \text{in } (0, 1) \\ u(x) = g(x) \quad \text{at } 0, 1 \end{array} \quad \rightarrow \quad \begin{array}{l} E = \| \mathcal{N}_{xx}(x) - f(x) \|^2 \\ + \| \mathcal{N}(x) - g(x) \|^2 \end{array}$$

Compute derivatives w.r.t. to x by the same backpropagation algorithm as for the weights

History

- Neural networks for DEs [Dissanayake and Phan-Thien, 1994, Lagaris et al., 1998]

Universal function approximator

Example: Poisson's equation in 1d

$$\begin{array}{l} u_{xx}(x) = f(x) \quad \text{in } (0, 1) \\ u(x) = g(x) \quad \text{at } 0, 1 \end{array} \quad \rightarrow \quad \begin{array}{l} E = \| \mathcal{N}_{xx}(x) - f(x) \|^2 \\ + \| \mathcal{N}(x) - g(x) \|^2 \end{array}$$

Compute derivatives w.r.t. to x by the same backpropagation algorithm as for the weights

History

- Neural networks for DEs [Dissanayake and Phan-Thien, 1994, Lagaris et al., 1998]
- Rediscovery as [physics-informed neural networks](#) [Raissi et al., 2017a,b, 2019]

Universal function approximator

Example: Poisson's equation in 1d

$$\begin{aligned} u_{xx}(x) &= f(x) & \text{in } (0, 1) \\ u(x) &= g(x) & \text{at } 0, 1 \end{aligned} \quad \rightarrow \quad \begin{aligned} E &= \|\mathcal{N}_{xx}(x) - f(x)\|^2 \\ &+ \|\mathcal{N}(x) - g(x)\|^2 \end{aligned}$$

Compute derivatives w.r.t. to x by the same backpropagation algorithm as for the weights

History

- Neural networks for DEs [Dissanayake and Phan-Thien, 1994, Lagaris et al., 1998]
- Rediscovery as [physics-informed neural networks](#) [Raissi et al., 2017a,b, 2019]
- Since then applied to many problems (forward, inverse, ...), extended in various ways (distributed PINNs, adaptive PINNs, ...), and made available as easy-to-use software packages (DeepXDE, SciANN, NVIDIA Modulus, NeuroDiffEq, NeuralPDE, ...)

Examples of PINNs

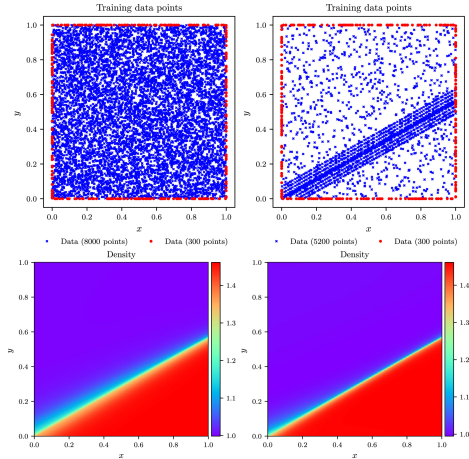


Figure from: *Physics-informed neural networks for high-speed flows* [Mao et al., 2020]

Examples of PINNs

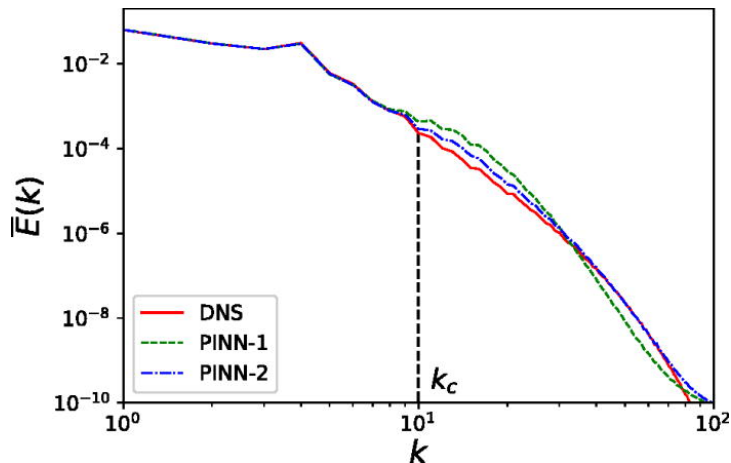


Figure from: *Physics-informed data based neural networks for two-dimensional turbulence*
[Kag et al., 2022]

Examples of PINNs

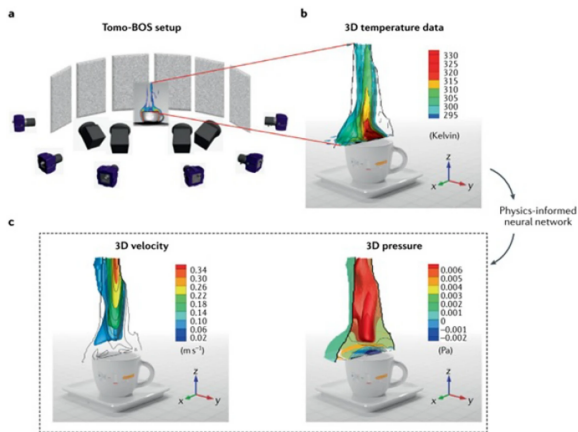
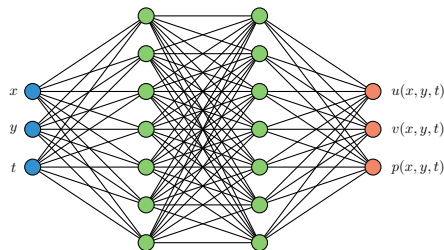


Figure from: *Physics-informed machine learning* [Karniadakis et al., 2021b]

Strengths of PINNs

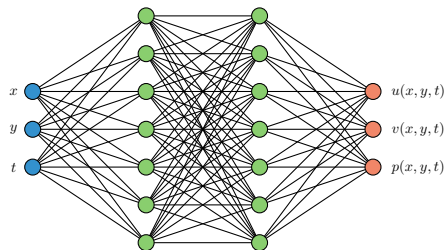
- ⊕ Easy to implement for 'any' physical model



$$E = MSE_{\text{NSE-}u}(u, v, p) + MSE_{\text{NSE-}v}(u, v, p) + MSE_{\text{NSE-}p}(u, v, p) \\ + MSE_{\text{BDR}}(u, v, p)$$

Strengths of PINNs

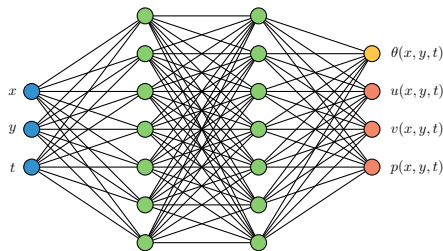
- ⊕ Easy to implement for 'any' physical model
- ⊕ Simulation/experimental data can be included



$$E = MSE_{NSE-u}(u, v, p) + MSE_{NSE-v}(u, v, p) + MSE_{NSE-p}(u, v, p) \\ + MSE_{BDR}(u, v, p) + \|u - u_{ref}\|^2 + \|v - v_{ref}\|^2 + \|p - p_{ref}\|^2$$

Strengths of PINNs

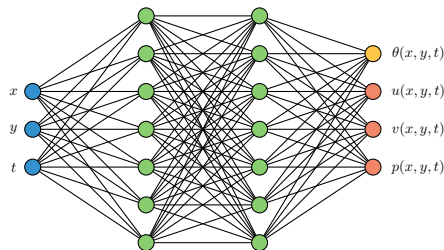
- ⊕ Easy to implement for 'any' physical model
- ⊕ Simulation/experimental data can be included
- ⊕ Easy prediction of derived quantities of interest or model parameters $\theta \rightarrow$ inverse problems



$$E = MSE_{NSE-u}(u^\theta, v^\theta, p^\theta) + MSE_{NSE-v}(u^\theta, v^\theta, p^\theta) + MSE_{NSE-p}(u^\theta, v^\theta, p^\theta) \\ + MSE_{BDR}(u^\theta, v^\theta, p^\theta) + \|u^\theta - u_{ref}^\theta\|^2 + \|v^\theta - v_{ref}^\theta\|^2 + \|p^\theta - p_{ref}^\theta\|^2$$

Strengths of PINNs

- ⊕ Easy to implement for 'any' physical model
- ⊕ Simulation/experimental data can be included
- ⊕ Easy prediction of derived quantities of interest or model parameters $\theta \rightarrow$ inverse problems
- ⊕ Flexibility to 'tune' network architecture to problem structure (e.g., separate \mathcal{N}_u , \mathcal{N}_v , \mathcal{N}_p)



$$E = MSE_{NSE-u}(u^\theta, v^\theta, p^\theta) + MSE_{NSE-v}(u^\theta, v^\theta, p^\theta) + MSE_{NSE-p}(u^\theta, v^\theta, p^\theta) \\ + MSE_{BDR}(u^\theta, v^\theta, p^\theta) + \|u^\theta - u_{\text{ref}}^\theta\|^2 + \|v^\theta - v_{\text{ref}}^\theta\|^2 + \|p^\theta - p_{\text{ref}}^\theta\|^2$$

Weaknesses of PINNs

- ⊖ Learn a single problem instance, i.e. PDE, bc's, geometry, ... are hard-coded in the loss function (unless problem parameters such as ν are defined as network inputs)

Weaknesses of PINNs

- ⊖ Learn a single problem instance, i.e. PDE, bc's, geometry, ... are hard-coded in the loss function (unless problem parameters such as ν are defined as network inputs)
- ⊖ Poor generalization capabilities, e.g., transient problems at $t > t_{\max}^{\text{train}}$

Weaknesses of PINNs

- ⊖ Learn a single problem instance, i.e. PDE, bc's, geometry, ... are hard-coded in the loss function (unless problem parameters such as ν are defined as network inputs)
- ⊖ Poor generalization capabilities, e.g., transient problems at $t > t_{\max}^{\text{train}}$
- ⊖ Point-wise output \rightarrow many forward evaluations needed to reconstruct solution fields

Weaknesses of PINNs

- ⊖ Learn a single problem instance, i.e. PDE, bc's, geometry, ... are hard-coded in the loss function (unless problem parameters such as ν are defined as network inputs)
- ⊖ Poor generalization capabilities, e.g., transient problems at $t > t_{\max}^{\text{train}}$
- ⊖ Point-wise output \rightarrow many forward evaluations needed to reconstruct solution fields
- ⊖ High computational costs during training especially in higher dimensions

Weaknesses of PINNs

- ⊖ Learn a single problem instance, i.e. PDE, bc's, geometry, ... are hard-coded in the loss function (unless problem parameters such as ν are defined as network inputs)
- ⊖ Poor generalization capabilities, e.g., transient problems at $t > t_{\max}^{\text{train}}$
- ⊖ Point-wise output \rightarrow many forward evaluations needed to reconstruct solution fields
- ⊖ High computational costs during training especially in higher dimensions
- ⊖ Hyperparameter tuning is an art

Weaknesses of PINNs

- ⊖ Learn a single problem instance, i.e. PDE, bc's, geometry, ... are hard-coded in the loss function (unless problem parameters such as ν are defined as network inputs)
- ⊖ Poor generalization capabilities, e.g., transient problems at $t > t_{\max}^{\text{train}}$
- ⊖ Point-wise output \rightarrow many forward evaluations needed to reconstruct solution fields
- ⊖ High computational costs during training especially in higher dimensions
- ⊖ Hyperparameter tuning is an art

Hint: Don't use PINNs as replacement for a single forward analysis. The high training costs must amortize over many queries, e.g., in inverse analysis or design optimization.

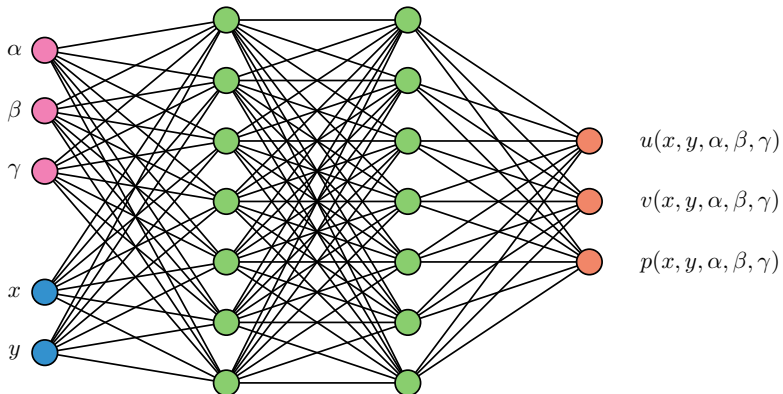
Weaknesses of PINNs

- ⊖ Learn a single problem instance, i.e. PDE, bc's, geometry, ... are hard-coded in the loss function (unless problem parameters such as ν are defined as network inputs)
- ⊖ Poor generalization capabilities, e.g., transient problems at $t > t_{\max}^{\text{train}}$
- ⊖ Point-wise output \rightarrow many forward evaluations needed to reconstruct solution fields
- ⊖ High computational costs during training especially in higher dimensions
- ⊖ Hyperparameter tuning is an art

Hint: Don't use PINNs as replacement for a single forward analysis. The high training costs must amortize over many queries, e.g., in inverse analysis or design optimization.

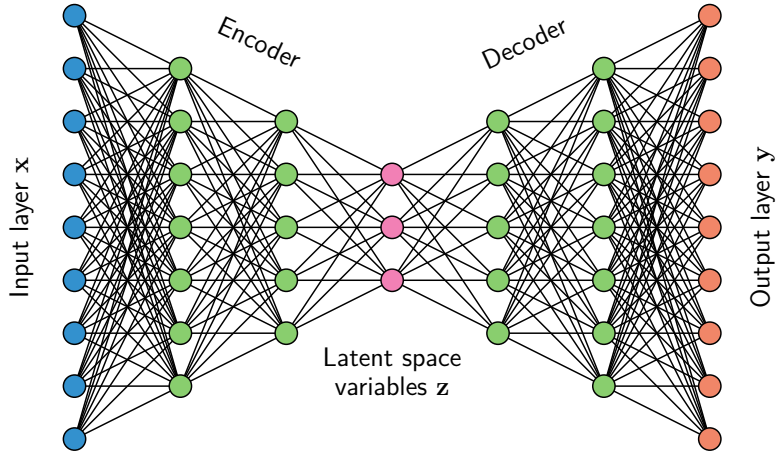
Challenge: How to feed 'designs' into the PINN framework?

Parameterized PINNs



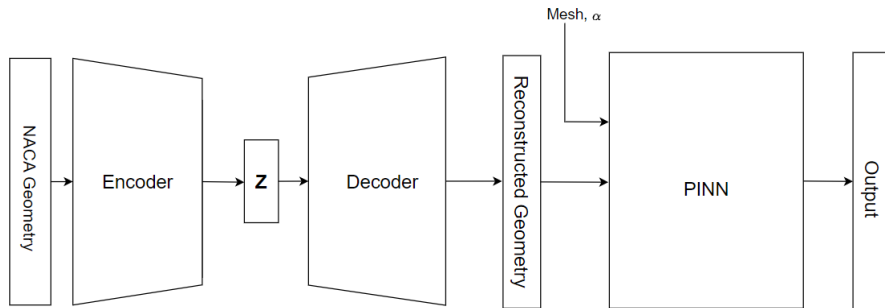
Note: Impractical if the number of design parameters is large (curse of dimensionality)

Auto encoders



Train network so that $x \equiv y$. Then separate the **encoder** from the **decoder** and use the latter to generate designs y from the latent space variables z serving as network inputs.

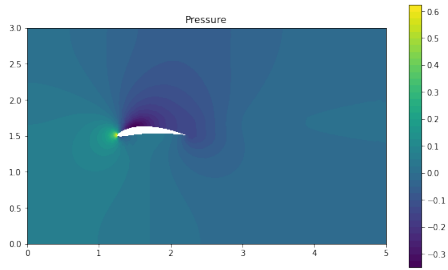
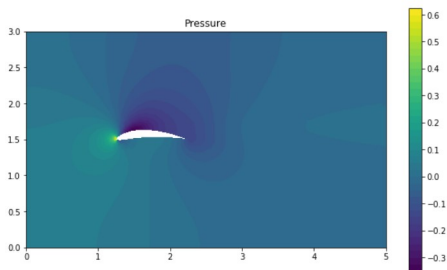
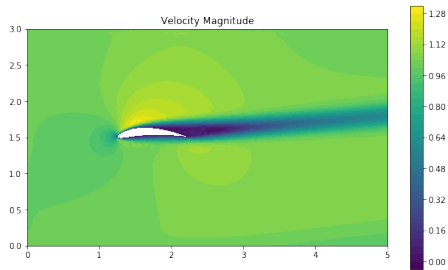
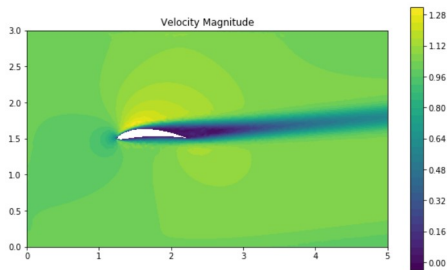
Latent space parameterized PINNs



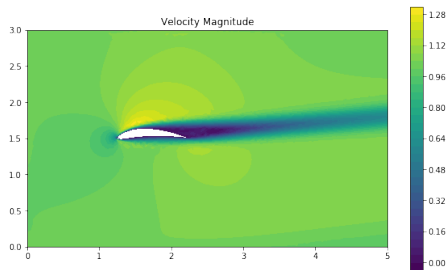
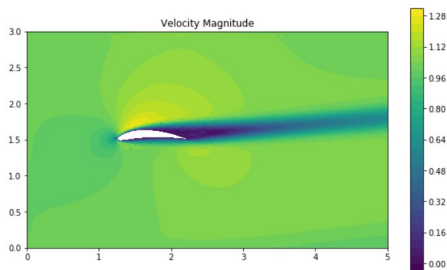
Physics-informed deep learning for computational fluid flow analysis [Kakkar, 2022]

Collaboration between TU Delft and Monolith AI Limited, UK

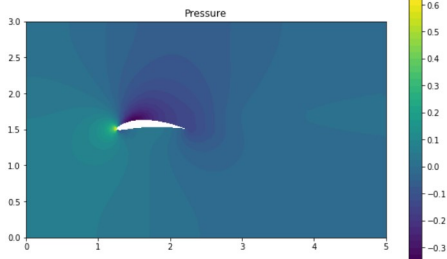
Latent space parameterized PINNs: NACA 4812 at 5 deg AoA



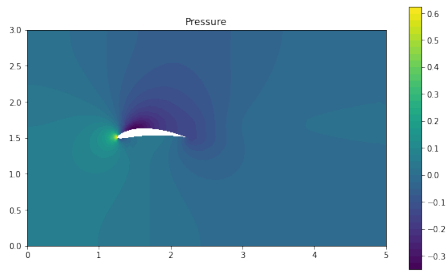
Latent space parameterized PINNs: NACA 4812 at 5 deg AoA



PINN



FEniCS



From function to operator learning

Function learning: learn the input-output relation of a function

$$f : x \in X \rightarrow Y$$

From function to operator learning

Function learning: learn the input-output relation of a function

$$f = \sin, \quad X = \mathbb{R}, \quad Y = [-1, 1] \quad \rightarrow \quad \text{learn } y = \sin(x)$$

From function to operator learning

Function learning: learn the input-output relation of a function

$$f = \sin, \quad X = \mathbb{R}, \quad Y = [-1, 1] \quad \rightarrow \quad \text{learn } y = \sin(x)$$

e.g., learn $\mathcal{N}_{\sin}(0) = 0$, $\mathcal{N}_{\sin}(\pi/2) = 1$, $\mathcal{N}_{\sin}(\pi) = 0$, \dots

From function to operator learning

Function learning: learn the **input-output relation of a function**

$$f = \sin, \quad X = \mathbb{R}, \quad Y = [-1, 1] \quad \rightarrow \quad \text{learn } y = \sin(x)$$

e.g., learn $\mathcal{N}_{\sin}(0) = 0$, $\mathcal{N}_{\sin}(\pi/2) = 1$, $\mathcal{N}_{\sin}(\pi) = 0$, ...

Operator learning¹: learn the **action of an operator on a function**, e.g.,

$$G : f \mapsto G(f), \quad G(f) : x \in X \rightarrow Y$$

¹Universal approximation theorem for operators [Chen and Chen, 1993]

From function to operator learning

Function learning: learn the **input-output relation of a function**

$$f = \sin, \quad X = \mathbb{R}, \quad Y = [-1, 1] \quad \rightarrow \quad \text{learn } y = \sin(x)$$

e.g., learn $\mathcal{N}_{\sin}(0) = 0, \quad \mathcal{N}_{\sin}(\pi/2) = 1, \quad \mathcal{N}_{\sin}(\pi) = 0, \quad \dots$

Operator learning¹: learn the **action of an operator on a function**, e.g.,

$$G = \frac{\partial \cdot}{\partial x}, \quad f = \sin \quad \rightarrow \quad \text{learn } y = G(f)(x) = \cos(x)$$

¹Universal approximation theorem for operators [Chen and Chen, 1993]

From function to operator learning

Function learning: learn the **input-output relation of a function**

$$f = \sin, \quad X = \mathbb{R}, \quad Y = [-1, 1] \quad \rightarrow \quad \text{learn } y = \sin(x)$$

e.g., learn $\mathcal{N}_{\sin}(0) = 0$, $\mathcal{N}_{\sin}(\pi/2) = 1$, $\mathcal{N}_{\sin}(\pi) = 0$, ...

Operator learning¹: learn the **action of an operator on a function**, e.g.,

$$G = \frac{\partial \cdot}{\partial x}, \quad f = \sin \quad \rightarrow \quad \text{learn } y = G(f)(x) = \cos(x)$$

e.g., learn $\mathcal{N}(\sin, 0) = 1$, $\mathcal{N}(\sin, \pi) = -1$, $\mathcal{N}(\cos, \pi/2) = -1, \dots$

¹Universal approximation theorem for operators [Chen and Chen, 1993]

Deep operator networks

DeepONets: *Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators* [Lu et al., 2021], many follow-up papers, e.g. *Physics-informed deep operator networks* [Goswami et al., 2022]

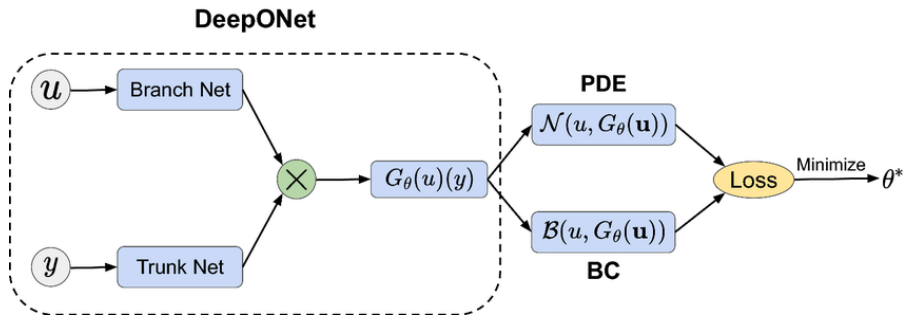


Figure from: *Improved architectures and training algorithms for deep operator networks* [Wang et al., 2022]

Simplified interpretation of DeepONets

Trunk network: learns a set of basis functions $(b_i(\mathbf{x}))_i$

Branch network: learns the basis coefficients $(c_i)_i$ (relative to this basis)

Cross product: combines basis coefficients and functions to produce solution values

$$u(\mathbf{x}) = \sum_i c_i b_i(\mathbf{x})$$

Simplified interpretation of DeepONets

Trunk network: learns a set of basis functions $(b_i(\mathbf{x}))_i$

Branch network: learns the basis coefficients $(c_i)_i$ (relative to this basis)

Cross product: combines basis coefficients and functions to produce solution values

$$u(\mathbf{x}) = \sum_i c_i b_i(\mathbf{x})$$

IgANets [Möller et al., 2021]: learn basis coefficients relative to a *fixed* B-spline basis

1 Introduction

Terminology

Deep learning 101

2 Physics-informed machine learning

Physics-informed neural networks

Deep Operator neural networks

3 IgANets

Concept and preliminary applications

B-spline basis functions

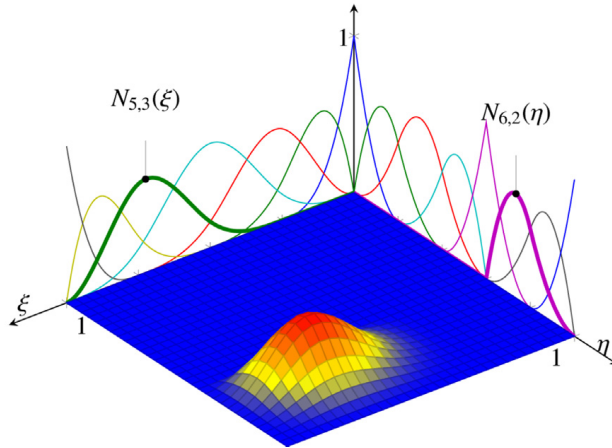


Figure from: *Geometrically nonlinear isogeometric analysis of laminated composite plates based on higher-order shear deformation theory* [Tran et al., 2015]

B-spline geometry models

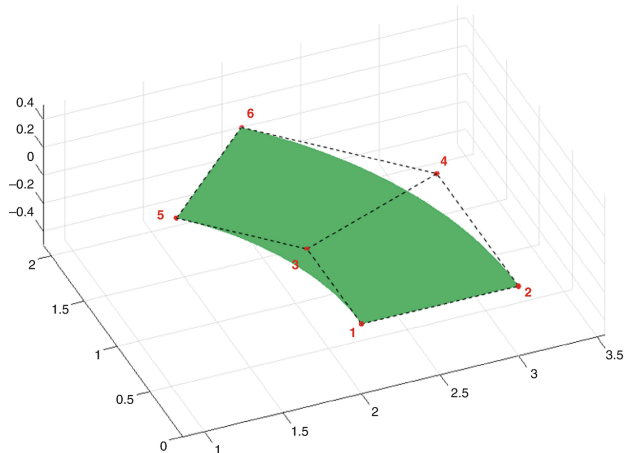


Figure from: *Reduced order isogeometric analysis approach for PDEs in parametrized domains* [Garotta et al., 2020]

(model problem)

$$-\Delta u_h = f_h \quad \text{in } \Omega_h \quad u_h = g_h \quad \text{on } \partial\Omega_h$$

(geometry)

$$\mathbf{x}_h(\xi, \eta) = \sum_{i=1}^n B_i(\xi, \eta) \cdot \mathbf{x}_i \quad \forall (\xi, \eta) \in [0, 1]^2$$

(load vector)

$$f_h \circ \mathbf{x}_h(\xi, \eta) = \sum_{i=1}^n B_i(\xi, \eta) \cdot \mathbf{f}_i \quad \forall (\xi, \eta) \in [0, 1]^2$$

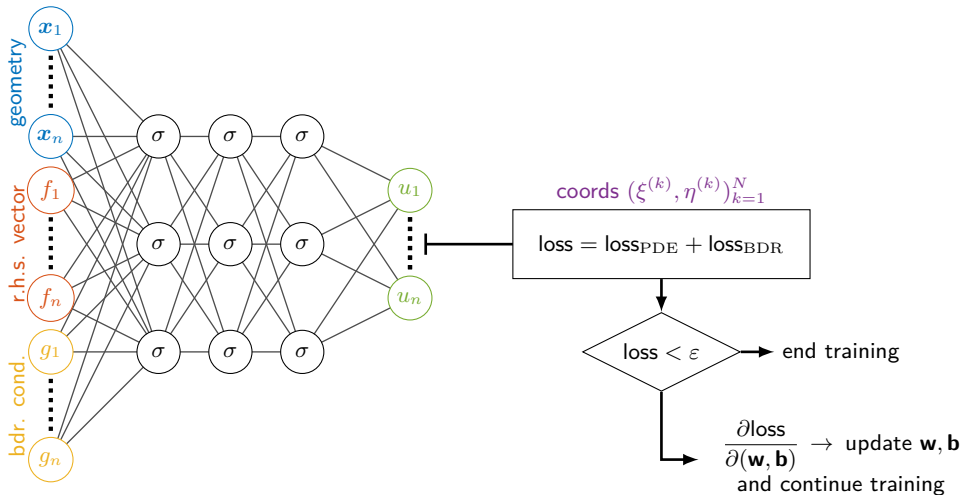
(boundary conditions)

$$g_h \circ \mathbf{x}_h(\xi, \eta) = \sum_{i=1}^n B_i(\xi, \eta) \cdot \mathbf{g}_i \quad \forall (\xi, \eta) \in \partial[0, 1]^2$$

(solution)

$$u_h \circ \mathbf{x}_h(\xi, \eta) = \sum_{i=1}^n B_i(\xi, \eta) \cdot u_i \quad \forall (\xi, \eta) \in [0, 1]^2$$

IgANet architecture



Loss function

$$\text{loss} = \text{loss}_{\text{PDE}} + \text{loss}_{\text{BDR}}$$

$$\text{loss}_{\text{PDE}} = \frac{1}{N_{\Omega}} \sum_{k=1}^{N_{\Omega}} \left| \Delta \left[u_h \circ \mathbf{x}_h \left(\xi^{(k)}, \eta^{(k)} \right) \right] - f_h \circ \mathbf{x}_h \left(\xi^{(k)}, \eta^{(k)} \right) \right|^2$$

$$\text{loss}_{\text{BDR}} = \frac{1}{N_{\Gamma}} \sum_{k=1}^{N_{\Gamma}} \left| u_h \circ \mathbf{x}_h \left(\xi^{(k)}, \eta^{(k)} \right) - g_h \circ \mathbf{x}_h \left(\xi^{(k)}, \eta^{(k)} \right) \right|^2$$

Two-level training strategy

For $[\mathbf{x}_1, \dots, \mathbf{x}_n] \in \mathcal{S}_{\text{geo}}$, $[f_1, \dots, f_n] \in \mathcal{S}_{\text{rhs}}$, $[g_1, \dots, g_n] \in \mathcal{S}_{\text{bcond}}$ **do**

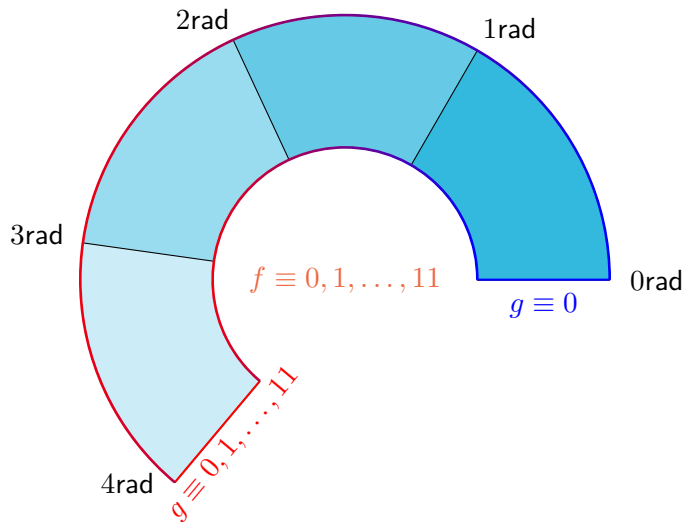
For batch of Greville abscissae $(\xi_k, \eta_k) \in [0, 1]^2$ **do**

$$\text{Train IgANet} \left(\left(\begin{bmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_n \end{bmatrix}, \begin{bmatrix} f_1 \\ \vdots \\ f_n \end{bmatrix}, \begin{bmatrix} g_1 \\ \vdots \\ g_n \end{bmatrix} ; (\xi_k, \eta_k)_{k=1}^{N_{\text{samples}}} \right) \mapsto \begin{bmatrix} u_1 \\ \vdots \\ u_n \end{bmatrix} \right)$$

EndFor

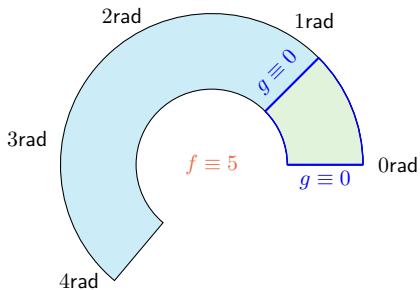
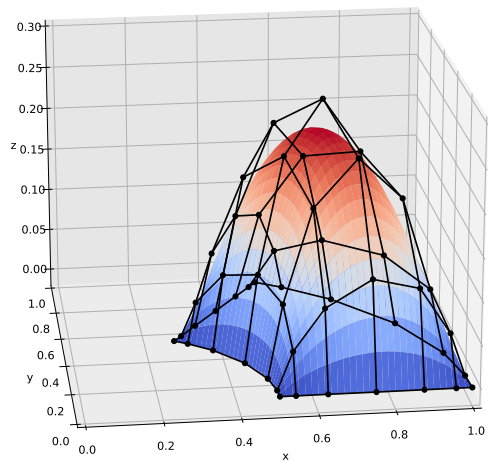
EndFor

Test case: Poisson's equation on a variable annulus



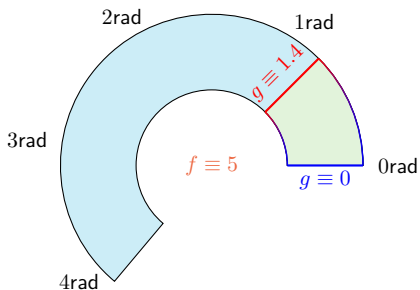
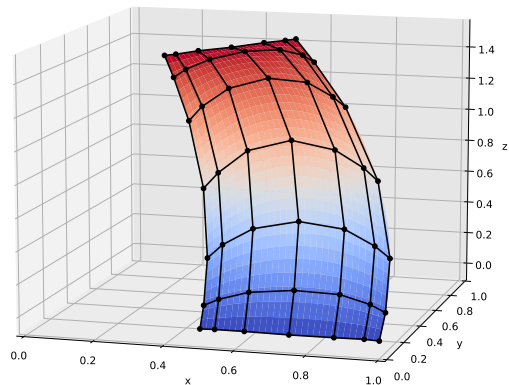
Figures from: PINNs for parametrized problems [van Ruiten, 2022]

Preliminary results



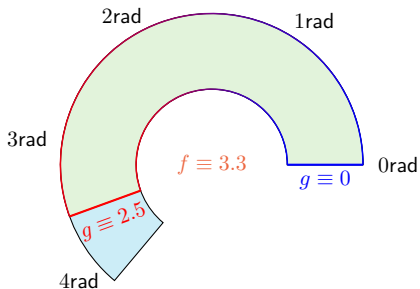
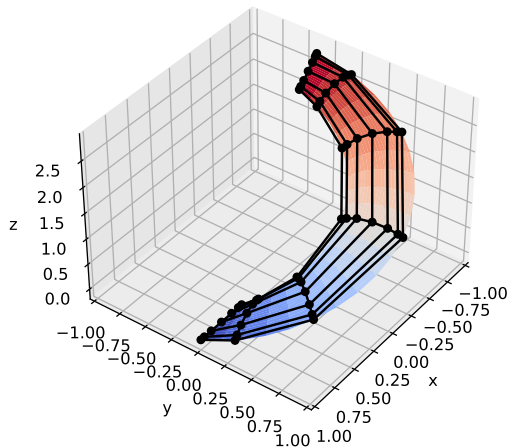
Figures from: PINNs for parametrized problems [van Ruiten, 2022]

Preliminary results



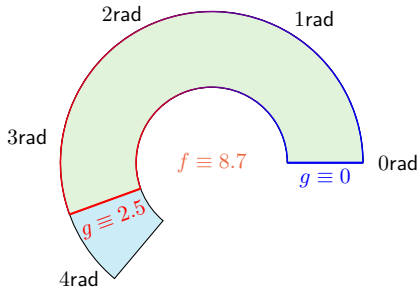
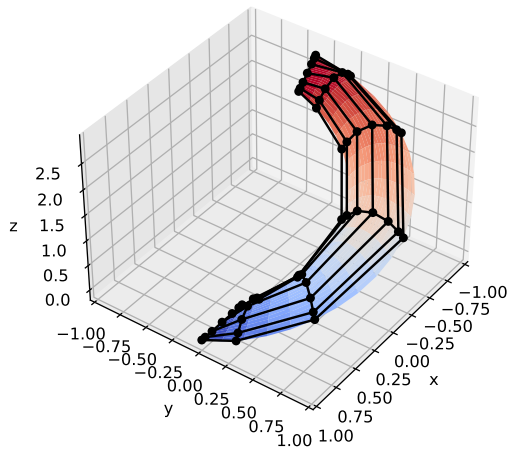
Figures from: PINNs for parametrized problems [van Ruiten, 2022]

Preliminary results



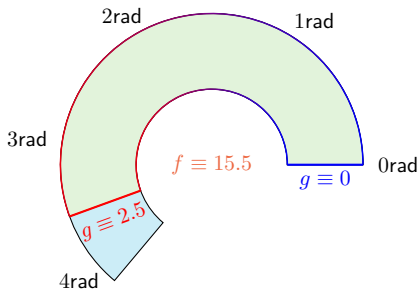
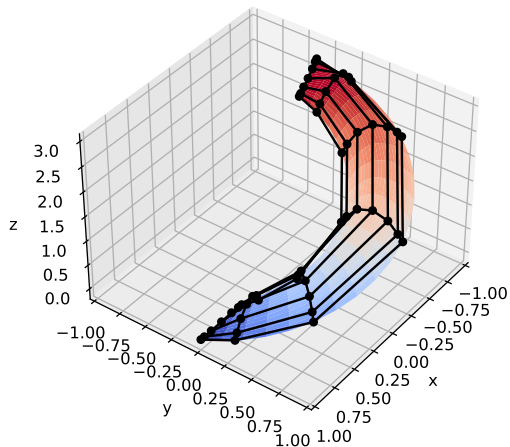
Figures from: PINNs for parametrized problems [van Ruiten, 2022]

Preliminary results



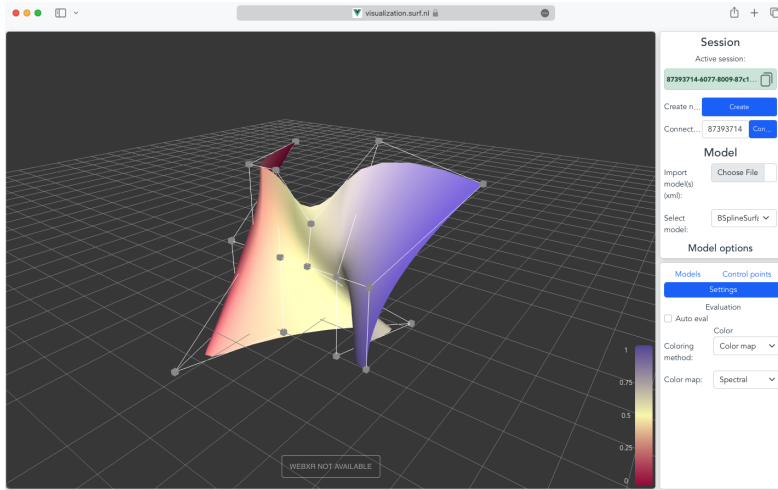
Figures from: PINNs for parametrized problems [van Ruiten, 2022]

Preliminary results



Figures from: PINNs for parametrized problems [van Ruiten, 2022]

Future application: interactive design-through-analysis



Collaboration with Scientific Visualization team at SURF NL

Physics-informed machine learning –
Hype or new trend in computational engineering?

Physics-informed machine learning – Hype or new trend in computational engineering?

- FEM [Courant, 1943], FVM [McDonald, 1971], Neural operators for differential equations [Dissanayake and Phan-Thien, 1994] ... just give it some more time

Physics-informed machine learning – Hype or new trend in computational engineering?

- FEM [Courant, 1943], FVM [McDonald, 1971], Neural operators for differential equations [Dissanayake and Phan-Thien, 1994] ... just give it some more time
- PIML will not replace FEM/FVM as simulation tool but enable new analyses (inverse modelling, quick/qualitative pre-design exploration, process monitoring/steering, ...)

Physics-informed machine learning – Hype or new trend in computational engineering?

- FEM [Courant, 1943], FVM [McDonald, 1971], Neural operators for differential equations [Dissanayake and Phan-Thien, 1994] ... just give it some more time
- PIML will not replace FEM/FVM as simulation tool but enable new analyses (inverse modelling, quick/qualitative pre-design exploration, process monitoring/steering, ...)

Call to action

- SciML is not the enemy nor a panacea. Explore its capabilities (and limitations) and combine it with today's tools where it brings added value to your workflow.

Physics-informed machine learning – Hype or new trend in computational engineering?

Matthias Möller

Department of Applied Mathematics, TU Delft, The Netherlands

6th Short Course & Forum on Computational Fluid Dynamics in Rotary Positive Displacement Machines

City, University of London, 10-09-2023

References I

- N. Baker, F. Alexander, T. Bremer, A. Hagberg, Y. Kevrekidis, H. Najm, M. Parashar, A. Patra, J. Sethian, S. Wild, K. Willcox, and S. Lee. Workshop report on basic research needs for scientific machine learning: Core technologies for artificial intelligence. Technical report, Feb. 2019. URL <https://doi.org/10.2172/1478744>.
- T. Chen and H. Chen. Approximations of continuous functionals by neural networks with application to dynamic systems. *IEEE Transactions on Neural Networks*, 4(6):910–918, 1993. doi: 10.1109/72.286886. URL <https://doi.org/10.1109/72.286886>.
- F. Courant. Variational method for the solution of problems of equilibrium and vibration, 1943.
- G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2(4):303–314, Dec. 1989. doi: 10.1007/bf02551274. URL <https://doi.org/10.1007/bf02551274>.

References II

- M. W. M. G. Dissanayake and N. Phan-Thien. Neural-network-based approximations for solving partial differential equations. *Communications in Numerical Methods in Engineering*, 10(3):195–201, Mar. 1994. doi: 10.1002/cnm.1640100303. URL <https://doi.org/10.1002/cnm.1640100303>.
- F. Garotta, N. Demo, M. Tezzele, M. Carraturo, A. Reali, and G. Rozza. Reduced order isogeometric analysis approach for PDEs in parametrized domains. In *Lecture Notes in Computational Science and Engineering*, pages 153–170. Springer International Publishing, 2020. doi: 10.1007/978-3-030-48721-8_7. URL https://doi.org/10.1007/978-3-030-48721-8_7.
- S. Goswami, A. Bora, Y. Yu, and G. E. Karniadakis. Physics-informed deep neural operator networks, 2022. URL <https://arxiv.org/abs/2207.05748>.
- Z. Hao, S. Liu, Y. Zhang, C. Ying, Y. Feng, H. Su, and J. Zhu. Physics-informed machine learning: A survey on problems, methods and applications, 2022. URL <https://arxiv.org/abs/2211.08064>.

References III

- K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, Jan. 1989. doi: 10.1016/0893-6080(89)90020-8. URL [https://doi.org/10.1016/0893-6080\(89\)90020-8](https://doi.org/10.1016/0893-6080(89)90020-8).
- V. Kag, K. Seshasayanan, and V. Gopinath. Physics-informed data based neural networks for two-dimensional turbulence. *Physics of Fluids*, 34(5), May 2022. doi: 10.1063/5.0090050. URL <https://doi.org/10.1063/5.0090050>.
- S. Kakkar. Physics-informed deep learning for computational fluid flow analysis: Coupling of physics-informed neural networks and autoencoders for aerodynamic flow predictions on variable geometries, 2022.
- G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, may 2021a. doi: 10.1038/s42254-021-00314-5. URL <https://doi.org/10.1038/s42254-021-00314-5>.

References IV

- G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, May 2021b. doi: 10.1038/s42254-021-00314-5. URL <https://doi.org/10.1038/s42254-021-00314-5>.
- A. Karpatne, G. Atluri, J. H. Faghmous, M. Steinbach, A. Banerjee, A. Ganguly, S. Shekhar, N. Samatova, and V. Kumar. Theory-guided data science: A new paradigm for scientific discovery from data. *IEEE Transactions on Knowledge and Data Engineering*, 29(10):2318–2331, oct 2017. doi: 10.1109/tkde.2017.2720168. URL <https://doi.org/10.1109/tkde.2017.2720168>.
- I. Lagaris, A. Likas, and D. Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks*, 9(5):987–1000, 1998. doi: 10.1109/72.712178. URL <https://doi.org/10.1109/72.712178>.

References V

- L. Lu, P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, Mar. 2021. doi: 10.1038/s42256-021-00302-5. URL <https://doi.org/10.1038/s42256-021-00302-5>.
- Z. Mao, A. D. Jagtap, and G. E. Karniadakis. Physics-informed neural networks for high-speed flows. *Computer Methods in Applied Mechanics and Engineering*, 360:112789, Mar. 2020. doi: 10.1016/j.cma.2019.112789. URL <https://doi.org/10.1016/j.cma.2019.112789>.
- P. W. Mcdonald. The computation of transonic flow through two-dimensional gas turbine cascades. 1971. URL <https://api.semanticscholar.org/CorpusID:121134844>.
- M. Möller, D. Toshniwal, and F. van Ruiten. Physics-informed machine learning embedded into isogeometric analysis. In *Mathematics: Key enabling technology for scientific machine learning*. Platform Wiskunde Nederland, 2021.

References VI

- C. Rackauckas, Y. Ma, J. Martensen, C. Warner, K. Zubov, R. Supekar, D. Skinner, and A. J. Ramadhan. Universal differential equations for scientific machine learning. *CoRR*, abs/2001.04385, 2020. URL <https://arxiv.org/abs/2001.04385>.
- M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations. *arXiv preprint arXiv:1711.10561*, 2017a.
- M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics informed deep learning (part ii): Data-driven discovery of nonlinear partial differential equations. *arXiv preprint arXiv:1711.10566*, 2017b.
- M. Raissi, P. Perdikaris, and G. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, feb 2019. doi: 10.1016/j.jcp.2018.10.045. URL <https://doi.org/10.1016%2Fj.jcp.2018.10.045>.
- N. Thuerey, P. Holl, M. Mueller, P. Schnell, F. Trost, and K. Um. Physics-based deep learning, 2021. URL <https://physicsbaseddeeplearning.org>.

References VII

- L. V. Tran, J. Lee, H. Nguyen-Van, H. Nguyen-Xuan, and M. A. Wahab. Geometrically nonlinear isogeometric analysis of laminated composite plates based on higher-order shear deformation theory. *International Journal of Non-Linear Mechanics*, 72:42–52, June 2015. doi: 10.1016/j.ijnonlinmec.2015.02.007. URL <https://doi.org/10.1016/j.ijnonlinmec.2015.02.007>.
- F. van Ruiten. Pinns for parametrized problems, 2022.
- S. Wang, H. Wang, and P. Perdikaris. Improved architectures and training algorithms for deep operator networks. *Journal of Scientific Computing*, 92(2), June 2022. doi: 10.1007/s10915-022-01881-0. URL <https://doi.org/10.1007/s10915-022-01881-0>.