An introduction to deep-learning-based methods for optimization and control of PDEs

Francisco Periago

Universidad Politécnica de Cartagena
https://multisimo.com/fpe/
https://github.com/fperiago/

XXI Jacques-Louis Lions Hispano-French School on Numerical Simulation in Physics and Engineering

Ciudad Real, july, 7-11, 2025

Outline of the course

- **Part I: Machine Learning basis.** Where and why Machine-Learning-based methods may be useful in the numerical approximation of PDEs-based models?

Outline of the course

- **Part I: Machine Learning basis.** Where and why Machine-Learning-based methods may be useful in the numerical approximation of PDEs-based models?
- **Part II: Physics Informed Neural Networks (PINNs)** A machine-learning-based method for solving direct, inverse, control, etc., PDEs-based problems for fixed initial and/or boundary conditions

Outline of the course

- **Part I: Machine Learning basis.** Where and why Machine-Learning-based methods may be useful in the numerical approximation of PDEs-based models?
- **Part II: Physics Informed Neural Networks (PINNs)** A machine-learning-based method for solving direct, inverse, control, etc., PDEs-based problems for fixed initial and/or boundary conditions
- **Part III: Deep Operator Network (DeepONet).** Learning the solution map of the above problems

Outline of the course

- **Part I: Machine Learning basis.** Where and why Machine-Learning-based methods may be useful in the numerical approximation of PDEs-based models?
- **Part II: Physics Informed Neural Networks (PINNs)** A machine-learning-based method for solving direct, inverse, control, etc., PDEs-based problems for fixed initial and/or boundary conditions
- **Part III: Deep Operator Network (DeepONet).** Learning the solution map of the above problems
- **Part IV: Numerical implementation via DeepXDE.** A Python library for scientific machine learning and physics-informed learning

Part I

Machine Learning Basis

Introduction: the set up of supervised learning

Introduction: the set up of supervised learning

Main Goal:

Introduction: the set up of supervised learning

Main Goal: approximate (as accurately as we can) an unknown function $f^* : \mathbb{R}^d \to \mathbb{R}^N$ from a dataset $S = \{(\boldsymbol{x}_i, \boldsymbol{y}_i = f^*(\boldsymbol{x}_i)), 1 \leq i \leq n\}$

Introduction: the set up of supervised learning

Main Goal: approximate (as accurately as we can) an unknown function $f^* : \mathbb{R}^d \to \mathbb{R}^N$ from a dataset $S = \{(\boldsymbol{x}_i, \boldsymbol{y}_i = f^*(\boldsymbol{x}_i)), 1 \leq i \leq n\}$
Two cases:

Introduction: the set up of supervised learning

Main Goal: approximate (as accurately as we can) an unknown function $f^* : \mathbb{R}^d \to \mathbb{R}^N$ from a dataset $S = \{(\boldsymbol{x}_i, \boldsymbol{y}_i = f^*(\boldsymbol{x}_i)), 1 \leq i \leq n\}$

Two cases:

1. regression: f^* takes continuous values, and

Introduction: the set up of supervised learning

Main Goal: approximate (as accurately as we can) an unknown function $f^* : \mathbb{R}^d \to \mathbb{R}^N$ from a dataset $S = \{(\boldsymbol{x}_i, \boldsymbol{y}_i = f^*(\boldsymbol{x}_i)), 1 \leq i \leq n\}$

Two cases:

1. regression: f^* takes continuous values, and
2. classification: f^* takes discrete values.

Introduction: the set up of supervised learning

Main Goal: approximate (as accurately as we can) an unknown function $f^* : \mathbb{R}^d \to \mathbb{R}^N$ from a dataset $S = \{(\boldsymbol{x}_i, \boldsymbol{y}_i = f^*(\boldsymbol{x}_i)), 1 \leq i \leq n\}$

Two cases:

1. regression: f^* takes continuous values, and
2. classification: f^* takes discrete values.

Standard procedure for supervised learning(regression)

Introduction: the set up of supervised learning

Main Goal: approximate (as accurately as we can) an unknown function $f^*: \mathbb{R}^d \to \mathbb{R}^N$ from a dataset $S = \{(\boldsymbol{x}_i, \boldsymbol{y}_i = f^*(\boldsymbol{x}_i)), 1 \leq i \leq n\}$
Two cases:

1. regression: f^* takes continuous values, and
2. classification: f^* takes discrete values.

Standard procedure for supervised learning(regression)

1. Choose a hypothesis space $\mathcal{H}_m$.

Introduction: the set up of supervised learning

Main Goal: approximate (as accurately as we can) an unknown function $f^* : \mathbb{R}^d \to \mathbb{R}^N$ from a dataset $S = \{(\boldsymbol{x}_i, \boldsymbol{y}_i = f^*(\boldsymbol{x}_i)), 1 \leq i \leq n\}$

Two cases:

1. regression: f^* takes continuous values, and
2. classification: f^* takes discrete values.

Standard procedure for supervised learning(regression)

1. Choose a hypothesis space $\mathcal{H}_m$. *Artificial neural networks* is the model of choice in Machine Learning.

Introduction: the set up of supervised learning

Main Goal: approximate (as accurately as we can) an unknown function $f^*: \mathbb{R}^d \to \mathbb{R}^N$ from a dataset $S = \{(\boldsymbol{x}_i, \boldsymbol{y}_i = f^*(\boldsymbol{x}_i)), 1 \le i \le n\}$

Two cases:

1. regression: f^* takes continuous values, and
2. classification: f^* takes discrete values.

Standard procedure for supervised learning(regression)

1. Choose a hypothesis space $\mathcal{H}_m$. *Artificial neural networks* is the model of choice in Machine Learning.
2. Choose a loss function. If we are interested in fitting the data, a popular choice is the so-called **training error**

$$\hat{\mathcal{R}}_n(f) = \frac{1}{n}\sum_{i=1}^{n}\left(f(\boldsymbol{\theta}; \boldsymbol{x}_i) - f^*(\boldsymbol{x}_i)\right)^2, \quad f \in \mathcal{H}_m. \tag{1}$$

Introduction: the set up of supervised learning

Main Goal: approximate (as accurately as we can) an unknown function $f^* : \mathbb{R}^d \to \mathbb{R}^N$ from a dataset $S = \{(\boldsymbol{x}_i, \boldsymbol{y}_i = f^*(\boldsymbol{x}_i)), 1 \leq i \leq n\}$
Two cases:

1. regression: f^* takes continuous values, and
2. classification: f^* takes discrete values.

Standard procedure for supervised learning(regression)

1. Choose a hypothesis space $\mathcal{H}_m$. *Artificial neural networks* is the model of choice in Machine Learning.
2. Choose a loss function. If we are interested in fitting the data, a popular choice is the so-called **training error**

$$\hat{\mathcal{R}}_n(f) = \frac{1}{n} \sum_{i=1}^{n} \left(f(\boldsymbol{\theta}; \boldsymbol{x}_i) - f^*(\boldsymbol{x}_i) \right)^2, \quad f \in \mathcal{H}_m. \tag{1}$$

3. Choose an optimization algorithm for computing the optimal parameters $\boldsymbol{\theta}$ that minimize the loss function.

Introduction: the set up of supervised learning

Main Goal: approximate (as accurately as we can) an unknown function $f^* : \mathbb{R}^d \to \mathbb{R}^N$ from a dataset $S = \{(\boldsymbol{x}_i, \boldsymbol{y}_i = f^*(\boldsymbol{x}_i)), 1 \leq i \leq n\}$

Two cases:

1. regression: f^* takes continuous values, and
2. classification: f^* takes discrete values.

Standard procedure for supervised learning(regression)

1. Choose a hypothesis space $\mathcal{H}_m$. *Artificial neural networks* is the model of choice in Machine Learning.
2. Choose a loss function. If we are interested in fitting the data, a popular choice is the so-called **training error**

$$\hat{\mathcal{R}}_n(f) = \frac{1}{n} \sum_{i=1}^{n} (f(\boldsymbol{\theta}; \boldsymbol{x}_i) - f^*(\boldsymbol{x}_i))^2, \quad f \in \mathcal{H}_m. \tag{1}$$

3. Choose an optimization algorithm for computing the optimal parameters $\boldsymbol{\theta}$ that minimize the loss function.

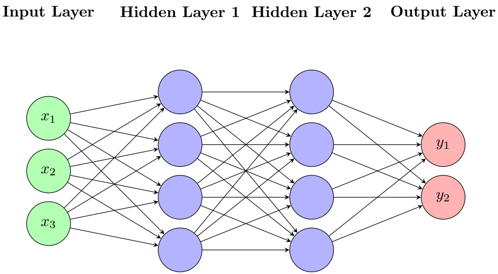
The overall objective is to minimize the **generalization error**

$$\mathcal{R}(f) = \mathbb{E}_{\boldsymbol{x} \sim \mathbb{P}} (f(\boldsymbol{\theta}; \boldsymbol{x}_i) - f^*(\boldsymbol{x}_i))^2, \quad f \in \mathcal{H}_m, \tag{2}$$

with $\mathbb{P}$ the (unknown) distribution of $\boldsymbol{x}$.

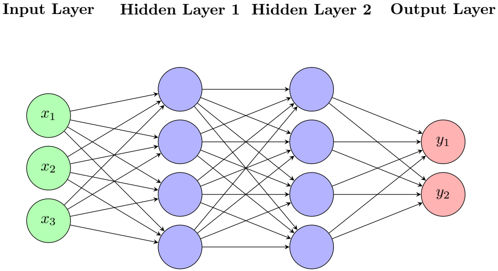
1. Hypothesis space: an example

A canonical example of an hypothesis space $\mathcal{H}_m$ (or a neural network architecture) is the so-called **Multi-Layer Perceptron (MLP)**.



1. Hypothesis space: an example

A canonical example of an hypothesis space $\mathcal{H}_m$ (or a neural network architecture) is the so-called **Multi-Layer Perceptron (MLP)**.

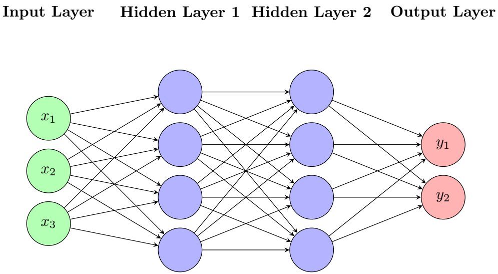


To each input $\boldsymbol{x} \in \mathbb{R}^d$ it associates the output $\boldsymbol{y} = f_m(\boldsymbol{x}) := \boldsymbol{x}^m$ defined by

$$\begin{cases} \boldsymbol{x}^{k+1} = \sigma\left(\omega^k \boldsymbol{x}^k + b^k\right) \quad \text{for } k = 0, 1, \cdots, m-1 \\ \boldsymbol{x}^0 = \boldsymbol{x}, \end{cases} \tag{3}$$

1. Hypothesis space: an example

A canonical example of an hypothesis space $\mathcal{H}_m$ (or a neural network architecture) is the so-called **Multi-Layer Perceptron (MLP)**.



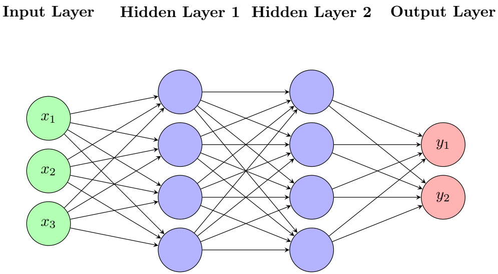
To each input $\boldsymbol{x} \in \mathbb{R}^d$ it associates the output $\boldsymbol{y} = f_m(\boldsymbol{x}) := \boldsymbol{x}^m$ defined by

$$\begin{cases} \boldsymbol{x}^{k+1} = \sigma\left(\omega^k \boldsymbol{x}^k + b^k\right) & \text{for } k = 0, 1, \cdots, m-1 \\ \boldsymbol{x}^0 = \boldsymbol{x}, & \end{cases} \tag{3}$$

or in compositional form $\boldsymbol{x}^m = \left(\sigma \circ \Lambda^{m-1} \circ \cdots \circ \sigma \circ \Lambda^0\right)(\boldsymbol{x})$, $\Lambda^k \boldsymbol{x} = \omega^k \boldsymbol{x} + b^k$,

1. Hypothesis space: an example

A canonical example of an hypothesis space $\mathcal{H}_m$ (or a neural network architecture) is the so-called **Multi-Layer Perceptron (MLP)**.



To each input $\boldsymbol{x} \in \mathbb{R}^d$ it associates the output $\boldsymbol{y} = f_m(\boldsymbol{x}) := \boldsymbol{x}^m$ defined by

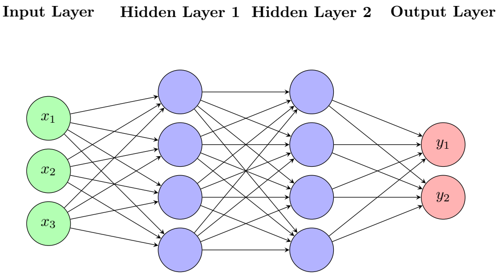
$$\begin{cases} \boldsymbol{x}^{k+1} = \sigma\left(\omega^k \boldsymbol{x}^k + b^k\right) & \text{for } k = 0, 1, \cdots, m-1 \\ \boldsymbol{x}^0 = \boldsymbol{x}, \end{cases} \tag{3}$$

or in compositional form $\boldsymbol{x}^m = \left(\sigma \circ \Lambda^{m-1} \circ \cdots \circ \sigma \circ \Lambda^0\right)(\boldsymbol{x})$, $\Lambda^k \boldsymbol{x} = \omega^k \boldsymbol{x} + b^k$,

- optimizable parameters $\boldsymbol{\theta}$: **weights** $\omega^k \in \mathbb{R}^{d_{k+1} \times d_k}$ and **biases** $b^k \in \mathbb{R}^{d_k}$

1. Hypothesis space: an example

A canonical example of an hypothesis space $\mathcal{H}_m$ (or a neural network architecture) is the so-called **Multi-Layer Perceptron (MLP)**.



To each input $\boldsymbol{x} \in \mathbb{R}^d$ it associates the output $\boldsymbol{y} = f_m(\boldsymbol{x}) := \boldsymbol{x}^m$ defined by

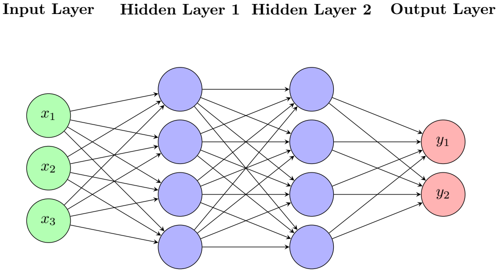
$$\begin{cases} \boldsymbol{x}^{k+1} = \sigma\left(\omega^k \boldsymbol{x}^k + b^k\right) & \text{for } k = 0, 1, \cdots, m-1 \\ \boldsymbol{x}^0 = \boldsymbol{x}, \end{cases} \tag{3}$$

or in compositional form $\boldsymbol{x}^m = \left(\sigma \circ \Lambda^{m-1} \circ \cdots \circ \sigma \circ \Lambda^0\right)(\boldsymbol{x})$, $\Lambda^k \boldsymbol{x} = \omega^k \boldsymbol{x} + b^k$,

- optimizable parameters $\boldsymbol{\theta}$: **weights** $\omega^k \in \mathbb{R}^{d_{k+1} \times d_k}$ and **biases** $b^k \in \mathbb{R}^{d_k}$
- m is the **depth** of the neural network,

1. Hypothesis space: an example

A canonical example of an hypothesis space $\mathcal{H}_m$ (or a neural network architecture) is the so-called **Multi-Layer Perceptron (MLP)**.



To each input $\boldsymbol{x} \in \mathbb{R}^d$ it associates the output $\boldsymbol{y} = f_m(\boldsymbol{x}) := \boldsymbol{x}^m$ defined by

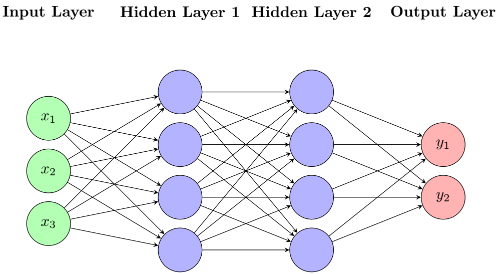
$$\begin{cases} \boldsymbol{x}^{k+1} = \sigma\left(\omega^k \boldsymbol{x}^k + b^k\right) \quad \text{for } k = 0, 1, \cdots, m-1 \\ \boldsymbol{x}^0 = \boldsymbol{x}, \end{cases} \tag{3}$$

or in compositional form $\boldsymbol{x}^m = \left(\sigma \circ \Lambda^{m-1} \circ \cdots \circ \sigma \circ \Lambda^0\right)(\boldsymbol{x})$, $\Lambda^k \boldsymbol{x} = \omega^k \boldsymbol{x} + b^k$,

- optimizable parameters $\boldsymbol{\theta}$: **weights** $\omega^k \in \mathbb{R}^{d_{k+1} \times d_k}$ and **biases** $b^k \in \mathbb{R}^{d_k}$
- m is the **depth** of the neural network,
- for any k, the vector $\boldsymbol{x}^k \in \mathbb{R}^{d_k}$ and d_k is the **width** of the layer k,

1. Hypothesis space: an example

A canonical example of an hypothesis space $\mathcal{H}_m$ (or a neural network architecture) is the so-called **Multi-Layer Perceptron (MLP)**.



To each input $\boldsymbol{x} \in \mathbb{R}^d$ it associates the output $\boldsymbol{y} = f_m(\boldsymbol{x}) := \boldsymbol{x}^m$ defined by

$$\begin{cases} \boldsymbol{x}^{k+1} = \sigma\left(\omega^k \boldsymbol{x}^k + b^k\right) & \text{for } k = 0, 1, \cdots, m-1 \\ \boldsymbol{x}^0 = \boldsymbol{x}, & \end{cases} \tag{3}$$

or in compositional form $\boldsymbol{x}^m = \left(\sigma \circ \Lambda^{m-1} \circ \cdots \circ \sigma \circ \Lambda^0\right)(\boldsymbol{x})$, $\Lambda^k \boldsymbol{x} = \omega^k \boldsymbol{x} + b^k$,

- optimizable parameters $\boldsymbol{\theta}$: **weights** $\omega^k \in \mathbb{R}^{d_{k+1} \times d_k}$ and **biases** $b^k \in \mathbb{R}^{d_k}$
- m is the **depth** of the neural network,
- for any k, the vector $\boldsymbol{x}^k \in \mathbb{R}^{d_k}$ and d_k is the **width** of the layer k,
- σ is a fixed nonlinear **activation function**

1. Hypothesis space: an example

More on the **activation function**:

1. Hypothesis space: an example

More on the **activation function:**
By abuse of notation, $\sigma : \mathbb{R}^d \to \mathbb{R}^d$ is defined component-wise by

$$\sigma(\boldsymbol{x})_j := \sigma(\boldsymbol{x}_j), \quad 1 \leq j \leq d.$$

1. Hypothesis space: an example

More on the **activation function**:
By abuse of notation, $\sigma : \mathbb{R}^d \to \mathbb{R}^d$ is defined component-wise by

$$\sigma(\boldsymbol{x})_j := \sigma(\boldsymbol{x}_j), \quad 1 \leq j \leq d.$$

Common choices include *rectifiers* such as ReLU: $\sigma(s) = \max\{s, 0\}$, and *sigmoids* such as $\sigma(s) = \tanh(s)$.

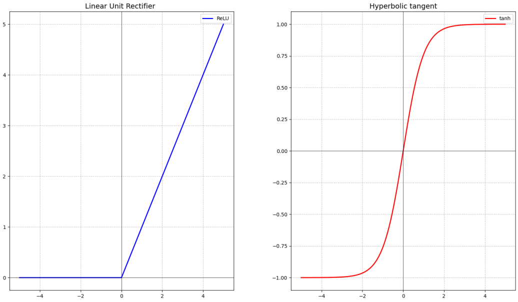


Figure: Linear Unit Rectifier (left) and hyperbolic tangent (right).

Important parameters to keep in mind

Important parameters to keep in mind

- m: **number of free parameters**

Important parameters to keep in mind

- m: number of free parameters
- n: size of the training dataset

Important parameters to keep in mind

- m: **number of free parameters**
- n: **size of the training dataset**
- t: **number of training steps**

Important parameters to keep in mind

- m: **number of free parameters**
- n: **size of the training dataset**
- t: **number of training steps**
- d: **input dimension**

Important parameters to keep in mind

- m**: number of free parameters**
- n**: size of the training dataset**
- t**: number of training steps**
- d**: input dimension**

Typically, $m, n, t \to \infty$ and $d >> 1$.

Important parameters to keep in mind

- m: **number of free parameters**
- n: **size of the training dataset**
- t: **number of training steps**
- d: **input dimension**

Typically, $m, n, t \to \infty$ and $d >> 1$.

Examples where d is large include:

Important parameters to keep in mind

- m: **number of free parameters**
- n: **size of the training dataset**
- t: **number of training steps**
- d: **input dimension**

Typically, $m, n, t \to \infty$ and $d >> 1$.

Examples where d is large include:

- radiactive transport equation ($d \geq 5$)

Important parameters to keep in mind

- m: **number of free parameters**
- n: **size of the training dataset**
- t: **number of training steps**
- d: **input dimension**

Typically, $m, n, t \to \infty$ and $d >> 1$.

Examples where d is large include:

- radiactive transport equation ($d \geq 5$)
- Boltzmann kinetic equations ($d = 6$)

Important parameters to keep in mind

- m: **number of free parameters**
- n: **size of the training dataset**
- t: **number of training steps**
- d: **input dimension**

Typically, $m, n, t \to \infty$ and $d >> 1$.

Examples where d is large include:

- radiactive transport equation ($d \geq 5$)
- Boltzmann kinetic equations ($d = 6$)
- nonlinear Schrödinger equation in the quantum many-body problem ($d \gg 1$)

Important parameters to keep in mind

- m: **number of free parameters**
- n: **size of the training dataset**
- t: **number of training steps**
- d: **input dimension**

Typically, $m, n, t \to \infty$ and $d >> 1$.

Examples where d is large include:

- radiactive transport equation ($d \geq 5$)
- Boltzmann kinetic equations ($d = 6$)
- nonlinear Schrödinger equation in the quantum many-body problem ($d \gg 1$)
- parameter-dependent (random) PDEs

Important parameters to keep in mind

- m: **number of free parameters**
- n: **size of the training dataset**
- t: **number of training steps**
- d: **input dimension**

Typically, $m, n, t \to \infty$ and $d >> 1$.

Examples where d is large include:

- radiactive transport equation ($d \geq 5$)
- Boltzmann kinetic equations ($d = 6$)
- nonlinear Schrödinger equation in the quantum many-body problem ($d \gg 1$)
- parameter-dependent (random) PDEs
- nonlinear Black- Scholes equation for pricing derivatives

Deep Learning opens a door to deal with real-world control problems

More situations that lead to very large d:

- turbulence modeling,
- plasticity models,
- multiscale,
- multiphysics,
- etc.

Deep Learning opens a door to deal with real-world control problems

More situations that lead to very large d:

- turbulence modeling,
- plasticity models,
- multiscale,
- multiphysics,
- etc.

The heart of the matter for the difficulties described above is our limited ability to handle functions of many variables, and this is exactly where **machine learning** *can make a difference.*

Weinan E. The dawning of a new era in applied mathematics , Notice of the AMS, 2021.

https://web.math.princeton.edu/~weinan/

Deep Learning opens a door to deal with real-world control problems

More situations that lead to very large d:

- turbulence modeling,
- plasticity models,
- multiscale,
- multiphysics,
- etc.

The heart of the matter for the difficulties described above is our limited ability to handle functions of many variables, and this is exactly where **machine learning** *can make a difference.*
Weinan E. The dawning of a new era in applied mathematics , Notice of the AMS, 2021.

https://web.math.princeton.edu/~weinan/

Machine learning is a promising tool to deal with high-dimensional problems

Additionally, ...

The methods of this course are (very) easy to implement for a large class of mathematical models. Therefore, it may help you to

- test new models very quickly

Additionally, ...

The methods of this course are (very) easy to implement for a large class of mathematical models. Therefore, it may help you to

- test new models very quickly
- incorporate experimental data in your models

Additionally, ...

The methods of this course are (very) easy to implement for a large class of mathematical models. Therefore, it may help you to

- test new models very quickly
- incorporate experimental data in your models
- have very quick predictions on your models (maybe not so accurate)

Part II

Physics Informed Neural Networks (PINNs)

Main objectives and references

Goals

Main objectives and references

Goals

- Study the basis of PINNs algorithm for solving a variety of PDEs-based mathematical models

Main objectives and references

Goals

- Study the basis of PINNs algorithm for solving a variety of PDEs-based mathematical models
- Gain some computational skills on PINNs -based software

Main objectives and references

Goals

- Study the basis of PINNs algorithm for solving a variety of PDEs-based mathematical models
- Gain some computational skills on PINNs -based software

Main objectives and references

Goals

- Study the basis of PINNs algorithm for solving a variety of PDEs-based mathematical models
- Gain some computational skills on PINNs -based software

References

- Raisi, M., Perdikaris, P. and Karniadakis, G: Physics-informed neural networks: *A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations*, J. Comput. Physics **378**, 686-707, 2019.
- S Wang, S Sankaran, H Wang, P Perdikaris:*An expert's guide to training physics-informed neural networks.* arXiv:2308.08468, 178, 2023.
- Lu, L., Meng, X., Mao, Z. and Karniadakis, G.:*DeepXDE: A Deep Learning Library for Solving Differential Equations*, SIAM Review, **63** (1), 208-228, 2021.
- García-Cervera, C.J.; Kessler, M.; Periago, F.: *Control of partial differential equations via physics-informed neural networks.* J. Optim. Theory Appl. **196** (2), 391-414, 2023.

A toy model: null control of the wave equation

$$\begin{cases} y_{tt} - \Delta y = 0, & \text{in } Q_T \\ y(x,0) = y^0(x), & \text{in } \Omega \\ y_t(x,0) = y^1(x) & \text{in } \Omega \\ y(x,t) = 0, & \text{on } \Gamma_D \times (0,T) \\ y(x,t) = u(x,t) & \text{on } \Gamma_C \times (0,T) \end{cases}$$

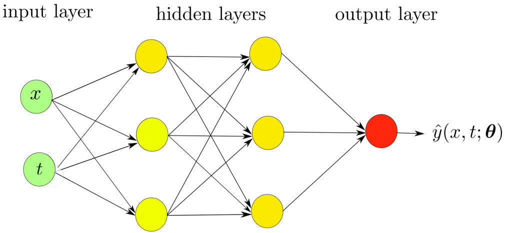
Goal: Compute $u(x,t)$ such that

$$y(x,T) = y_t(x,T) = 0 \quad x \in \Omega.$$

Numerical approximation using PINNs

Step 1: Neural network

A surrogate $\hat{y}(x, t; \boldsymbol{\theta})$ of the state variable $y(x, t)$ is constructed as



$$\begin{cases} \text{input layer:} & \mathcal{N}^0(\boldsymbol{x}) = \boldsymbol{x} = (x, t) \in \mathbb{R}^{d+1} \\ \text{hidden layers:} & \mathcal{N}^\ell(\boldsymbol{x}) = \sigma\left(\boldsymbol{W}^\ell \mathcal{N}^{\ell-1}(\boldsymbol{x}) + \boldsymbol{b}^\ell\right) \in \mathbb{R}^{N_\ell}, \quad \ell = 1, \cdots, L-1 \\ \text{output layer:} & \hat{y}\left(\boldsymbol{x}; \boldsymbol{\theta}\right) = \mathcal{N}^L(\boldsymbol{x}) = \boldsymbol{W}^L \mathcal{N}^{L-1}(\boldsymbol{x}) + \boldsymbol{b}^L \in \mathbb{R} \end{cases}$$

- $\mathcal{N}^\ell : \mathbb{R}^{d_{in}} \to \mathbb{R}^{d_{out}}$ is the ℓ layer with N_ℓ neurons,
- $\boldsymbol{W}^\ell \in \mathbb{R}^{N_\ell \times N_{\ell-1}}$ and $\boldsymbol{b}^\ell \in \mathbb{R}^{N_\ell}$ are, respectively, the weights and biases so that $\boldsymbol{\theta} = \left\{\boldsymbol{W}^\ell, \boldsymbol{b}^\ell\right\}_{1 \le \ell \le L}$ are the parameters of the neural network, and
- σ is an activation function, e.g. $\sigma(s) = \tanh(s)$

Numerical approximation using PINNs

Step 2: Training dataset

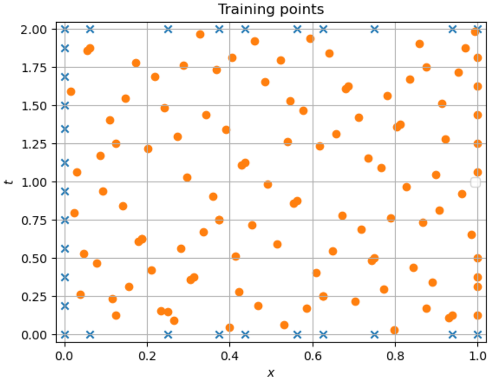


Figure: Illustration of a training dataset (based on Sobol points) in the domain $Q_2 = (0,1) \times (0,2)$. Interior points are marked with circles and boundary points in blue color. $\mathbf{x}_j = (x_j, t_j)$ are the features.

Numerical approximation using PINNs

Step 3: **Loss function.** **Labels equal zero**

$$
\begin{aligned}
\mathcal{L}_{\text{int}}\left(\boldsymbol{\theta}; \mathcal{T}_{\text{int}}\right) &= \textstyle\sum_{j=1}^{N_{\text{int}}} w_{j,\text{int}} |\hat{y}_{tt}(\boldsymbol{x}_j; \boldsymbol{\theta}) - \Delta \hat{y}(\boldsymbol{x}_j; \boldsymbol{\theta})|^2, && \boldsymbol{x}_j \in \mathcal{T}_{\text{int}} \\
\mathcal{L}_{\Gamma_D}\left(\boldsymbol{\theta}; \mathcal{T}_{\Gamma_D}\right) &= \textstyle\sum_{j=1}^{N_b} w_{j,b} |\hat{y}(\boldsymbol{x}_j; \boldsymbol{\theta})|^2, && \boldsymbol{x}_j \in \mathcal{T}_{\Gamma_D} \\
\mathcal{L}^{\text{pos}}_{t=0}\left(\boldsymbol{\theta}; \mathcal{T}_{t=0}\right) &= \textstyle\sum_{j=1}^{N_0} w_{j,0} |\hat{y}(\boldsymbol{x}_j; \boldsymbol{\theta}) - y^0(\boldsymbol{x}_j)|^2, && \boldsymbol{x}_j \in \mathcal{T}_{t=0} \\
\mathcal{L}^{\text{vel}}_{t=0}\left(\boldsymbol{\theta}; \mathcal{T}_{t=0}\right) &= \textstyle\sum_{j=1}^{N_0} w_{j,0} |\hat{y}_t(\boldsymbol{x}_j; \boldsymbol{\theta}) - y^1(\boldsymbol{x}_j)|^2, && \boldsymbol{x}_j \in \mathcal{T}_{t=0} \\
\mathcal{L}^{\text{pos}}_{t=T}\left(\boldsymbol{\theta}; \mathcal{T}_{t=T}\right) &= \textstyle\sum_{j=1}^{N_T} w_{j,T} |\hat{y}(\boldsymbol{x}_j; \boldsymbol{\theta})|^2, && \boldsymbol{x}_j \in \mathcal{T}_{t=T} \\
\mathcal{L}^{\text{vel}}_{t=T}\left(\boldsymbol{\theta}; \mathcal{T}_{t=T}\right) &= \textstyle\sum_{j=1}^{N_T} w_{j,T} |\hat{y}_t(\boldsymbol{x}_j; \boldsymbol{\theta})|^2, && \boldsymbol{x}_j \in \mathcal{T}_{t=T},
\end{aligned}
$$

where $w_{j,\text{int}}$, $w_{j,b}$, $w_{j,0}$ and $w_{j,T}$ are the weights of suitable quadrature rules.

$$
\begin{aligned}
\mathcal{L}\left(\boldsymbol{\theta}; \mathcal{T}\right) &= \lambda_1 \mathcal{L}_{\text{int}}\left(\boldsymbol{\theta}; \mathcal{T}_{\text{int}}\right) \\
&+ \lambda_2 \mathcal{L}_{\Gamma_D}\left(\boldsymbol{\theta}; \mathcal{T}_{\Gamma_D}\right) \\
&+ \lambda_3 \mathcal{L}^{\text{pos}}_{t=0}\left(\boldsymbol{\theta}; \mathcal{T}_{t=0}\right) + \lambda_4 \mathcal{L}^{\text{vel}}_{t=0}\left(\boldsymbol{\theta}; \mathcal{T}_{t=0}\right) \\
&+ \lambda_5 \mathcal{L}^{\text{pos}}_{t=T}\left(\boldsymbol{\theta}; \mathcal{T}_{t=T}\right) + \lambda_6 \mathcal{L}^{\text{vel}}_{t=T}\left(\boldsymbol{\theta}; \mathcal{T}_{t=T}\right).
\end{aligned}
$$

Numerical approximation using PINNs

Step 4: Training process

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathcal{L}\left(\boldsymbol{\theta}; \mathcal{T}\right).$$

The approximation $\hat{u}(t; \boldsymbol{\theta}^*)$ of the control $u(x, t)$ is

$$\hat{u}(x, t; \boldsymbol{\theta}^*) = \hat{y}(x, t; \boldsymbol{\theta}^*), \quad x \in \Gamma_C,\ 0 \leq t \leq T.$$

Numerical approximation using PINNs

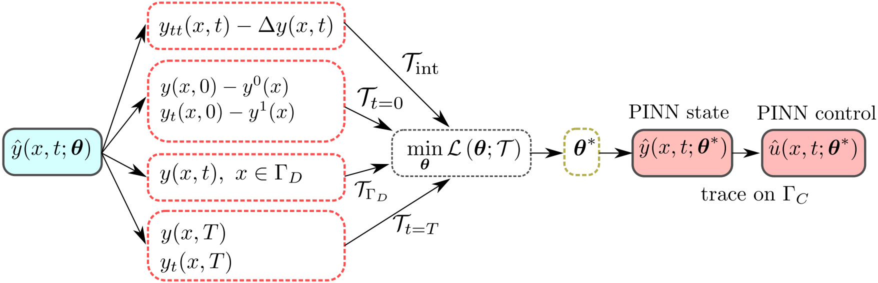
Step 4: Training process

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathcal{L}\left(\boldsymbol{\theta}; \mathcal{T}\right).$$

The approximation $\hat{u}(t; \boldsymbol{\theta}^*)$ of the control $u(x, t)$ is

$$\hat{u}(x, t; \boldsymbol{\theta}^*) = \hat{y}(x, t; \boldsymbol{\theta}^*), \quad x \in \Gamma_C,\ 0 \leq t \leq T.$$

To sump up:



Approximation theory and convergence analysis

Why MLP is a suitable prediction model ?

Approximation theory and convergence analysis

Why MLP is a suitable prediction model ?

Let us consider the hypothesis space of single-layer neural nets

$$\mathcal{H}_m := \left\{ y_m(\boldsymbol{x}) := \sum_{i=1}^{m} a_i \sigma\left(\boldsymbol{\omega}_i \boldsymbol{x} + b_i\right) : \boldsymbol{x}, \boldsymbol{\omega}_i \in \mathbb{R}^{d+1}, a_i, b_i \in \mathbb{R} \right\}.$$

Approximation theory and convergence analysis

Why MLP is a suitable prediction model ?

Let us consider the hypothesis space of single-layer neural nets

$$\mathcal{H}_m := \left\{ y_m(\boldsymbol{x}) := \sum_{i=1}^{m} a_i \sigma\left(\boldsymbol{\omega}_i \boldsymbol{x} + b_i\right) : \boldsymbol{x}, \boldsymbol{\omega}_i \in \mathbb{R}^{d+1}, a_i, b_i \in \mathbb{R} \right\}.$$

Theorem (Pinkus Universal Approximation Theorem)

Let $f \in C^k(\mathbb{R}^{d+1})$. Assume that the activation function $\sigma \in C^k(\mathbb{R})$ is not a polynomial. Then, for any compact set $K \subset \mathbb{R}^{d+1}$ and any $\varepsilon > 0$ there exists $m \in \mathbb{N}$ and $y_m \in \mathcal{H}_m$ such that

$$\max_{\boldsymbol{x} \in K} \left| D^{\ell} f(\boldsymbol{x}) - D^{\ell} y_m(\boldsymbol{x}) \right| \leq \varepsilon$$

for all multiindex $\ell \leq k$. Moreover, each $a_i = a_i(f)$ is a continuous linear functional defined on K.

Pinkus, A.: *Approximation theory of the MLP model in neural networks* **Acta numer. 8**, 143-195, 1999.

Why is so crucial that the activation function not to be a polynomial?

Approximation theory and convergence analysis

Why is so crucial that the activation function not to be a polynomial?

If σ is a polynomial of degree m, then $\sigma(\omega x + b)$ is a polynomial of total degree at most m. Thus, $\mathcal{H}_m$ is the space of all polynomials of degree m, which is not dense in $C(K)$, $K \subset \mathbb{R}$ compact.

Why is so crucial that the activation function not to be a polynomial?

If σ is a polynomial of degree m, then $\sigma\left(\omega x+b\right)$ is a polynomial of total degree at most m. Thus, $\mathcal{H}_m$ is the space of all polynomials of degree m, which is not dense in $C(K)$, $K\subset\mathbb{R}$ compact.
Conversely, let $\sigma\in C^{\infty}(\mathbb{R})$. Consider the space

$$\mathcal{N}\left(\sigma;\mathbb{R},\mathbb{R}\right)=\mathrm{span}\left\{\sigma(wx+b),\quad w,b\in\mathbb{R}\right\}.$$

Why is so crucial that the activation function not to be a polynomial?

If σ is a polynomial of degree m, then $\sigma(\omega x + b)$ is a polynomial of total degree at most m. Thus, $\mathcal{H}_m$ is the space of all polynomials of degree m, which is not dense in $C(K)$, $K \subset \mathbb{R}$ compact.
Conversely, let $\sigma \in C^\infty(\mathbb{R})$. Consider the space

$$\mathcal{N}(\sigma; \mathbb{R}, \mathbb{R}) = \operatorname{span}\{\sigma(wx+b), \quad w, b \in \mathbb{R}\}.$$

A classical result in Calculus states that if $\sigma \in C^\infty$ on any open interval and is not a polynomial thereon, then there exists a point $b^\star$ in that interval such that $\sigma^{(k)}(b^\star) \neq 0$ for all $k = 0, 1, 2, \cdots$.

Approximation theory and convergence analysis

Why is so crucial that the activation function not to be a polynomial?

If σ is a polynomial of degree m, then $\sigma(\omega x + b)$ is a polynomial of total degree at most m. Thus, $\mathcal{H}_m$ is the space of all polynomials of degree m, which is not dense in $C(K)$, $K \subset \mathbb{R}$ compact.
Conversely, let $\sigma \in C^\infty(\mathbb{R})$. Consider the space

$$\mathcal{N}(\sigma; \mathbb{R}, \mathbb{R}) = \operatorname{span}\{\sigma(wx + b), \quad w, b \in \mathbb{R}\}.$$

A classical result in Calculus states that if $\sigma \in C^\infty$ on any open interval and is not a polynomial thereon, then there exists a point $b^\star$ in that interval such that $\sigma^{(k)}(b^\star) \neq 0$ for all $k = 0, 1, 2, \cdots$. For $h \neq 0$,

$$\frac{\sigma((w + h)x + b^\star) - \sigma(wx + b^\star)}{h} \in \mathcal{N}(\sigma; \mathbb{R}, \mathbb{R})$$

and so

$$\frac{d}{dw}\sigma(wx + b^\star)|_{w=0} = x\sigma'(b^\star) \in \overline{\mathcal{N}(\sigma; \mathbb{R}, \mathbb{R})}.$$

Approximation theory and convergence analysis

Why is so crucial that the activation function not to be a polynomial?

If σ is a polynomial of degree m, then $\sigma\left(\omega x+b\right)$ is a polynomial of total degree at most m. Thus, $\mathcal{H}_m$ is the space of all polynomials of degree m, which is not dense in $C(K)$, $K\subset\mathbb{R}$ compact.
Conversely, let $\sigma\in C^{\infty}(\mathbb{R})$. Consider the space

$$\mathcal{N}\left(\sigma;\mathbb{R},\mathbb{R}\right)=\operatorname{span}\left\{\sigma(wx+b),\quad w,b\in\mathbb{R}\right\}.$$

A classical result in Calculus states that if $\sigma\in C^{\infty}$ on any open interval and is not a polynomial thereon, then there exists a point $b^{\star}$ in that interval such that $\sigma^{(k)}(b^{\star})\neq 0$ for all $k=0,1,2,\cdots$. For $h\neq 0$,

$$\frac{\sigma\left((w+h)x+b^{\star}\right)-\sigma\left(wx+b^{\star}\right)}{h}\in\mathcal{N}\left(\sigma;\mathbb{R},\mathbb{R}\right)$$

and so

$$\frac{d}{dw}\sigma\left(wx+b^{\star}\right)|_{w=0}=x\sigma'(b^{\star})\in\overline{\mathcal{N}\left(\sigma;\mathbb{R},\mathbb{R}\right)}.$$

Analogously,

$$\frac{d^k}{dw^k}\sigma\left(wx+b^{\star}\right)|_{w=0}=x^k\sigma^k(b^{\star})\in\overline{\mathcal{N}\left(\sigma;\mathbb{R},\mathbb{R}\right)}.$$

Why is so crucial that the activation function not to be a polynomial?

If σ is a polynomial of degree m, then $\sigma\left(\omega x+b\right)$ is a polynomial of total degree at most m. Thus, $\mathcal{H}_m$ is the space of all polynomials of degree m, which is not dense in $C(K)$, $K\subset\mathbb{R}$ compact.
Conversely, let $\sigma\in C^{\infty}(\mathbb{R})$. Consider the space

$$\mathcal{N}\left(\sigma;\mathbb{R},\mathbb{R}\right)=\operatorname{span}\left\{\sigma(wx+b),\quad w,b\in\mathbb{R}\right\}.$$

A classical result in Calculus states that if $\sigma\in C^{\infty}$ on any open interval and is not a polynomial thereon, then there exists a point $b^{\star}$ in that interval such that $\sigma^{(k)}(b^{\star})\neq 0$ for all $k=0,1,2,\cdots$. For $h\neq 0$,

$$\frac{\sigma\left((w+h)x+b^{\star}\right)-\sigma\left(wx+b^{\star}\right)}{h}\in\mathcal{N}\left(\sigma;\mathbb{R},\mathbb{R}\right)$$

and so

$$\frac{d}{dw}\sigma\left(wx+b^{\star}\right)|_{w=0}=x\sigma'(b^{\star})\in\overline{\mathcal{N}\left(\sigma;\mathbb{R},\mathbb{R}\right)}.$$

Analogously,

$$\frac{d^k}{dw^k}\sigma\left(wx+b^{\star}\right)|_{w=0}=x^k\sigma^k(b^{\star})\in\overline{\mathcal{N}\left(\sigma;\mathbb{R},\mathbb{R}\right)}.$$

Since $\sigma^k(b^{\star})\neq 0$ for all k, then $\overline{\mathcal{N}\left(\sigma;\mathbb{R},\mathbb{R}\right)}$ contains all polynomials. By Weierstrass theorem, $\mathcal{N}\left(\sigma;\mathbb{R},\mathbb{R}\right)$ is dense in $C\left(K\right)$ for any compact set $K\subset\mathbb{R}$.

Estimates on generalization error for the null control of the wave equation

Estimates on generalization error for the null control of the wave equation

Training error

$$\begin{aligned}\mathcal{E}_{\text{train}} \quad &:= \mathcal{E}_{\text{train, int}} + \mathcal{E}_{\text{train, boundary}} + \mathcal{E}_{\text{train, initialpos}} + \mathcal{E}_{\text{train, initialvel}} \\ &+\mathcal{E}_{\text{train, finalpos}} + \mathcal{E}_{\text{train, finalvel}},\end{aligned}$$

$$\begin{cases} \mathcal{E}_{\text{train, int}} & = \left(\mathcal{L}_{\text{int}}\left(\boldsymbol{\theta}^*; \mathcal{T}_{\text{int}}\right)\right)^{1/2} \\ \mathcal{E}_{\text{train, boundary}} & = \left(\mathcal{L}_{\Gamma_D}\left(\boldsymbol{\theta}^*; \mathcal{T}_{\Gamma_D}\right)\right)^{1/2} \\ \mathcal{E}_{\text{train, initialpos}} & = \left(\mathcal{L}_{t=0}^{\text{pos}}\left(\boldsymbol{\theta}^*; \mathcal{T}_{t=0}\right)\right)^{1/2} \\ \mathcal{E}_{\text{train, initialvel}} & = \left(\mathcal{L}_{t=0}^{\text{vel}}\left(\boldsymbol{\theta}^*; \mathcal{T}_{t=0}\right)\right)^{1/2} \\ \mathcal{E}_{\text{train, finalpos}} & = \left(\mathcal{L}_{t=T}^{\text{pos}}\left(\boldsymbol{\theta}^*; \mathcal{T}_{t=T}\right)\right)^{1/2} \\ \mathcal{E}_{\text{train, finalvel}} & = \left(\mathcal{L}_{t=T}^{\text{vel}}\left(\boldsymbol{\theta}^*; \mathcal{T}_{t=T}\right)\right)^{1/2}, \end{cases}$$

Estimates on generalization error for the null control of the wave equation

Training error

$$
\begin{aligned}
\mathcal{E}_{\text{train}} \quad &:= \mathcal{E}_{\text{train, int}} + \mathcal{E}_{\text{train, boundary}} + \mathcal{E}_{\text{train, initialpos}} + \mathcal{E}_{\text{train, initialvel}} \\
&+ \mathcal{E}_{\text{train, finalpos}} + \mathcal{E}_{\text{train, finalvel}},
\end{aligned}
$$

$$
\begin{cases}
\mathcal{E}_{\text{train, int}} & = \left(\mathcal{L}_{\text{int}}\left(\boldsymbol{\theta}^*; \mathcal{T}_{\text{int}}\right)\right)^{1/2} \\
\mathcal{E}_{\text{train, boundary}} & = \left(\mathcal{L}_{\Gamma_D}\left(\boldsymbol{\theta}^*; \mathcal{T}_{\Gamma_D}\right)\right)^{1/2} \\
\mathcal{E}_{\text{train, initialpos}} & = \left(\mathcal{L}_{t=0}^{\text{pos}}\left(\boldsymbol{\theta}^*; \mathcal{T}_{t=0}\right)\right)^{1/2} \\
\mathcal{E}_{\text{train, initialvel}} & = \left(\mathcal{L}_{t=0}^{\text{vel}}\left(\boldsymbol{\theta}^*; \mathcal{T}_{t=0}\right)\right)^{1/2} \\
\mathcal{E}_{\text{train, finalpos}} & = \left(\mathcal{L}_{t=T}^{\text{pos}}\left(\boldsymbol{\theta}^*; \mathcal{T}_{t=T}\right)\right)^{1/2} \\
\mathcal{E}_{\text{train, finalvel}} & = \left(\mathcal{L}_{t=T}^{\text{vel}}\left(\boldsymbol{\theta}^*; \mathcal{T}_{t=T}\right)\right)^{1/2},
\end{cases}
$$

Generalization error **for control and state**

$$
\begin{cases}
\mathcal{E}_{\text{gener}}(u) := \|u - \hat{u}\|_{L^2(\Gamma_C;(0,T))} \\
\mathcal{E}_{\text{gener}}(y) := \|y - \hat{y}\|_{C\left(0,T;L^2(\Omega)\right)\cap C^1\left(0,T;H^{-1}(\Omega)\right)}
\end{cases}
$$

Approximation theory and convergence analysis

Theorem (Estimates on generalization error)

Assume that both $y, \hat{y} \in C^2\left(\overline{Q_T}\right)$. Then

$$\begin{aligned}
\mathcal{E}_{gener}(u) \quad & \leq C \Big(\mathcal{E}_{train,\ int} + C_{q_{int}}^{1/2} N_{int}^{-\alpha_{int}/2} \\
& + \mathcal{E}_{train,\ boundary} + C_{qb}^{1/2} N_b^{-\alpha_b/2} \\
& + \mathcal{E}_{train,\ initialpos} + C_{qip}^{1/2} N_0^{-\alpha_{ip}/2} \\
& + \mathcal{E}_{train,\ initialvel} + C_{qiv}^{1/2} N_0^{-\alpha_{iv}/2} \\
& + \mathcal{E}_{train,\ finalpos} + C_{qfp}^{1/2} N_T^{-\alpha_{fp}/2} \\
& + \mathcal{E}_{train,\ finalvel} + C_{fv}^{1/2} N_T^{-\alpha_{fv}/2} \Big),
\end{aligned}$$

where $C = C(\Omega, T)$, and consequently $C = C(d)$ also depends on the spatial dimension d. A similar estimate holds for the state variable.
Moreover, training errors converge to zero as the size of the NN and the number of training points go to infinity.

García-Cervera, C., Kessler, M., Periago, F.: *Control of Partial Differential Equations via Physics-Informed Neural Networks* **J. Optim. Th. Appl. 196**, 391–414, 2023.

Recommendations

- **Data normalization.** It is important to ensure that the target output variables vary within a reasonable range. One way to achieve this is through non-dimensionalization of the PDE.

Recommendations

- **Data normalization.** It is important to ensure that the target output variables vary within a reasonable range. One way to achieve this is through non-dimensionalization of the PDE.
- **Network Architecture.** It is recommended to use Multi-layer Perceptrons (MLPs) with depth and width ranging from 3 to 6, and 128 to 512, respectively. As for the activation function, the hyperbolic tangent. For initialization of the network parameters, a Glorot uniform distribution.

Recommendations

- **Data normalization.** It is important to ensure that the target output variables vary within a reasonable range. One way to achieve this is through non-dimensionalization of the PDE.
- **Network Architecture.** It is recommended to use Multi-layer Perceptrons (MLPs) with depth and width ranging from 3 to 6, and 128 to 512, respectively. As for the activation function, the hyperbolic tangent. For initialization of the network parameters, a Glorot uniform distribution.
- **Loss balancing.** $\mathcal{L}(\theta) = \lambda_{ic}\mathcal{L}_{ic}(\theta) + \lambda_{bc}\mathcal{L}_{bc}(\theta) + \lambda_{r}\mathcal{L}_{r}(\theta)$

Recommendations

- **Data normalization.** It is important to ensure that the target output variables vary within a reasonable range. One way to achieve this is through non-dimensionalization of the PDE.
- **Network Architecture.** It is recommended to use Multi-layer Perceptrons (MLPs) with depth and width ranging from 3 to 6, and 128 to 512, respectively. As for the activation function, the hyperbolic tangent. For initialization of the network parameters, a Glorot uniform distribution.
- **Loss balancing.** $\mathcal{L}(\theta) = \lambda_{ic}\mathcal{L}_{ic}(\theta) + \lambda_{bc}\mathcal{L}_{bc}(\theta) + \lambda_r\mathcal{L}_r(\theta)$
  1. Compute

$$\begin{cases} \hat{\lambda}_{ic} = \frac{\|\nabla_\theta \mathcal{L}_{ic}(\theta)\| + \|\nabla_\theta \mathcal{L}_{bc}(\theta)\| + \|\nabla_\theta \mathcal{L}_r(\theta)\|}{\|\nabla_\theta \mathcal{L}_{ic}(\theta)\|} \\ \hat{\lambda}_{bc} = \frac{\|\nabla_\theta \mathcal{L}_{ic}(\theta)\| + \|\nabla_\theta \mathcal{L}_{bc}(\theta)\| + \|\nabla_\theta \mathcal{L}_r(\theta)\|}{\|\nabla_\theta \mathcal{L}_{bc}(\theta)\|} \\ \hat{\lambda}_r = \frac{\|\nabla_\theta \mathcal{L}_{ic}(\theta)\| + \|\nabla_\theta \mathcal{L}_{bc}(\theta)\| + \|\nabla_\theta \mathcal{L}_r(\theta)\|}{\|\nabla_\theta \mathcal{L}_r(\theta)\|} \end{cases} \tag{4}$$

Recommendations

- **Data normalization.** It is important to ensure that the target output variables vary within a reasonable range. One way to achieve this is through non-dimensionalization of the PDE.
- **Network Architecture.** It is recommended to use Multi-layer Perceptrons (MLPs) with depth and width ranging from 3 to 6, and 128 to 512, respectively. As for the activation function, the hyperbolic tangent. For initialization of the network parameters, a Glorot uniform distribution.
- **Loss balancing.** $\mathcal{L}(\theta) = \lambda_{ic}\mathcal{L}_{ic}(\theta) + \lambda_{bc}\mathcal{L}_{bc}(\theta) + \lambda_r\mathcal{L}_r(\theta)$
  1. Compute

$$\begin{cases} \hat{\lambda}_{ic} = \frac{\|\nabla_\theta \mathcal{L}_{ic}(\theta)\| + \|\nabla_\theta \mathcal{L}_{bc}(\theta)\| + \|\nabla_\theta \mathcal{L}_r(\theta)\|}{\|\nabla_\theta \mathcal{L}_{ic}(\theta)\|} \\ \hat{\lambda}_{bc} = \frac{\|\nabla_\theta \mathcal{L}_{ic}(\theta)\| + \|\nabla_\theta \mathcal{L}_{bc}(\theta)\| + \|\nabla_\theta \mathcal{L}_r(\theta)\|}{\|\nabla_\theta \mathcal{L}_{bc}(\theta)\|} \\ \hat{\lambda}_{r} = \frac{\|\nabla_\theta \mathcal{L}_{ic}(\theta)\| + \|\nabla_\theta \mathcal{L}_{bc}(\theta)\| + \|\nabla_\theta \mathcal{L}_r(\theta)\|}{\|\nabla_\theta \mathcal{L}_{r}(\theta)\|} \end{cases} \tag{4}$$

  2. Update the weights $\lambda = (\lambda_{ic}, \lambda_{bc}, \lambda_r)$ by using a moving average

$$\lambda_{\text{new}} = \alpha\lambda_{\text{old}} + (1-\alpha)\hat{\lambda}_{\text{new}}$$

Recommendations

- **Random weight factorization** may improve the performance of PINNs:

$$w^{(k,\ell)} = s^{(k,\ell)} v^{(k,\ell)},$$

with $s^{(k,\ell)}$ is a trainable scale factor assigned to each individual neuron.

S. Wang, et al..: *Random weight factorization improves the training of continuous neural representations.* **arXiv:2210.01274,** 2022.

Recommendations

- **Random weight factorization** may improve the performance of PINNs:

$$w^{(k,\ell)} = s^{(k,\ell)} v^{(k,\ell)},$$

with $s^{(k,\ell)}$ is a trainable scale factor assigned to each individual neuron.

S. Wang, et al..: *Random weight factorization improves the training of continuous neural representations.* **arXiv:2210.01274,** 2022.

- **Optimization algorithm.** The Adaptive with Moment (Adam) algorithm is the most widely used to minimise the loss function. To speed up convergence near a local minimum, it is convenient to combine Adam (say, first 20000 iterations) with a quasi-Newton method like L-BFGS.

Shortcomings

- PINN has difficulty in approximating functions that have **steep gradients**. A Residual-based Adaptive Refinement algorithm has been proposed which add more residual points in the locations where the PDE residual is large.

Lu, L., Meng, X., Mao, Z. and Karniadakis, G.: *DeepXDE: A Deep Learning Library for Solving Differential Equations*, **SIAM Review, 63** (1), 208-228, 2021.

Shortcomings

- PINN has difficulty in approximating functions that have **steep gradients**. A Residual-based Adaptive Refinement algorithm has been proposed which add more residual points in the locations where the PDE residual is large.

  Lu, L., Meng, X., Mao, Z. and Karniadakis, G.: *DeepXDE: A Deep Learning Library for Solving Differential Equations*, **SIAM Review, 63** (1), 208-228, 2021.

- **MLPs are biased towards learning low frecuencies functions.** This can be mitigated by using a random Fourier feature embedding like

$$\gamma(x) = (\cos(Bx), \sin(Bx)),$$

  where B_{ij} are sampled from Gaussians. This maps input coordinates into high feecuency signals before passing through the network.

  M. Tancik, et al..: *Fourier features let networks learn high frequency functions in low dimensional domains.* **arXiv:2006.10739,** 2020.

Shortcomings

- PINN has difficulty in approximating functions that have **steep gradients**. A Residual-based Adaptive Refinement algorithm has been proposed which add more residual points in the locations where the PDE residual is large.

  Lu, L., Meng, X., Mao, Z. and Karniadakis, G.: *DeepXDE: A Deep Learning Library for Solving Differential Equations*, **SIAM Review, 63** (1), 208-228, 2021.

- **MLPs are biased towards learning low frecuencies functions.** This can be mitigated by using a random Fourier feature embedding like

$$\gamma(x) = (\cos(Bx), \sin(Bx)),$$

  where B_{ij} are sampled from Gaussians. This maps input coordinates into high feecuency signals before passing through the network.

  M. Tancik, et al..: *Fourier features let networks learn high frequency functions in low dimensional domains.* **arXiv:2006.10739,** 2020.

- **PINNs may violate temporal casuality** when solving time-dependent PDEs. We should partition the temporal domain into M equal sequential segments and introduce $\mathcal{L}_r^i(\theta)$ to denote the PDE residual loss within the i-th segment. The PDE residual becomes $\mathcal{L}_r(\theta) = \sum_{i=1}^{M} \lambda_i \mathcal{L}_r^i(\theta)$.

  S Wang, S Sankaran, H Wang, P Perdikaris:*An expert's guide to training physics-informed neural networks.* **ArXiv:2308.08468,** 2023.

Part III

Deep Operator Network (DeepONet)

Main objectives and references

Goals

Main objectives and references

Goals

- Given two function spaces X and Y, and an operator

$$\mathcal{G} : X \to Y$$

the main goal is to learn (approximate) the operator $\mathcal{G}$ from a dataset.

Main objectives and references

Goals

- Given two function spaces X and Y, and an operator

$$\mathcal{G} : X \to Y$$

  the main goal is to learn (approximate) the operator $\mathcal{G}$ from a dataset.
- Gain some computational skills on DeepONet -based software

Main objectives and references

Goals

- Given two function spaces X and Y, and an operator

$$\mathcal{G} : X \to Y$$

the main goal is to learn (approximate) the operator $\mathcal{G}$ from a dataset.

- Gain some computational skills on DeepONet -based software

References

Lu, L., Jin, P., Pang, G., Zhang, Z., Karniadakis, G.E.:*Learning nonlinear operators via deeponet based on the universal approximation theorem of operators.* **Nature Machine Intelligence 3**(3), 218-229, 2021.

Lanthaler, S., Mishra, S., Karniadakis, G.E.:*Error estimates for DeepONets: a deep learning framework in infinite dimensions.* **Trans. Math. Appl. 6**(1), 1–144, 2022.

García-Cervera, C.J.; Kessler, M.; Pedregal, P.; Periago, F.: *Universal Approximation of Set-Valued Maps and Application to Control.* **Submitted,** 2025.

Problem setup

For the sake of clarity, we focus on the control problem

$$\begin{cases} y_{tt} - y_{xx} = 0, & \text{in } (0,1) \times (0,2) \\ y(x,0) = y^0(x), & \text{on } (0,1) \\ y_t(x,0) = y^1(x) & \text{on } (0,1) \\ y(0,t) = 0, & \text{on } (0,2) \\ y(1,t) = u(t) & \text{on } (0,2) \\ y(x,2) = y_t(x,2) = 0, & \text{on } (0,1). \end{cases}$$

Problem setup

For the sake of clarity, we focus on the control problem

$$\begin{cases} y_{tt} - y_{xx} = 0, & \text{in } (0,1) \times (0,2) \\ y(x,0) = y^0(x), & \text{on } (0,1) \\ y_t(x,0) = y^1(x) & \text{on } (0,1) \\ y(0,t) = 0, & \text{on } (0,2) \\ y(1,t) = u(t) & \text{on } (0,2) \\ y(x,2) = y_t(x,2) = 0, & \text{on } (0,1). \end{cases}$$

Goal: **approximate the controllabilty mapping**

$$\begin{array}{rcl} \mathcal{G} : \quad L^2(0,1) \times H^{-1}(0,1) & \to & L^2(0,2) \\ (y^0, y^1) & \mapsto & \mathcal{G}(y^0, y^1) := u \end{array}$$

where u is the **unique** control of minimal L^2-norm.

Problem setup

For the sake of clarity, we focus on the control problem

$$\begin{cases} y_{tt} - y_{xx} = 0, & \text{in } (0,1)\times(0,2) \\ y(x,0) = y^0(x), & \text{on } (0,1) \\ y_t(x,0) = y^1(x) & \text{on } (0,1) \\ y(0,t) = 0, & \text{on } (0,2) \\ y(1,t) = u(t) & \text{on } (0,2) \\ y(x,2) = y_t(x,2) = 0, & \text{on } (0,1). \end{cases}$$

Goal: approximate the controllabilty mapping

$$\begin{array}{rcl} \mathcal{G}: \quad L^2(0,1)\times H^{-1}(0,1) & \to & L^2(0,2) \\ (y^0, y^1) & \mapsto & \mathcal{G}(y^0,y^1) := u \end{array}$$

where u is the **unique** control of minimal L^2-norm.
The operator $\mathcal{G}$ is well-defined (uniqueness of the control), linear and continuous. Continuity is a consequence of the observability inequality

$$\|u\|_{L^2(0,2)} \leq C\left(\|y^0\|_{L^2(0,1)} + \|y^1\|_{H^{-1}(0,1)}\right)$$

Thus, $\mathcal{G}$ is Lipschitz continuous.

Machine Learning setup

- **Dataset**
  We fix a set of **sensor points** $\{x_1, x_2, \cdots, x_m\} \subset [0, 1]$. The information of each selected continuous initial datum (y^0, y^1) is encoded in the vector

$$(y^0(x_1), y^0(x_2), \cdots, y^0(x_m); y^1(x_1), y^1(x_2), \cdots, y^1(x_m)) \equiv y^{\text{initial}}$$

Machine Learning setup

- **Dataset**

  We fix a set of **sensor points** $\{x_1, x_2, \cdots, x_m\} \subset [0, 1]$. The information of each selected continuous initial datum (y^0, y^1) is encoded in the vector

  $$(y^0(x_1), y^0(x_2), \cdots, y^0(x_m); y^1(x_1), y^1(x_2), \cdots, y^1(x_m)) \equiv y^{\text{initial}}$$

  We also take $t \in [0, 2]$.

Machine Learning setup

- **Dataset**

  We fix a set of **sensor points** $\{x_1, x_2, \cdots, x_m\} \subset [0, 1]$. The information of each selected continuous initial datum (y^0, y^1) is encoded in the vector

  $$(y^0(x_1), y^0(x_2), \cdots, y^0(x_m); y^1(x_1), y^1(x_2), \cdots, y^1(x_m)) \equiv y^{\text{initial}}$$

  We also take $t \in [0, 2]$. Putting all together,

  $$\{(y_\ell^{\text{initial}}; t_\ell), \quad 1 \leq \ell \leq N\} \tag{5}$$

Machine Learning setup

- **Dataset**

  We fix a set of **sensor points** $\{x_1, x_2, \cdots, x_m\} \subset [0, 1]$. The information of each selected continuous initial datum (y^0, y^1) is encoded in the vector

  $$(y^0(x_1), y^0(x_2), \cdots, y^0(x_m); y^1(x_1), y^1(x_2), \cdots, y^1(x_m)) \equiv y^{\text{initial}}$$

  We also take $t \in [0, 2]$. Putting all together,

  $$\{(y_\ell^{\text{initial}}; t_\ell), \quad 1 \le \ell \le N\} \tag{5}$$

  The corresponding **labels** are

  $$\{u_\ell = u(y_\ell^{\text{initial}}; t_\ell), \quad 1 \le \ell \le N\}, \tag{6}$$

  the control at time t_ℓ associated with the initial datum y_ℓ^{initial}.

Machine Learning setup

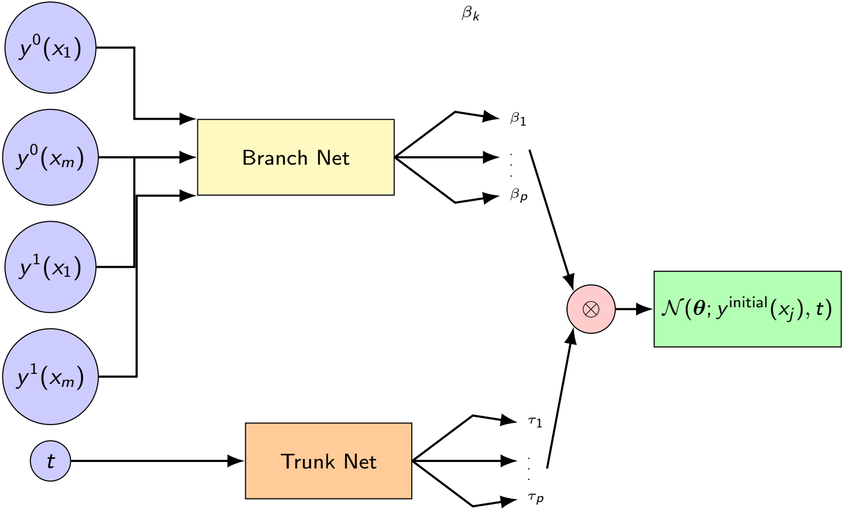
- **Dataset**
  We fix a set of **sensor points** $\{x_1, x_2, \cdots, x_m\} \subset [0,1]$. The information of each selected continuous initial datum (y^0, y^1) is encoded in the vector
  $$(y^0(x_1), y^0(x_2), \cdots, y^0(x_m); y^1(x_1), y^1(x_2), \cdots, y^1(x_m)) \equiv y^{\text{initial}}$$
  We also take $t \in [0,2]$. Putting all together,
  $$\{(y_\ell^{\text{initial}}; t_\ell), \quad 1 \leq \ell \leq N\} \tag{5}$$
  The corresponding **labels** are
  $$\{u_\ell = u(y_\ell^{\text{initial}}; t_\ell), \quad 1 \leq \ell \leq N\}, \tag{6}$$
  the control at time t_ℓ associated with the initial datum y_ℓ^{initial}.
- **Hypothesis space: the neural network**
  We will use the so-called **DeepONet**, which takes the form
  $$\mathcal{N}(\boldsymbol{\theta}; (y^{\text{initial}}(x_j); t)) := \sum_{k=1}^{p} \sum_{i=1}^{n} c_i^k \sigma\left(\sum_{j=1}^{m} \xi_{ij}^k y^{\text{initial}}(x_j) + \theta_i^k\right) \cdot \sigma(w_k \cdot t + \eta_k) \tag{7}$$
  where $\boldsymbol{\theta} = (c_i^k, \xi_{ij}^k, \theta_i^k, w_k, \eta_k)$ is the set of parameters of the net.

Machine Learning setup

- **Dataset**
  We fix a set of **sensor points** $\{x_1, x_2, \cdots, x_m\} \subset [0,1]$. The information of each selected continuous initial datum (y^0, y^1) is encoded in the vector
  $$(y^0(x_1), y^0(x_2), \cdots, y^0(x_m); y^1(x_1), y^1(x_2), \cdots, y^1(x_m)) \equiv y^{\text{initial}}$$
  We also take $t \in [0,2]$. Putting all together,
  $$\{(y_\ell^{\text{initial}}; t_\ell), \quad 1 \leq \ell \leq N\} \tag{5}$$
  The corresponding **labels** are
  $$\{u_\ell = u(y_\ell^{\text{initial}}; t_\ell), \quad 1 \leq \ell \leq N\}, \tag{6}$$
  the control at time t_ℓ associated with the initial datum y_ℓ^{initial}.
- **Hypothesis space: the neural network**
  We will use the so-called **DeepONet**, which takes the form
  $$\mathcal{N}(\boldsymbol{\theta}; (y^{\text{initial}}(x_j); t)) := \sum_{k=1}^{p} \sum_{i=1}^{n} c_i^k \sigma \left(\sum_{j=1}^{m} \xi_{ij}^k y^{\text{initial}}(x_j) + \theta_i^k \right) \cdot \sigma(w_k \cdot t + \eta_k) \tag{7}$$
  where $\boldsymbol{\theta} = (c_i^k, \xi_{ij}^k, \theta_i^k, w_k, \eta_k)$ is the set of parameters of the net.
- **Loss function: Mean Squared Error (MSE)**
  $$\text{MSE}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{\ell=1}^{N} |\mathcal{N}(\boldsymbol{\theta}; (y_\ell^{\text{initial}}; t_\ell)) - u_\ell|^2 \tag{8}$$

DeepONet's architecture

$$\mathcal{N}(\boldsymbol{\theta}; (y^{\text{initial}}(x_j); t)) := \sum_{k=1}^{p} \underbrace{\sum_{i=1}^{n} c_i^k \sigma\left(\sum_{j=1}^{m} \xi_{ij}^k y^{\text{initial}}(x_j) + \theta_i^k\right)}_{\beta_k} \cdot \underbrace{\sigma(w_k \cdot t + \eta_k)}_{\tau_k}$$



Where does this architecture come from ?

Theorem (Universal Approximation Theorem for Functions)

Suppose that $K \in \mathbb{R}^d$ is compact, $U \subset C(K)$ is compact, and $\sigma \in$ is not a polynomial. Then, for any $\varepsilon > 0$ there exist a positive integer n, real numbers θ_i, $\omega_i \in \mathbb{R}^n$, independent of $f \in U$, and constants $c_i = c_i(f)$ depending on f, such that

$$|f(x) - \sum_{i=1}^{n} c_i \sigma(\omega_i \cdot x + \theta_i)| < \varepsilon$$

holds for all $x \in K$ and $f \in U$. Moreover, each $c_i(f)$ is a continuous linear functional defined on U.

Where does this architecture come from ?

Theorem (Universal Approximation Theorem for Functions)

Suppose that $K \in \mathbb{R}^d$ *is compact,* $U \subset C(K)$ *is compact, and* $\sigma \in$ *is not a polynomial. Then, for any* $\varepsilon > 0$ *there exist a positive integer* n*, real numbers* θ_i*,* $\omega_i \in \mathbb{R}^n$*, independent of* $f \in U$*, and constants* $c_i = c_i(f)$ *depending on* f*, such that*

$$|f(x) - \sum_{i=1}^{n} c_i \sigma(\omega_i \cdot x + \theta_i)| < \varepsilon$$

holds for all $x \in K$ *and* $f \in U$*. Moreover, each* $c_i(f)$ *is a continuous linear functional defined on* U*.*

Theorem (Universal Approximation Theorem for Functionals)

Suppose that σ *is not a polynomial,* X *is a Banach space,* $K \subset X$ *is a compact set,* V *is a compact set in* $C(K)$*, and* $f : V \to \mathbb{R}$ *is a continuous functional. Then for any* $\varepsilon > 0$*, there are a positive integer* n*,* m *sensor points* $x_1, x_2, \cdots, x_m \in K$*, and real constants* c_i, θ_i, ξ_{ij}*,* $1 \leq i \leq n$*,* $1 \leq j \leq m$*, such that*

$$|f(y) - \sum_{i=1}^{n} c_i \sigma\left(\sum_{j=1}^{m} \xi_{ij} y(x_j) + \theta_i\right)| < \varepsilon, \quad \text{for all } u \in V.$$

Where does this architecture come from ?

Theorem (Universal Approximation Theorem for Operators)

Suppose that σ is not a polynomial, X is a Banach space, $K_1 \subset X$, $K_2 \subset \mathbb{R}^d$ are compact sets, V is a compact set in $C(K_1)$, and $\mathcal{G} : V \to C(K_2)$ is a continuous operator. Then for any $\varepsilon > 0$, there are a positive integers n, p, m sensor points $x_1, x_2, \cdots, x_m \in K_1$, and real constants $c_i^k, \theta_i^k, \xi_{ij}^k, \eta_k$ such that

$$|\mathcal{G}(y)(t) - \sum_{k=1}^{p} \sum_{i=1}^{n} c_i^k \sigma \left(\sum_{j=1}^{m} \xi_{ij}^k y(x_j) + \theta_i^k \right) \sigma(\omega_k \cdot t + \zeta_k)| < \varepsilon, \quad \forall y \in V, t \in K_2$$

Chen, T., Chen, H.:*Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems.* **IEEE Transactions on Neural Networks 6**(4), 911-917, 1995.

DeepONet setup: Dataset

Definition (Data for DeepONet approximation)

Assume that $X \hookrightarrow L^2(D)$, and $Y \hookrightarrow L^2(U)$, for some Banach spaces X, Y. The pair $(\mu, \mathcal{G})$ is said to be **data for DeepONet approximation** provided $\mu \in \mathcal{P}_2(X)$ is Borel measurable with finite second moments, there exists a Borel set $A \subset X$, composed of continuous functions, $\mu(A) = 1$, and $\mathcal{G} : X \to Y$ is a Borel measurable mapping with $\|\mathcal{G}\|_{L^2(\mu)} < \infty$.

DeepONet setup: Dataset

Definition (Data for DeepONet approximation)

Assume that $X \hookrightarrow L^2(D)$, and $Y \hookrightarrow L^2(U)$, for some Banach spaces X, Y. The pair $(\mu, \mathcal{G})$ is said to be **data for DeepONet approximation** provided $\mu \in \mathcal{P}_2(X)$ is Borel measurable with finite second moments, there exists a Borel set $A \subset X$, composed of continuous functions, $\mu(A) = 1$, and $\mathcal{G} : X \to Y$ is a Borel measurable mapping with $\|\mathcal{G}\|_{L^2(\mu)} < \infty$.

We need an **encoding operator** $\mathcal{E} : X \to \mathbb{R}^m$ to represent functions as vectors

DeepONet setup: Dataset

Definition (Data for DeepONet approximation)

Assume that $X \hookrightarrow L^2(D)$, and $Y \hookrightarrow L^2(U)$, for some Banach spaces X, Y. The pair $(\mu, \mathcal{G})$ is said to be **data for DeepONet approximation** provided $\mu \in \mathcal{P}_2(X)$ is Borel measurable with finite second moments, there exists a Borel set $A \subset X$, composed of continuous functions, $\mu(A) = 1$, and $\mathcal{G} : X \to Y$ is a Borel measurable mapping with $\|\mathcal{G}\|_{L^2(\mu)} < \infty$.

We need an **encoding operator** $\mathcal{E} : X \to \mathbb{R}^m$ to represent functions as vectors
How to sample continuous functions for the initial data (y^0, y^1) ?

DeepONet setup: Dataset

Definition (Data for DeepONet approximation)

Assume that $X \hookrightarrow L^2(D)$, and $Y \hookrightarrow L^2(U)$, for some Banach spaces X, Y. The pair $(\mu, \mathcal{G})$ is said to be **data for DeepONet approximation** provided $\mu \in \mathcal{P}_2(X)$ is Borel measurable with finite second moments, there exists a Borel set $A \subset X$, composed of continuous functions, $\mu(A) = 1$, and $\mathcal{G} : X \to Y$ is a Borel measurable mapping with $\|\mathcal{G}\|_{L^2(\mu)} < \infty$.

We need an **encoding operator** $\mathcal{E} : X \to \mathbb{R}^m$ to represent functions as vectors
How to sample continuous functions for the initial data (y^0, y^1) ?
We choose $\mu \in \mathcal{P}\left(C(0,1)\right)$ a probability measure whose probability law is given by a Karhunen-Loève expansion

$$\mu \sim \sum_{j=1}^{\infty} \sqrt{\lambda_j}\xi_j(\omega)\varphi_j(x) \tag{9}$$

We take $\xi_j \sim \mathcal{N}(0,1)$ i.i.d. standard Gaussian variables, and (λ_j, φ_j) the eigenpairs associated with squared exponential covariance function

$$C(x, x') = \sigma^2 \exp\left(-\frac{(x-x')^2}{2\ell^2}\right). \tag{10}$$

DeepONet setup: Dataset

Definition (Data for DeepONet approximation)

Assume that $X \hookrightarrow L^2(D)$, and $Y \hookrightarrow L^2(U)$, for some Banach spaces X, Y. The pair $(\mu, \mathcal{G})$ is said to be **data for DeepONet approximation** provided $\mu \in \mathcal{P}_2(X)$ is Borel measurable with finite second moments, there exists a Borel set $A \subset X$, composed of continuous functions, $\mu(A) = 1$, and $\mathcal{G} : X \to Y$ is a Borel measurable mapping with $\|\mathcal{G}\|_{L^2(\mu)} < \infty$.

We need an **encoding operator** $\mathcal{E} : X \to \mathbb{R}^m$ to represent functions as vectors
How to sample continuous functions for the initial data (y^0, y^1) ?
We choose $\mu \in \mathcal{P}\left(C(0,1)\right)$ a probability measure whose probability law is given by a Karhunen-Loève expansion

$$\mu \sim \sum_{j=1}^{\infty} \sqrt{\lambda_j}\xi_j(\omega)\varphi_j(x) \tag{9}$$

We take $\xi_j \sim \mathcal{N}(0,1)$ i.i.d. standard Gaussian variables, and (λ_j, φ_j) the eigenpairs associated with squared exponential covariance function

$$C(x, x') = \sigma^2 \exp\left(-\frac{(x - x')^2}{2\ell^2}\right). \tag{10}$$

Thus, we truncate (9) and sample the Gaussian random variables. Remember that by Mercer's theorem $\{\varphi_j\}$ is an ortonormal basis of $L^2(0,1)$.

DeepONet setup: Dataset

After fixing a set of **sensor points** $\{x_1, x_2, \cdots, x_m\} \subset [0, 1]$, the information of each selected continuous initial datum (y^0, y^1) is **encoded** in the vector

$$(y^0(x_1), y^0(x_2), \cdots, y^0(x_m); y^1(x_1), y^1(x_2), \cdots, y^1(x_m)) \equiv y^{\text{initial}}$$

DeepONet setup: Dataset

After fixing a set of **sensor points** $\{x_1, x_2, \cdots, x_m\} \subset [0, 1]$, the information of each selected continuous initial datum (y^0, y^1) is **encoded** in the vector

$$(y^0(x_1), y^0(x_2), \cdots, y^0(x_m); y^1(x_1), y^1(x_2), \cdots, y^1(x_m)) \equiv y^{\text{initial}}$$

The encoding operator $\mathcal{E} : C(0,1) \times C(0,1) \to \mathbb{R}^m \times \mathbb{R}^m$ maps $(y^0, y^1) \mapsto y^{\text{initial}}$

DeepONet setup: Dataset

After fixing a set of **sensor points** $\{x_1, x_2, \cdots, x_m\} \subset [0,1]$, the information of each selected continuous initial datum (y^0, y^1) is **encoded** in the vector

$$(y^0(x_1), y^0(x_2), \cdots, y^0(x_m); y^1(x_1), y^1(x_2), \cdots, y^1(x_m)) \equiv y^{\text{initial}}$$

The encoding operator $\mathcal{E} : C(0,1) \times C(0,1) \to \mathbb{R}^m \times \mathbb{R}^m$ maps $(y^0, y^1) \mapsto y^{\text{initial}}$
We repeat this process for a number of initial conditions (y^0_j, y^1_j) and store the corresponding y^{initial}_{jk} in the rows of the so-called matrix of **features.**

DeepONet setup: Dataset

After fixing a set of **sensor points** $\{x_1, x_2, \cdots, x_m\} \subset [0,1]$, the information of each selected continuous initial datum (y^0, y^1) is **encoded** in the vector

$$(y^0(x_1), y^0(x_2), \cdots, y^0(x_m); y^1(x_1), y^1(x_2), \cdots, y^1(x_m)) \equiv y^{\text{initial}}$$

The encoding operator $\mathcal{E} : C(0,1) \times C(0,1) \to \mathbb{R}^m \times \mathbb{R}^m$ maps $(y^0, y^1) \mapsto y^{\text{initial}}$
We repeat this process for a number of initial conditions (y_j^0, y_j^1) and store the corresponding y_{jk}^{initial} in the rows of the so-called matrix of **features.**
The controls $u_j(t) := u_j(t; (y_j^0, y_j^1))$, evaluated at some time t are the so-called vector of **labels**).

DeepONet setup: Dataset

After fixing a set of **sensor points** $\{x_1, x_2, \cdots, x_m\} \subset [0,1]$, the information of each selected continuous initial datum (y^0, y^1) is **encoded** in the vector

$$(y^0(x_1), y^0(x_2), \cdots, y^0(x_m); y^1(x_1), y^1(x_2), \cdots, y^1(x_m)) \equiv y^{\text{initial}}$$

The encoding operator $\mathcal{E} : C(0,1) \times C(0,1) \to \mathbb{R}^m \times \mathbb{R}^m$ maps $(y^0, y^1) \mapsto y^{\text{initial}}$

We repeat this process for a number of initial conditions (y_j^0, y_j^1) and store the corresponding y_{jk}^{initial} in the rows of the so-called matrix of **features.**

The controls $u_j(t) := u_j(t; (y_j^0, y_j^1))$, evaluated at some time t are the so-called vector of **labels**).

Putting all together, the **training dataset** is

$$\{(y_{jk}^{\text{initial}}; u_j(t)), 1 \leq j \leq N, 1 \leq k \leq 2mm\},$$

evaluated at a finite selection of times t. Precisely,

$$\left[\begin{array}{cccc}
y_{1,1}^{\text{initial}} & \cdots & y_{1,2m}^{\text{initial}} & u_1(t_1) \\
\cdots & \cdots & \cdots & \cdots \\
y_{1,1}^{\text{initial}} & \cdots & y_{1,2m}^{\text{initial}} & u_1(t_\ell) \\
\hline
\cdots & \cdots & \cdots & \cdots \\
\cdots & \cdots & \cdots & \cdots \\
\hline
y_{N,1}^{\text{initial}} & \cdots & y_{N,2m}^{\text{initial}} & u_N(t_1) \\
\cdots & \cdots & \cdots & \cdots \\
y_{N,1}^{\text{initial}} & \cdots & y_{N,2m}^{\text{initial}} & u_N(t_\ell)
\end{array}\right] \tag{11}$$

Error analysis and the curse of dimensionality

When we talk about error, what are we talking about?

Error analysis and the curse of dimensionality

When we talk about error, what are we talking about?

- **Approximation error:** This is the distance between the Hypothesis space and the operator $\mathcal{G}$ to be approximated, i.e., if $\mathcal{F}$ is a fixed space of DeepONets, then

$$\mathcal{N}_{\mathcal{F}} = \arg\min_{\mathcal{N}\in\mathcal{F}} \mathcal{L}(\mathcal{N}) := \int_{L^2(D)} \int_U |\mathcal{G}(y)(t) - \mathcal{N}_{\mathcal{F}}(y)(t)|^2 \, dt d\mu(y),$$

then **approximation error** is

$$\mathcal{E}_{\text{approx}} := \sqrt{\mathcal{L}(\mathcal{N}_{\mathcal{F}})}.$$

Error analysis and the curse of dimensionality

When we talk about error, what are we talking about?

- **Approximation error:** This is the distance between the Hypothesis space and the operator $\mathcal{G}$ to be approximated, i.e., if $\mathcal{F}$ is a fixed space of DeepONets, then

$$\mathcal{N}_{\mathcal{F}} = \arg\min_{\mathcal{N}\in\mathcal{F}} \mathcal{L}(\mathcal{N}) := \int_{L^2(D)} \int_U |\mathcal{G}(y)(t) - \mathcal{N}_{\mathcal{F}}(y)(t)|^2 \, dt d\mu(y),$$

then **approximation error** is

$$\mathcal{E}_{\text{approx}} := \sqrt{\mathcal{L}(\mathcal{N}_{\mathcal{F}})}.$$

- **Generalization (or estimation) error:** We approximate $\mathcal{N}_{\mathcal{F}}$ by using a specific (training) dataset $\mathcal{T}$, and a **empirical loss**. So, we get

$$\mathcal{N}_{\mathcal{T}} = \arg\min_{\mathcal{N}\in\mathcal{F}} \mathcal{L}_M(\mathcal{N}) := \frac{|U|}{M} \sum_{j=1}^{M} |\mathcal{G}(y_j)(t_j) - \mathcal{N}(y_j)(t_j)|^2$$

Thus, $\mathcal{E}_{\text{gener}} := \left(\mathcal{L}(\mathcal{N}_{\mathcal{F}}) - \mathcal{L}_M(\mathcal{N}_{\mathcal{T}})\right)^{1/2}$.

Error analysis and the curse of dimensionality

When we talk about error, what are we talking about?

- **Approximation error:** This is the distance between the Hypothesis space and the operator $\mathcal{G}$ to be approximated, i.e., if $\mathcal{F}$ is a fixed space of DeepONets, then

$$\mathcal{N}_{\mathcal{F}} = \arg\min_{\mathcal{N}\in\mathcal{F}} \mathcal{L}(\mathcal{N}) := \int_{L^2(D)} \int_U |\mathcal{G}(y)(t) - \mathcal{N}_{\mathcal{F}}(y)(t)|^2 \, dt d\mu(y),$$

  then **approximation error** is

$$\mathcal{E}_{\text{approx}} := \sqrt{\mathcal{L}(\mathcal{N}_{\mathcal{F}})}.$$

- **Generalization (or estimation) error:** We approximate $\mathcal{N}_{\mathcal{F}}$ by using a specific (training) dataset $\mathcal{T}$, and a **empirical loss**. So, we get

$$\mathcal{N}_{\mathcal{T}} = \arg\min_{\mathcal{N}\in\mathcal{F}} \mathcal{L}_M(\mathcal{N}) := \frac{|U|}{M} \sum_{j=1}^{M} |\mathcal{G}(y_j)(t_j) - \mathcal{N}(y_j)(t_j)|^2$$

  Thus, $\mathcal{E}_{\text{gener}} := \left(\mathcal{L}(\mathcal{N}_{\mathcal{F}}) - \mathcal{L}_M(\mathcal{N}_{\mathcal{T}})\right)^{1/2}$.

- **Optimization error:** The empirical loss is highly nonlinear, non-convex so that we compute a local minimum $\mathcal{N}_M$ of $\mathcal{L}_M$. Optimization error is then

$$\mathcal{E}_{\text{optim}} := \|\mathcal{N}_M - \mathcal{N}_{\mathcal{T}}\|^{1/2}.$$

Error analysis and the curse of dimensionality

$$\mathcal{E}_{\text{total}} = \mathcal{E}_{\text{approx}} + \mathcal{E}_{\text{gener}} + \mathcal{E}_{\text{optim}}$$

Definition (Curse of dimensionality)

For a given $\varepsilon > 0$, let $\mathcal{N}_\varepsilon$ be a DeepONet such that $\mathcal{E}_{\text{approx}} < \varepsilon$, and

$$\text{size}\left(\mathcal{N}_\varepsilon\right) \sim \mathcal{O}\left(\varepsilon^{-\vartheta_\varepsilon}\right) \quad \text{for some } \vartheta_\varepsilon \geq 0.$$

Our DeepONet approximation, with underlying measure μ, is said to *incurr a curse of dimensionality* if $\lim_{\varepsilon\to 0} \vartheta_\varepsilon = +\infty$ and *breaks the curse of dimensionality* if $\lim_{\varepsilon\to 0} \vartheta_\varepsilon = \overline{\vartheta} < +\infty$.

[1] Size of a neural network is understood as the number of non-vanishing parameters of the net.

Error analysis and the curse of dimensionality

$$\mathcal{E}_{\text{total}} = \mathcal{E}_{\text{approx}} + \mathcal{E}_{\text{gener}} + \mathcal{E}_{\text{optim}}$$

Definition (Curse of dimensionality)

For a given $\varepsilon > 0$, let $\mathcal{N}_\varepsilon$ be a DeepONet such that $\mathcal{E}_{\text{approx}} < \varepsilon$, and

$$\text{size}\,(\mathcal{N}_\varepsilon) \sim \mathcal{O}\left(\varepsilon^{-\vartheta_\varepsilon}\right) \quad \text{for some } \vartheta_\varepsilon \geq 0.$$

Our DeepONet approximation, with underlying measure μ, is said to *incurr a curse of dimensionality* if $\lim_{\varepsilon\to 0} \vartheta_\varepsilon = +\infty$ and *breaks the curse of dimensionality* if $\lim_{\varepsilon\to 0} \vartheta_\varepsilon = \overline{\vartheta} < +\infty$.

Yarotsky proved that the approximation of a general Lipschitz function to accuracy ε requires a ReLU network of size[1] $\varepsilon^{-m(\varepsilon)/2}$, with $m(\varepsilon) \to \infty$ as $\varepsilon \to 0$, and hence suffers from the curse of dimensionality.

Yarotsky, D.: *Optimal approximation of continuous functions by very deep relu networks.* **Conference on Learning Theory. PMLR,** 639-649, 2018.

In our setting, m is the number of sensors for the enconding operator $u \mapsto \mathcal{E}(u) = (u(x_1), \cdots, u(x_m))$.

[1] Size of a neural network is understood as the number of non-vanishing parameters of the net.

Error analysis and the curse of dimensionality

However, for some classes of linear and nonlinear operators, the DeepONet approximation may break the curse of dimensionality for **approximation error.**

Lanthaler, S., Mishra, S., Karniadakis, G.E.:*Error estimates for DeepONets: a deep learning framework in infinite dimensions.* **Trans. Math. Appl. 6**(1), 1–144, 2022.

Some examples are:

Error analysis and the curse of dimensionality

However, for some classes of linear and nonlinear operators, the DeepONet approximation may break the curse of dimensionality for **approximation error.**

Lanthaler, S., Mishra, S., Karniadakis, G.E.: *Error estimates for DeepONets: a deep learning framework in infinite dimensions.* **Trans. Math. Appl. 6**(1), 1–144, 2022.

Some examples are:

- holomorphic mappings $[-1,1]^N \to V$, $y \mapsto \mathcal{G}(y)$ with V an arbitrary Banach space,

Error analysis and the curse of dimensionality

However, for some classes of linear and nonlinear operators, the DeepONet approximation may break the curse of dimensionality for **approximation error.**

Lanthaler, S., Mishra, S., Karniadakis, G.E.: *Error estimates for DeepONets: a deep learning framework in infinite dimensions.* **Trans. Math. Appl. 6**(1), 1–144, 2022.

Some examples are:

- holomorphic mappings $[-1,1]^N \to V$, $y \mapsto \mathcal{G}(y)$ with V an arbitrary Banach space,
- some PDE operators like
  1. parametric elliptic PDEs: $\mathcal{G} : a \mapsto y$, where y solves

$$-\nabla \cdot (a\nabla y) = f$$

Error analysis and the curse of dimensionality

However, for some classes of linear and nonlinear operators, the DeepONet approximation may break the curse of dimensionality for **approximation error.**

Lanthaler, S., Mishra, S., Karniadakis, G.E.: *Error estimates for DeepONets: a deep learning framework in infinite dimensions.* **Trans. Math. Appl. 6**(1), 1–144, 2022.

Some examples are:

- holomorphic mappings $[-1,1]^N \to V$, $y \mapsto \mathcal{G}(y)$ with V an arbitrary Banach space,
- some PDE operators like
  1. parametric elliptic PDEs: $\mathcal{G} : a \mapsto y$, where y solves
  $$-\nabla \cdot (a \nabla y) = f$$
  2. parabolic nonlinear PDEs: $\mathcal{G} : y^0 \mapsto y(T)$, where y solves
  $$\begin{cases} \frac{\partial y}{\partial t} = \Delta y + f(y) \\ y(0) = y^0 \end{cases}$$

Error analysis and the curse of dimensionality

However, for some classes of linear and nonlinear operators, the DeepONet approximation may break the curse of dimensionality for **approximation error.**

Lanthaler, S., Mishra, S., Karniadakis, G.E.:*Error estimates for DeepONets: a deep learning framework in infinite dimensions.* **Trans. Math. Appl. 6**(1), 1–144, 2022.

Some examples are:

- holomorphic mappings $[-1,1]^N \to V$, $y \mapsto \mathcal{G}(y)$ with V an arbitrary Banach space,
- some PDE operators like
  1. parametric elliptic PDEs: $\mathcal{G} : a \mapsto y$, where y solves
  $$-\nabla \cdot (a\nabla y) = f$$
  2. parabolic nonlinear PDEs: $\mathcal{G} : y^0 \mapsto y(T)$, where y solves
  $$\begin{cases} \frac{\partial y}{\partial t} = \Delta y + f(y) \\ y(0) = y^0 \end{cases}$$
  3. scalar conservation laws: $\mathcal{G} : y^0 \mapsto y(T)$, where y solves t
  $$\begin{cases} \partial_t y + \partial_x (f(y)) \\ y(0) = y^0 \end{cases}$$

Error analysis and the curse of dimensionality

However, for some classes of linear and nonlinear operators, the DeepONet approximation may break the curse of dimensionality for **approximation error.**

Lanthaler, S., Mishra, S., Karniadakis, G.E.: *Error estimates for DeepONets: a deep learning framework in infinite dimensions.* **Trans. Math. Appl. 6**(1), 1–144, 2022.

Some examples are:

- holomorphic mappings $[-1,1]^N \to V$, $y \mapsto \mathcal{G}(y)$ with V an arbitrary Banach space,
- some PDE operators like
  1. parametric elliptic PDEs: $\mathcal{G} : a \mapsto y$, where y solves
  $$-\nabla \cdot (a\nabla y) = f$$
  2. parabolic nonlinear PDEs: $\mathcal{G} : y^0 \mapsto y(T)$, where y solves
  $$\begin{cases} \frac{\partial y}{\partial t} = \Delta y + f(y) \\ y(0) = y^0 \end{cases}$$
  3. scalar conservation laws: $\mathcal{G} : y^0 \mapsto y(T)$, where y solves t
  $$\begin{cases} \partial_t y + \partial_x (f(y)) \\ y(0) = y^0 \end{cases}$$
- bounded linear operators $\mathcal{G} : L^2(D) \to L^2(U)$

Error analysis and the curse of dimensionality

As for generalization error under suitable boundedness and Lipschitz continuity assumptions one gets

$$\mathcal{E}_{\text{gener}} \leq \frac{C}{\sqrt{N}} \left(1 + C d_\theta \log\left(CB\sqrt{N}\right)\right)^{2\kappa+\frac{1}{2}}$$

where N is the number of sampling functions, d_θ is the number of parameters of the DeepONet, and C, B and κ are suitable constants.

Lanthaler, S., Mishra, S., Karniadakis, G.E.: *Error estimates for DeepONets: a deep learning framework in infinite dimensions.* **Trans. Math. Appl. 6**(1), 1–144, 2022.

Part IV

Numerical Implementation via DeepXDE

PINN. Experiment 1: the Laplace-Poisson equation

$$\begin{cases} -\Delta u(x) = -2d, & x \in D := (0,1)^d \\ u(x) = \sum_{k=1}^{d} x_k^2, & x \in \partial D \end{cases}$$

This problem has the exact solution

$$u_{\text{exact}}(x) = \sum_{k=1}^{d} x_k^2$$

PINN. Experiment 1: the Laplace-Poisson equation

$$\begin{cases} -\Delta u(x) = -2d, & x \in D := (0,1)^d \\ u(x) = \sum_{k=1}^{d} x_k^2, & x \in \partial D \end{cases}$$

This problem has the exact solution

$$u_{\text{exact}}(x) = \sum_{k=1}^{d} x_k^2$$

Python script is available at https://github.com/fperiago/deepcontrol

PINN. Experiment 1: the Laplace-Poisson equation

$$\begin{cases} -\Delta u(x) = -2d, & x \in D := (0,1)^d \\ u(x) = \sum_{k=1}^{d} x_k^2, & x \in \partial D \end{cases}$$

This problem has the exact solution

$$u_{\text{exact}}(x) = \sum_{k=1}^{d} x_k^2$$

Python script is available at https://github.com/fperiago/deepcontrol

Table: L^2- relative error $\frac{\| u_{\text{exact}} - u_{\text{PINN}} \|_{L^2(D)}}{\| u_{\text{exact}} \|_{L^2(D)}}$ for different values of the spatial dimension d and number of training points $N = N_{\text{interior}} + N_{\text{boundary}}$

	$N = 100 + 40$	$N = 1000 + 400$	$N = 10000 + 4000$
$d = 2$	9×10^{-4}	1.2×10^{-4}	7.5×10^{-5}
$d = 3$	1.2×10^{-3}	1.4×10^{-4}	1.3×10^{-3}
$d = 5$	2.5×10^{-2}	7.5×10^{-4}	1.9×10^{-4}
$d = 10$	2.2×10^{-1}	4.2×10^{-3}	2.2×10^{-3}
$d = 20$	3.1×10^{-1}	2.6×10^{-2}	2.5×10^{-3}

PINN. Experiment 1: the Laplace-Poisson equation

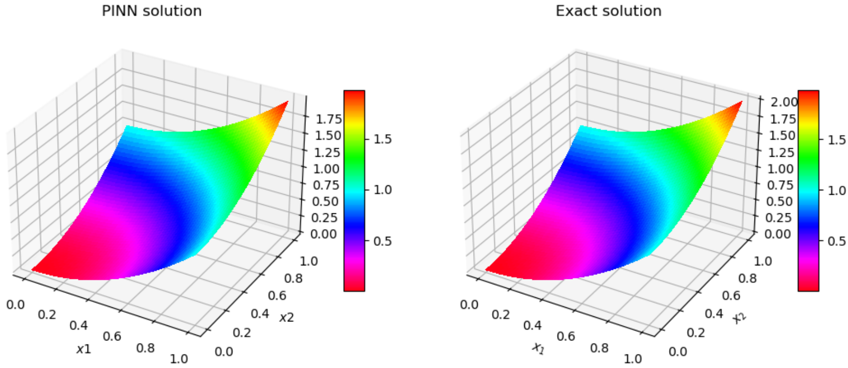


Figure: Experiment 1. Comparison between PINN (or predicted) solution u_{PINN} **(left)**, and exact solution u_{exact} **(right)**. Neural network composed of 3 hidden layers and 50 neurons in each layer. No regularization. Number of training points $N_{\text{interior}} = 100$, $N_{\text{boundary}} = 40$.

PINN. Experiment 2: exact control of the wave equation

Compute $u(t)$ such that the solution $y(x,t)$ of the problem

$$\begin{cases} y_{tt} - y_{xx} = 0, & \text{in } (0,1)\times(0,2) \\ y(x,0) = \sin(\pi x), & \text{in } (0,1) \\ y_t(x,0) = 0 & \text{in } (0,1) \\ y(0,t) = 0, & \text{on } (0,2) \\ y(1,t) = u(t) & \text{on } (0,2) \end{cases}$$

satifies

$$y(x,2) = y_t(x,2) = 0.$$

PINN. Experiment 2: exact control of the wave equation

Compute $u(t)$ such that the solution $y(x,t)$ of the problem

$$\begin{cases} y_{tt} - y_{xx} = 0, & \text{in } (0,1)\times(0,2) \\ y(x,0) = \sin(\pi x), & \text{in } (0,1) \\ y_t(x,0) = 0 & \text{in } (0,1) \\ y(0,t) = 0, & \text{on } (0,2) \\ y(1,t) = u(t) & \text{on } (0,2) \end{cases}$$

satifies

$$y(x,2) = y_t(x,2) = 0.$$

This problem has an exact solution which can be obtained trough D'Alembert formula. Indeed, by considering the function

$$\tilde{y^0}(x) = \begin{cases} \sin(\pi x) & -1 \le x \le 1 \\ 0 & \text{elsewhere,} \end{cases}$$

the explicit exact state is given by

$$y(x,t) = \frac{1}{2}\left(\tilde{y^0}(x-t) + \tilde{y^0}(x+t)\right), \quad 0 \le x \le 1,\ 0 \le t \le 2,$$

PINN. Experiment 2: exact control of the wave equation

Compute $u(t)$ such that the solution $y(x,t)$ of the problem

$$\begin{cases} y_{tt} - y_{xx} = 0, & \text{in } (0,1)\times(0,2) \\ y(x,0) = \sin(\pi x), & \text{in } (0,1) \\ y_t(x,0) = 0 & \text{in } (0,1) \\ y(0,t) = 0, & \text{on } (0,2) \\ y(1,t) = u(t) & \text{on } (0,2) \end{cases}$$

satifies

$$y(x,2) = y_t(x,2) = 0.$$

This problem has an exact solution which can be obtained trough D'Alembert formula. Indeed, by considering the function

$$\tilde{y^0}(x) = \begin{cases} \sin(\pi x) & -1 \leq x \leq 1 \\ 0 & \text{elsewhere,} \end{cases}$$

the explicit exact state is given by

$$y(x,t) = \frac{1}{2}\left(\tilde{y^0}(x-t) + \tilde{y^0}(x+t)\right), \quad 0 \leq x \leq 1,\ 0 \leq t \leq 2,$$

and the exact control is

$$u(t) = \begin{cases} \frac{1}{2} y^0(1-t) & 0 \leq t \leq 1 \\ \\ -\frac{1}{2} y^0(t-1) & 1 \leq t \leq 2. \end{cases}$$

PINN. Experiment 2: exact control of the wave equation

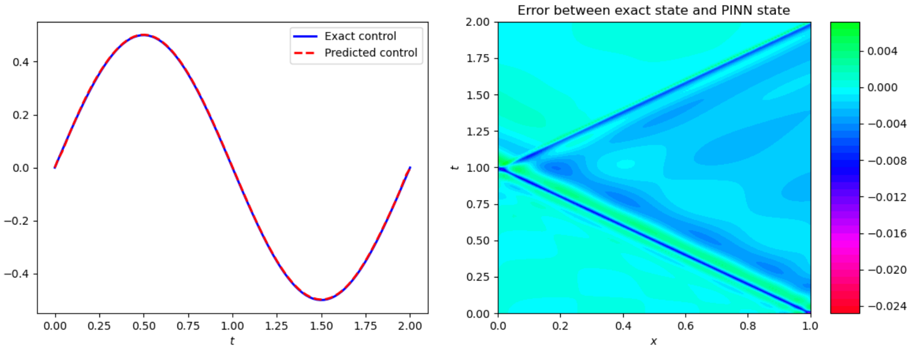


Figure: Experiment 1 (linear wave equation). Comparison between exact control $u(t)$ and PINN (or predicted) control $\hat{u}(t;\boldsymbol{\theta}^*)$ **(left)**, and error between exact state and PINN state, i.e. $y(x,t) - \hat{y}(x,t;\boldsymbol{\theta}^*)$ **(right)**. Neural network composed of 4 hidden layers and 50 neurons in each layer. No regularization. Number of training points $N = 10300$.

DeepONet. Experiment 3: generation of training and test datasets

$$\begin{aligned} \mathcal{G}: \quad L^2(0,1) \times H^{-1}(0,1) & \to L^2(0,2) \\ (y^0, y^1) & \mapsto \mathcal{G}(y^0, y^1) := u \end{aligned}$$

where u is the unique control of minimal L^2-norm of the system

$$\begin{cases} y_{tt} - y_{xx} = 0, & \text{in } (0,1) \times (0,2) \\ y(x,0) = y^0(x), & \text{on } (0,1) \\ y_t(x,0) = y^1(x) & \text{on } (0,1) \\ y(0,t) = 0, & \text{on } (0,2) \\ y(1,t) = u(t) & \text{on } (0,2) \\ y(x,2) = y_t(x,2) = 0, & \text{on } (0,1). \end{cases}$$

DeepONet. Experiment 3: generation of training and test datasets

$$\begin{array}{rll} \mathcal{G}: & L^2(0,1)\times H^{-1}(0,1) & \to L^2(0,2) \\ & (y^0,y^1) & \mapsto \mathcal{G}(y^0,y^1) := u \end{array}$$

where u is the unique control of minimal L^2-norm of the system

$$\begin{cases} y_{tt} - y_{xx} = 0, & \text{in } (0,1)\times(0,2) \\ y(x,0) = y^0(x), & \text{on } (0,1) \\ y_t(x,0) = y^1(x) & \text{on } (0,1) \\ y(0,t) = 0, & \text{on } (0,2) \\ y(1,t) = u(t) & \text{on } (0,2) \\ y(x,2) = y_t(x,2) = 0, & \text{on } (0,1). \end{cases}$$

The **matrix of features** corresponds to initial conditions $(y^0(x), y^1(x))$ evaluated at a number of sensor points $x_j \in (0,1)$. It is computed by sampling a Gaussian random field

$$a(x,\omega) = \sum_{i=1}^{\infty} \sqrt{\lambda_i} e_i(x) \xi_i(\omega), \tag{12}$$

where ξ_i are iid standard Gaussian variables, and $\{\lambda_i, e_i(x)\}_{i=1}^{\infty}$ are the eigenvalues and normalized eigenfuncions of the operator

$$\mathcal{C}(\phi)(x) = \int_0^1 C(x,x')\phi(x')\,dx', \quad C(x,x') = \sigma^2 \exp\left(-\frac{|x-x'|^2}{2\ell}\right)$$

DeepONet. Experiment 3: generation of training and test datasets

The **vector of labels** corresponds to the exact control, which is given by

$$u(t) = \begin{cases} \frac{1}{2} y^0 (1-t) + \frac{1}{2} \int_{1-t}^{1} y^1(s)\, ds & 0 \le t \le 1 \\ \\ -\frac{1}{2} y^0 (t-1) + \frac{1}{2} \int_{t-1}^{1} y^1(s)\, ds & 1 \le t \le 2. \end{cases} \tag{13}$$

A data point is a triplet $((y^0, y^1), t, \mathcal{G}(y^0, y^1)(t))$.

DeepONet. Experiment 3: generation of training and test datasets

The **vector of labels** corresponds to the exact control, which is given by

$$u(t) = \begin{cases} \frac{1}{2} y^0 (1-t) + \frac{1}{2} \int_{1-t}^{1} y^1(s)\, ds & 0 \leq t \leq 1 \\ \\ -\frac{1}{2} y^0 (t-1) + \frac{1}{2} \int_{t-1}^{1} y^1(s)\, ds & 1 \leq t \leq 2. \end{cases} \tag{13}$$

A data point is a triplet $((y^0, y^1), t, \mathcal{G}(y^0, y^1)(t))$.Moreover, a specific input (y^0, y^1) appears in multiple data points with different values of t. Thus,

DeepONet. Experiment 3: generation of training and test datasets

The **vector of labels** corresponds to the exact control, which is given by

$$u(t) = \begin{cases} \frac{1}{2} y^0 (1-t) + \frac{1}{2} \int_{1-t}^{1} y^1(s)\, ds & 0 \leq t \leq 1 \\ \\ -\frac{1}{2} y^0 (t-1) + \frac{1}{2} \int_{t-1}^{1} y^1(s)\, ds & 1 \leq t \leq 2. \end{cases} \tag{13}$$

A data point is a triplet $((y^0, y^1), t, \mathcal{G}(y^0, y^1)(t))$.Moreover, a specific input (y^0, y^1) appears in multiple data points with different values of t. Thus,

$$X_{\text{train}} = \left[\begin{array}{cccc} y_{1,1}^{\text{initial}} & \cdots & y_{1,2m}^{\text{initial}} & t_1 \\ \cdots & \cdots & \cdots & \cdots \\ y_{1,1}^{\text{initial}} & \cdots & y_{1,2m}^{\text{initial}} & t_\ell \\ \hline \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ \hline y_{N,1}^{\text{initial}} & \cdots & y_{N,2m}^{\text{initial}} & t_1 \\ \cdots & \cdots & \cdots & \cdots \\ y_{N,1}^{\text{initial}} & \cdots & y_{N,2m}^{\text{initial}} & t_\ell \end{array}\right], \qquad y_{\text{train}} = \left[\begin{array}{c} u_1(t_1) \\ \cdots \\ u_1(t_\ell) \\ \hline \cdots \\ \cdots \\ \hline u_N(t_1) \\ \cdots \\ u_N(t_\ell) \end{array}\right]$$

DeepONet. Experiment 4: fitting the deeponet model

The implementation in DeepXDE is as follows:

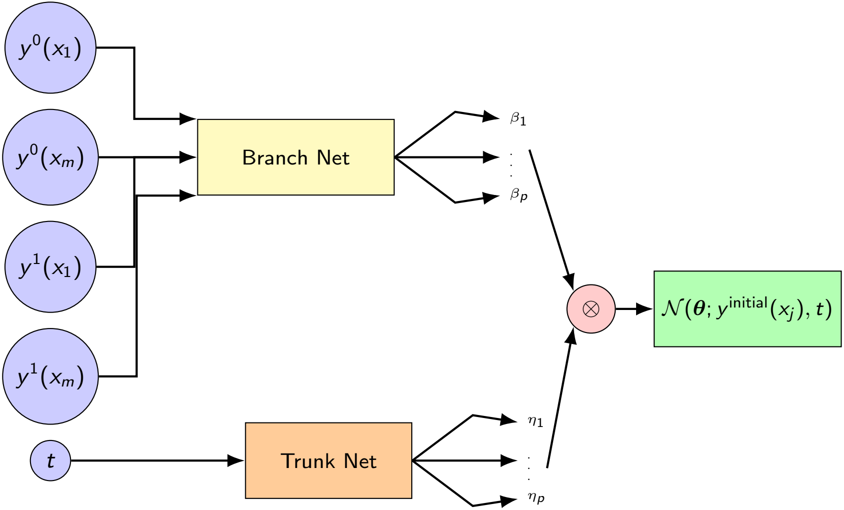
```
    X_train = (X_train[:, :-1], X_train[:, -1:])
    X_test = (X_test[:, :-1], X_test[:, -1:])

    data = dde.data.Triple(
    X_train=X_train, y_train=y_train, X_test=X_test, y_test=
        y_test
    )
```

Lu, L., Jin, P., Pang, G., Zhang, Z., Karniadakis, G.E.:*Learning nonlinear operators via deeponet based on the universal approximation theorem of operators.* **Nature Machine Intelligence 3**(3), 218-229, 2021.

DeepONet. Experiment 4: fitting the deeponet model

Next, we select the architecture of the DeepONet



DeepONet. Experiment 4: fitting the deeponet model

The implementation in DeepXDE is as follows:

```python
m = X_train_input_dimension  # inferred from training data

dim_t = 1  # input dimension of the trunk

p = 10  # output dimension of the branch and trunk nets

net = dde.nn.DeepONet(
[m, 40, p],         # dimensions of the branch net
[dim_t, 40, p],     # dimensions of the trunk net
"relu",             # activation function
"Glorot normal",    # initialization of parameters
)

model = dde.Model(data, net)

model.compile("adam", lr=0.001)

losshistory, train_state = model.train(iterations=ITERATIONS)

dde.saveplot(losshistory, train_state, issave=True, isplot=True)
```

DeepONet. Experiment 4: Smooth initial conditions

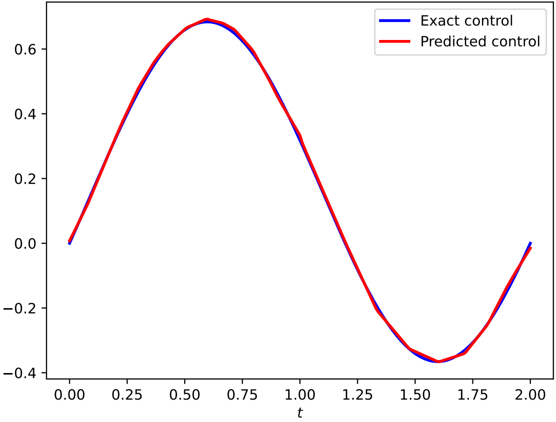


Figure: Experiment 4: wave equation. Exact versus predicted solutions for THE smooth initial conditions $y^0(x) = y^1(x) = \sin(\pi x)$. $n_{functions} = 10^4$, $(\ell_0, \ell_1) = (0.25, 0.125)$, $n_{sensors} = 101$, $p = 40$. Relative error $\approx 1\%$.

DeepONet. Experiment 4: Unsmooth initial conditions

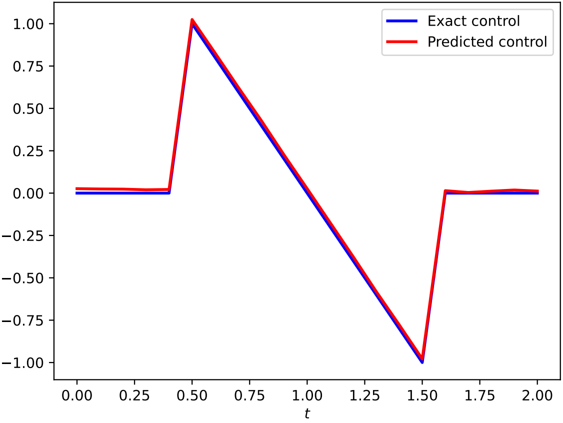


Figure: Experiment 4: wave equation. Exact versus predicted solutions for the unsmooth initial conditions $y^0(x) = \begin{cases} 4x, & 0 \leq x \leq 0.5 \\ 0, & 0.5 < x \leq 1 \end{cases}, \quad y^1(x) = 0.$ $n_{functions} = 10^4$, $(\ell_0, \ell_1) = (0.03125, 0.03125)$, $p = 100$. $n_{sensors} = 11$. Relative error $\approx 4\%$.

Work supported by

- Fundación Séneca (Agencia de Ciencia y Tecnología de la Región de Murcia (Spain)) through the programme for the development of scientific and technical research by competitive groups (21996/PI/22).

- Grants PID2022-141957OA-C22 funded by MCIN/AEI/10.13039/501100011033, by RDF A way of making Europe

Thank you for your attention !