Breaking the curse of dimensionality in some control and parametric problems for PDEs

Francisco Periago

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

Kick-off workshop of Working Group 2 "ML for CT"

Cost Action 24136
Interactions between Control Theory and Machine Learning

Working Group 2:
Machine Learning for Control Theory

Outline

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

Outline

1. **Machine Learning basis.** Where and why Machine-Learning-based methods may be useful in the numerical approximation of PDEs-based models?
2. **Physics Informed Neural Networks (PINNs).** Solving an exact controllability problem via PINNs

Outline

1. **Machine Learning basis.** Where and why Machine-Learning-based methods may be useful in the numerical approximation of PDEs-based models?
2. **Physics Informed Neural Networks (PINNs).** Solving an exact controllability problem via PINNs
3. **Numerical implementation via DeepXDE.** A brief introduction

Outline

1. **Machine Learning basis.** Where and why Machine-Learning-based methods may be useful in the numerical approximation of PDEs-based models?
2. **Physics Informed Neural Networks (PINNs).** Solving an exact controllability problem via PINNs
3. **Numerical implementation via DeepXDE.** A brief introduction
4. **Deep Operator Network (DeepONet).** Learning the controllability map via DeepONet

Outline

1. **Machine Learning basis.** Where and why Machine-Learning-based methods may be useful in the numerical approximation of PDEs-based models?
2. **Physics Informed Neural Networks (PINNs).** Solving an exact controllability problem via PINNs
3. **Numerical implementation via DeepXDE.** A brief introduction
4. **Deep Operator Network (DeepONet).** Learning the controllability map via DeepONet
5. **The curse of dimensionality**. The Everest of challenges!

Outline

1. **Machine Learning basis.** Where and why Machine-Learning-based methods may be useful in the numerical approximation of PDEs-based models?
2. **Physics Informed Neural Networks (PINNs).** Solving an exact controllability problem via PINNs
3. **Numerical implementation via DeepXDE.** A brief introduction
4. **Deep Operator Network (DeepONet).** Learning the controllability map via DeepONet
5. **The curse of dimensionality**. The Everest of challenges!
6. **Some related Open Problems**

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 $\mathcal{H}_m$.

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 $S = \{(\boldsymbol{x}_i, \boldsymbol{y}_i = f^*(\boldsymbol{x}_i)), 1 \leq i \leq n\}$

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**

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

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**

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

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

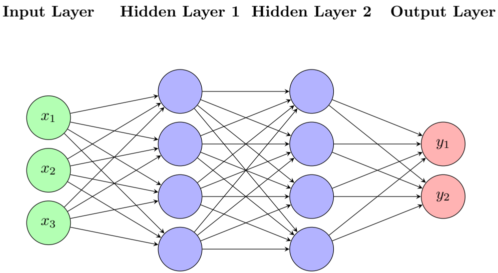
The overall objective is to minimize the **generalization error**

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

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

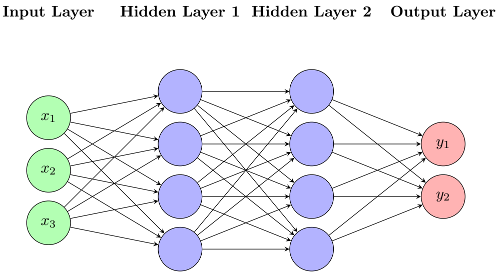
Hypothesis space: an example

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



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)**.



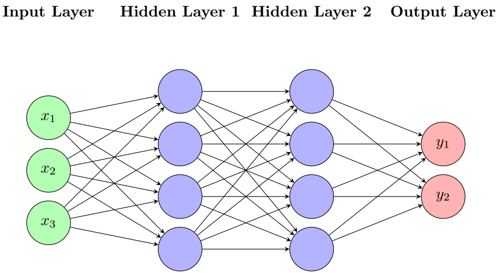
To each input $\boldsymbol{x} \in \mathbb{R}^d$ it associates the output $\boldsymbol{y} = f_m(\boldsymbol{x}) := \boldsymbol{x}^m$ defined by

$$\begin{cases} \text{input layer:} & \boldsymbol{x}^0 = \boldsymbol{x}, \\ \text{hidden layers:} & \boldsymbol{x}^{k+1} = \sigma\left(\omega^{k+1}\boldsymbol{x}^k + b^{k+1}\right) \quad \text{for } k = 0, 1, \cdots, m-2 \\ \text{output layer:} & \boldsymbol{x}^m = \omega^m \boldsymbol{x}^{m-1} + b^m, \end{cases}$$

Hypothesis space: an example

Cost Action 24136
Interactions between Control Theory and Machine Learning
Working Group 2: Machine Learning for Control Theory

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



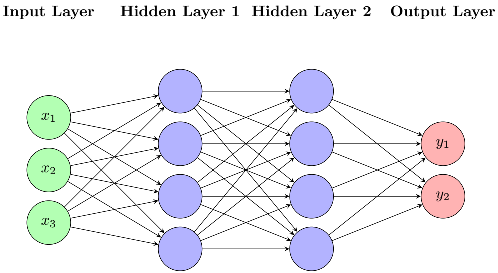
To each input $\boldsymbol{x} \in \mathbb{R}^d$ it associates the output $\boldsymbol{y} = f_m(\boldsymbol{x}) := \boldsymbol{x}^m$ defined by

$$
\begin{cases}
\text{input layer:} & \boldsymbol{x}^0 = \boldsymbol{x}, \\
\text{hidden layers:} & \boldsymbol{x}^{k+1} = \sigma\left(\omega^{k+1}\boldsymbol{x}^k + b^{k+1}\right) \quad \text{for } k = 0, 1, \cdots, m-2 \\
\text{output layer:} & \boldsymbol{x}^m = \omega^m \boldsymbol{x}^{m-1} + b^m,
\end{cases}
$$

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

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)**.

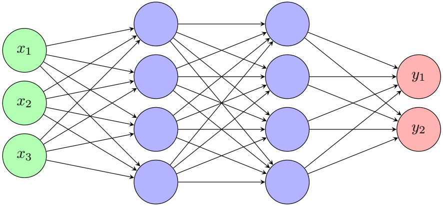


To each input $\boldsymbol{x} \in \mathbb{R}^d$ it associates the output $\boldsymbol{y} = f_m(\boldsymbol{x}) := \boldsymbol{x}^m$ defined by

$$\begin{cases} \text{input layer:} & \boldsymbol{x}^0 = \boldsymbol{x}, \\ \text{hidden layers:} & \boldsymbol{x}^{k+1} = \sigma\left(\omega^{k+1}\boldsymbol{x}^k + b^{k+1}\right) \quad \text{for } k = 0, 1, \cdots, m-2 \\ \text{output layer:} & \boldsymbol{x}^m = \omega^m \boldsymbol{x}^{m-1} + b^m, \end{cases}$$

- 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,

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



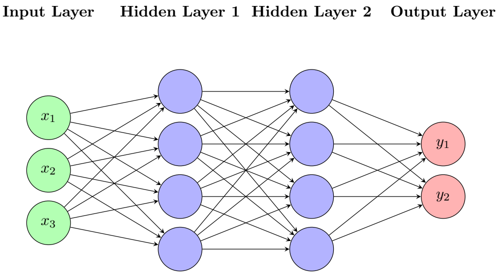
To each input $\boldsymbol{x} \in \mathbb{R}^d$ it associates the output $\boldsymbol{y} = f_m(\boldsymbol{x}) := \boldsymbol{x}^m$ defined by

$$\begin{cases} \text{input layer:} & \boldsymbol{x}^0 = \boldsymbol{x}, \\ \text{hidden layers:} & \boldsymbol{x}^{k+1} = \sigma\left(\omega^{k+1}\boldsymbol{x}^k + b^{k+1}\right) \quad \text{for } k = 0, 1, \cdots, m-2 \\ \text{output layer:} & \boldsymbol{x}^m = \omega^m \boldsymbol{x}^{m-1} + b^m, \end{cases}$$

- 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,

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

$$\begin{cases} \text{input layer:} & \boldsymbol{x}^0 = \boldsymbol{x}, \\ \text{hidden layers:} & \boldsymbol{x}^{k+1} = \sigma\left(\omega^{k+1}\boldsymbol{x}^k + b^{k+1}\right) \quad \text{for } k = 0, 1, \cdots, m-2 \\ \text{output layer:} & \boldsymbol{x}^m = \omega^m \boldsymbol{x}^{m-1} + b^m, \end{cases}$$

- 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**

Hypothesis space: an example

More on the **activation function**:

More on the **activation function:**

By abuse of notation, $\sigma : \mathbb{R}^d \to \mathbb{R}^d$ is defined component-wise by

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

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 $\sigma(s) = \tanh(s)$.

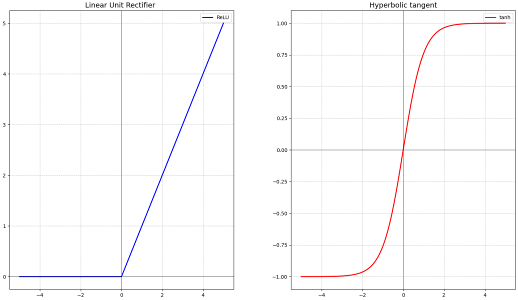


Figure: Linear Unit Rectifier (left) and hyperbolic tangent (right).

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

ML opens a door to tackle real problems

More situations that lead to very large d:

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

ML opens a door to tackle real 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/

ML opens a door to tackle real 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

Part II

Physics Informed Neural Networks (PINNs)

A toy model: null control of the wave equation

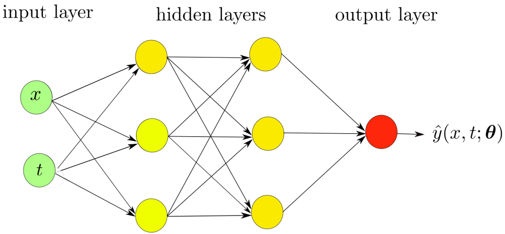
$$
\begin{cases}
y_{tt} - \Delta y = 0, & \text{in } Q_T \\
y(x,0) = y^0(x), & \text{in } \Omega \\
y_t(x,0) = y^1(x) & \text{in } \Omega \\
y(x,t) = 0, & \text{on } \Gamma_D \times (0,T) \\
y(x,t) = u(x,t) & \text{on } \Gamma_C \times (0,T)
\end{cases}
$$

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

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

Step 1: Neural network

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



$$
\begin{cases}
\text{input layer:} & \boldsymbol{x}^0(\boldsymbol{x}) = \boldsymbol{x} = (x, t) \in \mathbb{R}^{d+1} \\
\text{hidden layers:} & \boldsymbol{x}^{k+1} = \sigma\left(\omega^{k+1}\boldsymbol{x}^k + b^{k+1}\right) \quad \text{for } k = 0, 1, \cdots, m-2 \\
\text{output layer:} & \boldsymbol{x}^m = \omega^m \boldsymbol{x}^{m-1} + b^m,
\end{cases}
$$

- $\omega^\ell \in \mathbb{R}^{N_\ell \times N_{\ell-1}}$ and $b^\ell \in \mathbb{R}^{N_\ell}$ are, respectively, the weights and biases so that $\boldsymbol{\theta} = \left\{\omega^\ell, b^\ell\right\}_{1 \leq \ell \leq m}$ are the parameters of the neural network, and
- σ is an activation function, e.g. $\sigma(s) = \tanh(s)$

Step 2: Training dataset

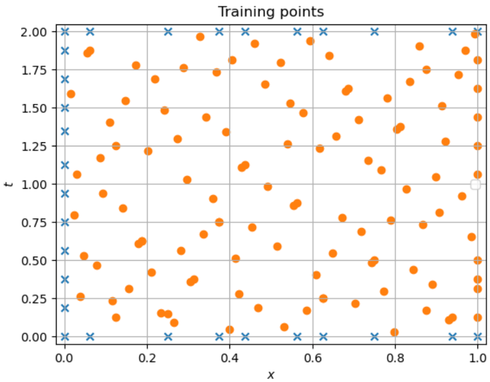


Figure: Illustration of a training dataset (based on Sobol points) in the domain $Q_2 = (0,1) \times (0,2)$. Interior points are marked with circles and boundary points in blue color. $\mathbf{x}_j = (x_j, t_j)$ are the features.

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

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

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

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

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

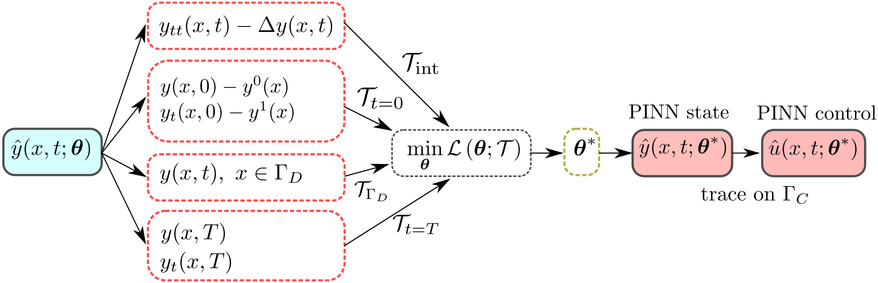
Step 4: **Training process**

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

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

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

To sump up:



Why MLP is a suitable prediction model ?

Why MLP is a suitable prediction model ?

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

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

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.

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

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

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

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

Training error

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

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

Generalization error for control and state

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

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.

Part III

Numerical Implementation via DeepXDE

A few words on DeepXDE

Lu, Lu and Meng, Xuhui and Mao, Zhiping and Karniadakis, George Em: *DeepXDE: A deep learning library for solving differential equations* **SIAM Review 63**(1), 208–228, 2021.

- DeepXDE is a Python library for scientific machine learning and physics-informed learning.

A few words on DeepXDE

Lu, Lu and Meng, Xuhui and Mao, Zhiping and Karniadakis, George Em: *DeepXDE: A deep learning library for solving differential equations* **SIAM Review 63**(1), 208–228, 2021.

- DeepXDE is a Python library for scientific machine learning and physics-informed learning.
- It was developed by Lu Lu under the supervision of Prof. George Karniadakis at Brown University.

A few words on DeepXDE

Lu, Lu and Meng, Xuhui and Mao, Zhiping and Karniadakis, George Em: *DeepXDE: A deep learning library for solving differential equations* **SIAM Review 63**(1), 208–228, 2021.

- DeepXDE is a Python library for scientific machine learning and physics-informed learning.
- It was developed by Lu Lu under the supervision of Prof. George Karniadakis at Brown University.
- It is currently maintained by Lu Lu at Yale University.
  https://github.com/lululxvi/deepxde

A few words on DeepXDE

Lu, Lu and Meng, Xuhui and Mao, Zhiping and Karniadakis, George Em: *DeepXDE: A deep learning library for solving differential equations* **SIAM Review 63**(1), 208–228, 2021.

- DeepXDE is a Python library for scientific machine learning and physics-informed learning.
- It was developed by Lu Lu under the supervision of Prof. George Karniadakis at Brown University.
- It is currently maintained by Lu Lu at Yale University.
  https://github.com/lululxvi/deepxde
- For an introductory course focused on control of PDEs you may visit
  https://github.com/fperiago/pinn_deeponet_for_beginners

Experiment 1: 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

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

Experiment 1: 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 by means of 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

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

Experiment 1: 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 by means of D'Alembert formula. Indeed, by considering the function

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

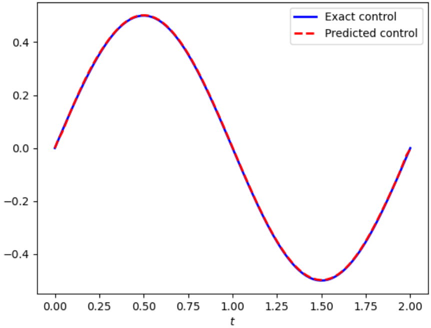
the explicit exact state is given by

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

and the exact control is

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

Experiment 1: control of the wave equation



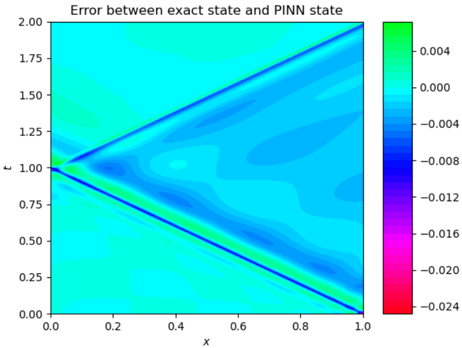


Figure: Experiment 1 (linear wave equation). Comparison between exact control $u(t)$ and PINN (or predicted) control $\hat{u}(t;\boldsymbol{\theta}^*)$ **(left)**, and error between exact state and PINN state, i.e. $y(x,t) - \hat{y}(x,t;\boldsymbol{\theta}^*)$ **(right)**. Neural network composed of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

Part IV

Deep Operator Network (DeepONet)

Learning the controllability map

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

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

Learning the controllability map

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: to approximate the controllabilty mapping

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

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

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: to approximate the controllabilty mapping

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

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

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

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

- **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,

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

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,

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

The corresponding **labels** are

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

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,
  $$\{(y_\ell^{\text{initial}}; t_\ell), \quad 1 \leq \ell \leq N\}$$
  The corresponding **labels** are
  $$\{u_\ell = u(y_\ell^{\text{initial}}; t_\ell), \quad 1 \leq \ell \leq N\},$$
  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
  $$\mathcal{N}(\boldsymbol{\theta}; (y^{\text{initial}}(x_j); t)) := \sum_{k=1}^{p} \sum_{i=1}^{n} c_i^k \sigma \left(\sum_{j=1}^{m} \xi_{ij}^k y^{\text{initial}}(x_j) + \theta_i^k \right) \cdot \sigma(w_k \cdot t + \eta_k)$$
  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,

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

  The corresponding **labels** are

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

  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

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

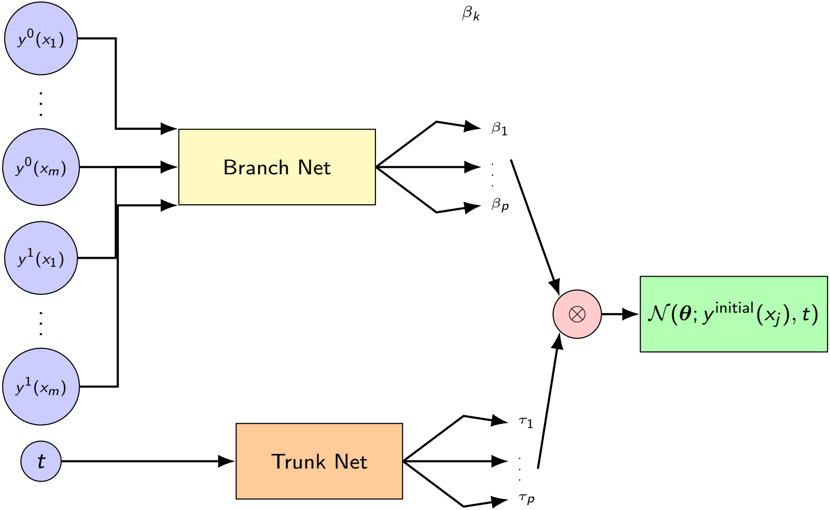
  where $\boldsymbol{\theta} = (c_i^k, \xi_{ij}^k, \theta_i^k, w_k, \eta_k)$ is the set of parameters of the net.

- **Loss function: Mean Squared Error (MSE)**

$$\text{MSE}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{\ell=1}^{N} |\mathcal{N}(\boldsymbol{\theta}; (y_\ell^{\text{initial}}; t_\ell)) - u_\ell|^2$$

DeepONet's architecture

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



DeepONet's architecture

From a different perspective, the approximation scheme may be decomposed as

$$\begin{array}{ccc} X & \xrightarrow{\mathcal{G}} & Y \\ \downarrow \mathcal{E} & & \uparrow \mathcal{R} \\ \mathbb{R}^{2m} & \xrightarrow{\mathcal{A}} & \mathbb{R}^{p} \end{array}$$

where:

DeepONet's architecture

From a different perspective, the approximation scheme may be decomposed as

$$
\begin{array}{ccc}
X & \xrightarrow{\mathcal{G}} & Y \\
\downarrow \mathcal{E} & & \uparrow \mathcal{R} \\
\mathbb{R}^{2m} & \xrightarrow{\mathcal{A}} & \mathbb{R}^{p}
\end{array}
$$

where:

- $\mathcal{E} : (y^0, y^1) \mapsto y^{initial}$ is an *encoder*,

DeepONet's architecture

From a different perspective, the approximation scheme may be decomposed as

$$\begin{array}{ccc} X & \xrightarrow{\mathcal{G}} & Y \\ \downarrow \mathcal{E} & & \uparrow \mathcal{R} \\ \mathbb{R}^{2m} & \xrightarrow{\mathcal{A}} & \mathbb{R}^{p} \end{array}$$

where:

- $\mathcal{E} : (y^0, y^1) \mapsto y^{initial}$ is an *encoder*,
- $\mathcal{A} : y^{initial} \mapsto \beta_k(y^{initial})$ is a finite dimensional *approximation operator*, and

DeepONet's architecture

From a different perspective, the approximation scheme may be decomposed as

$$\begin{array}{ccc} X & \xrightarrow{\mathcal{G}} & Y \\ \downarrow \mathcal{E} & & \uparrow \mathcal{R} \\ \mathbb{R}^{2m} & \xrightarrow{\mathcal{A}} & \mathbb{R}^{p} \end{array}$$

where:

- $\mathcal{E} : (y^0, y^1) \mapsto y^{initial}$ is an *encoder*,
- $\mathcal{A} : y^{initial} \mapsto \beta_k(y^{initial})$ is a finite dimensional *approximation operator*, and
- $\mathcal{R} : \beta_k(y^{initial}) \mapsto \sum_{k=1}^{p} \beta_k(y^{\text{initial}})\tau_k(t)$ is a *reconstruction operator.*

Thus,

$$\mathcal{G} \approx \mathcal{R} \circ \mathcal{A} \circ \mathcal{E}.$$

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

$$\mu \sim \sum_{j=1}^{\infty} \sqrt{\lambda_j} \xi_j(\omega) \varphi_j(x) \tag{3}$$

We take $\xi_j \sim \mathcal{N}(0,1)$ i.i.d. standard Gaussian variables, and (λ_j, φ_j) are the eigenvalues and normalized eigenfuncions of the operator

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

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

$$\mu \sim \sum_{j=1}^{\infty} \sqrt{\lambda_j}\xi_j(\omega)\varphi_j(x) \tag{3}$$

We take $\xi_j \sim \mathcal{N}(0,1)$ i.i.d. standard Gaussian variables, and (λ_j, φ_j) are the eigenvalues and normalized eigenfuncions of the operator

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

Thus, we truncate (3) 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_j^0, y_j^1) and store the corresponding y_{jk}^{initial} 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

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

evaluated at a finite selection of times t. Precisely,

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

Experiment 2: Smooth initial conditions

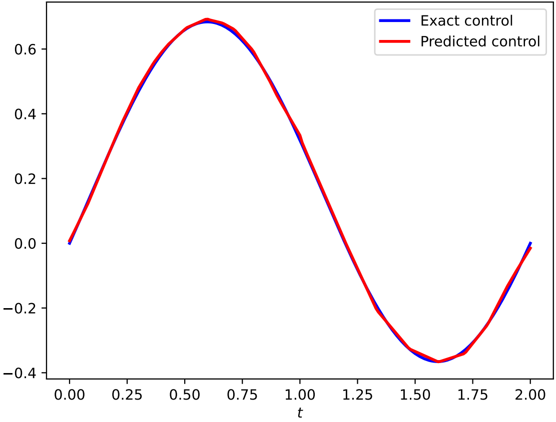


Figure: Experiment 2: wave equation. Exact versus predicted solutions for the smooth initial conditions $y^0(x) = y^1(x) = \sin(\pi x)$. $n_{functions} = 10^4$, $(\ell_0, \ell_1) = (0.25, 0.125)$, $n_{sensors} = 101$, $p = 40$. Relative error $\approx 1\%$.

Experiment 3: Unsmooth initial conditions

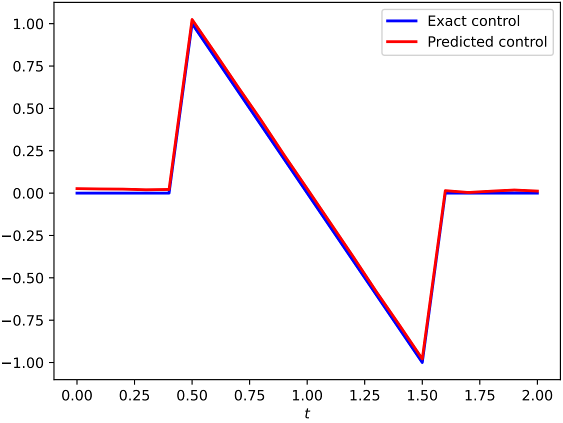


Figure: Experiment 3: wave equation. Exact versus predicted solutions for the unsmooth initial conditions $y^0(x) = \begin{cases} 4x, & 0 \leq x \leq 0.5 \\ 0, & 0.5 < x \leq 1 \end{cases}, \quad y^1(x) = 0.$

$n_{functions} = 10^4$, $(\ell_0, \ell_1) = (0.03125, 0.03125)$, $p = 100$. $n_{sensors} = 11$. Relative error $\approx 4\%$.

Part V

The curse of Dimensionality (CoD)

Curse of dimensionality

Let $f \in H^\alpha$ and let f_m be a two-layer approximation of f.

Curse of dimensionality

Let $f \in H^\alpha$ and let f_m be a two-layer approximation of f.
Approximation error (due to the choice of $\mathcal{H}_m$): typically

$$\|f - f_m\|_{L^2} \leq C m^{-\alpha/d} \|f\|_{H^\alpha}$$

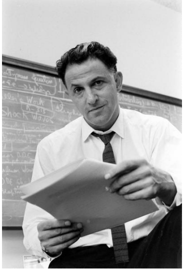
If $m^{-\alpha/d} = 0.1$, then $m = 10^{d/\alpha} = 10^d$, if $\alpha = 1$. **Curse of Dimensionality**

Curse of dimensionality

Let $f \in H^\alpha$ and let f_m be a two-layer approximation of f.
Approximation error (due to the choice of $\mathcal{H}_m$): typically

$$\|f - f_m\|_{L^2} \leq Cm^{-\alpha/d} \|f\|_{H^\alpha}$$

If $m^{-\alpha/d} = 0.1$, then $m = 10^{d/\alpha} = 10^d$, if $\alpha = 1$. **Curse of Dimensionality**



... a computational solution ... is quite definitely not routine when the number of variables is large. All this may be subsumed under the heading
Curse of Dimensionality.
R. Bellman. Dynamic Programming, 1957.

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

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

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

then **approximation error** is

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

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

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

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

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

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

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

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

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

- **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

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

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.

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

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
  $$\begin{cases} \partial_t y + \partial_x (f(y)) = 0 \\ y(0) = y^0 \end{cases}$$

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
  $$\begin{cases} \partial_t y + \partial_x (f(y)) = 0 \\ y(0) = y^0 \end{cases}$$
- bounded linear operators $\mathcal{G} : L^2(D) \to L^2(U)$

Application to our controllability setting

Set $\mathbb{T} = [0, \pi]$. Consider the operator

$$\mathcal{G} : L^2(\mathbb{T}) \times H^{-1}(\mathbb{T}) \to L^2(0, 4\pi), \quad (y^0, y^1) \mapsto \mathcal{G}(y^0, y^1) = u$$

Application to our controllability setting

Set $\mathbb{T} = [0, \pi]$. Consider the operator

$$\mathcal{G} : L^2(\mathbb{T}) \times H^{-1}(\mathbb{T}) \to L^2(0, 4\pi), \quad (y^0, y^1) \mapsto \mathcal{G}(y^0, y^1) = u$$

We take $X = C(\mathbb{T}) \times C(\mathbb{T})$. For the sake of simplicity, we consider the measure $\mu \times \mu$, where μ is the one dimensional measure associated with squared exponential covariance funcion

$$C(x, x') = e^{-\frac{|x-x'|^2}{2\ell^2}}$$

Application to our controllability setting

Set $\mathbb{T} = [0, \pi]$. Consider the operator

$$\mathcal{G} : L^2(\mathbb{T}) \times H^{-1}(\mathbb{T}) \to L^2(0, 4\pi), \quad (y^0, y^1) \mapsto \mathcal{G}(y^0, y^1) = u$$

We take $X = C(\mathbb{T}) \times C(\mathbb{T})$. For the sake of simplicity, we consider the measure $\mu \times \mu$, where μ is the one dimensional measure associated with squared exponential covariance funcion

$$C(x, x') = e^{-\frac{|x-x'|^2}{2\ell^2}}$$

The corresponding eigenpairs are given by

$$\phi_k(x) = e^{-ikx}, \quad \lambda_k = \sqrt{2\pi}\ell e^{-(\ell k)^2/2}$$

Application to our controllability setting

Set $\mathbb{T} = [0, \pi]$. Consider the operator

$$\mathcal{G} : L^2(\mathbb{T}) \times H^{-1}(\mathbb{T}) \to L^2(0, 4\pi), \quad (y^0, y^1) \mapsto \mathcal{G}(y^0, y^1) = u$$

We take $X = C(\mathbb{T}) \times C(\mathbb{T})$. For the sake of simplicity, we consider the measure $\mu \times \mu$, where μ is the one dimensional measure associated with squared exponential covariance funcion

$$C(x, x') = e^{-\frac{|x-x'|^2}{2\ell^2}}$$

The corresponding eigenpairs are given by

$$\phi_k(x) = e^{-ikx}, \quad \lambda_k = \sqrt{2\pi}\ell e^{-(\ell k)^2/2}$$

The key ingredient here is that in the decomposition

$$\begin{array}{ccc} X & \xrightarrow{\mathcal{G}} & Y \\ \downarrow \mathcal{E} & & \uparrow \mathcal{R} \\ \mathbb{R}^{2m} & \xrightarrow{\mathcal{A}} & \mathbb{R}^p \end{array}$$

the error associated with the linear operator $\mathcal{A}$ vanishes since if σ is an ReLU activation function, then

$$\mathcal{A}x + b = \sigma(\mathcal{A}x + b) - \sigma(-(\mathcal{A}x + b)).$$

Theorem (DeepONet approximation of controllability systems)

Consider the control system for $x \in \Omega \subset \mathbb{R}^d$ *and* $t > 0$*:*

$$\begin{cases} y' + Ay = 1_\omega u \\ y(0) = y^0 \end{cases}$$

and assume that for some $T > 0$ *there exists a unique control* $u \in L^2(\omega \times (0,T))$ *such that* $y(T) = 0$.
Consider the operator $\mathcal{G} : y^0 \to u$*. Then, for any fixed* $\varepsilon > 0$ *and* $\sigma > 0$*, there exists a DeepONet* $\mathcal{N} = (\beta, \tau)$ *achieving an approximation error*

$$\mathcal{E}_{approx} = \|\mathcal{G} - \mathcal{N}\|_{L^2(\mu)} \lesssim \varepsilon$$

and such that

$$size(\beta) \lesssim \log(\varepsilon^{-1}), \quad depth(\beta) \lesssim 1, \quad p \sim \log(\varepsilon^{-1})^d, \quad m \sim \log(\varepsilon^{-1})^{2d+\sigma}.$$

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

Estimates for generalization error for DeepONet

Boundedness assumption: there exists $\psi : L^2(D) \to [0, +\infty[$ such that

$$|\mathcal{G}(y)(t)| \leq \psi(y), \qquad \sup_{\theta \in [-B,B]^{d_\theta}} |\mathcal{N}_\theta(y)(t)| \leq \psi(y), \quad \forall y \in L^2(D), \forall t \in U,$$

and there exists $C, \kappa > 0$ such that

$$\psi(y) \leq C \left(1 + \|y\|_{L^2(D)}\right)^\kappa .$$

Lipschitz continuity assumption: there exists $\Phi : L^2(D) \to [0, +\infty[$ such that

$$|\mathcal{N}_\theta(y)(t) - \mathcal{N}_{\theta'}(y)(t)| \leq \Phi(y)\|\theta - \theta'\|_{\ell^\infty}, \quad \forall y \in L^2(D), \forall t \in U,$$

and there exists $C, \kappa > 0$ such that

$$\Phi(y) \leq C \left(1 + \|y\|_{L^2(D)}\right)^\kappa .$$

Theorem (Bound for generalization error)

Let $\mathcal{N}_\mathcal{F}$ and $\mathcal{N}_\mathcal{T}$ be as before. If the above two assumptions hold, then

$$\mathbb{E}\left[\mathcal{L}(\mathcal{N}_\mathcal{T}) - \mathcal{L}(\mathcal{N}_\mathcal{F})\right] \leq \frac{C}{\sqrt{N}} \left(1 + C d_\theta \log\left(CB\sqrt{N}\right)\right)^{2\kappa + \frac{1}{2}} \tag{4}$$

where the constant $C = C(\mu, \psi, \Phi)$ is independent of B and d_θ.

Part VI

Open problems

Open problem 1

Consider a control system:

$$\begin{cases} y' + Ay = Bu \\ y(0) = y^0 \end{cases}$$

Open problem 1

Consider a control system:

$$\begin{cases} y' + Ay = Bu \\ y(0) = y^0 \end{cases}$$

The mapping

$$y^0 \mapsto \mathcal{R}(T, y^0) = \left\{ y(T, y^0, u) \quad : \quad u \in \mathcal{U} \right\}$$

is set-valued.

Open problem 1

Consider a control system:

$$\begin{cases} y' + Ay = Bu \\ y(0) = y^0 \end{cases}$$

The mapping

$$y^0 \mapsto \mathcal{R}(T, y^0) = \left\{ y(T, y^0, u) \quad : \quad u \in \mathcal{U} \right\}$$

is set-valued.

Open problem 1: **learn this map with ML tools**

Open problem 1

Consider a control system:

$$\begin{cases} y' + Ay = Bu \\ y(0) = y^0 \end{cases}$$

The mapping

$$y^0 \mapsto \mathcal{R}(T, y^0) = \left\{ y(T, y^0, u) \quad : \quad u \in \mathcal{U} \right\}$$

is set-valued.

Open problem 1: learn this map with ML tools

Hint: a class of set-valued neural networks called **SIMONet** has been introduced in

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

Open problem 1

Consider a control system:

$$\begin{cases} y' + Ay = Bu \\ y(0) = y^0 \end{cases}$$

The mapping

$$y^0 \mapsto \mathcal{R}(T, y^0) = \left\{ y(T, y^0, u) \quad : \quad u \in \mathcal{U} \right\}$$

is set-valued.

Open problem 1: learn this map with ML tools

Hint: a class of set-valued neural networks called **SIMONet** has been introduced in

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

Required skills for applicants: Good command of Python and a basic knowledge of control theory

Open problem 1

Consider a control system:

$$\begin{cases} y' + Ay = Bu \\ y(0) = y^0 \end{cases}$$

The mapping

$$y^0 \mapsto \mathcal{R}(T, y^0) = \left\{ y(T, y^0, u) \quad : \quad u \in \mathcal{U} \right\}$$

is set-valued.

Open problem 1: learn this map with ML tools

Hint: a class of set-valued neural networks called **SIMONet** has been introduced in

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

Required skills for applicants: Good command of Python and a basic knowledge of control theory

Contact person: Francisco Periago. Email: f.periago@upct.es

Open problem 2

A two-layer neural network may be represented as

$$f_m(\boldsymbol{x}) = \frac{1}{m} \sum_{j=1}^{m} a_j \sigma \left(\boldsymbol{\omega}_j^T \boldsymbol{x} + b_j \right) \tag{5}$$

where $(a_j, \boldsymbol{\omega}_j, b_j)$ are the parameters and σ is the activation function.

Open problem 2

A two-layer neural network may be represented as

$$f_m(\boldsymbol{x}) = \frac{1}{m}\sum_{j=1}^{m} a_j \sigma\left(\boldsymbol{\omega}_j^T \boldsymbol{x} + b_j\right) \tag{5}$$

where $(a_j, \boldsymbol{\omega}_j, b_j)$ are the parameters and σ is the activation function.
Where does this expression come from?

Open problem 2

A two-layer neural network may be represented as

$$f_m(\boldsymbol{x}) = \frac{1}{m}\sum_{j=1}^{m} a_j \sigma\left(\boldsymbol{\omega}_j^T \boldsymbol{x} + b_j\right) \tag{5}$$

where $(a_j, \boldsymbol{\omega}_j, b_j)$ are the parameters and σ is the activation function. **Where does this expression come from?** Passing to the limit when the width of the hidden layer goes to infinity in (5) we get the representation formula

$$f_\rho(\boldsymbol{x}) = \int_{\mathbb{R}^{d+2}} a\sigma\left(\boldsymbol{\omega}^T \boldsymbol{x} + b\right)\, \rho\left(da, d\boldsymbol{\omega}, db\right) = \mathbb{E}_\rho\left[a\sigma(\boldsymbol{\omega}^T \boldsymbol{x})\right]$$

Open problem 2

A two-layer neural network may be represented as

$$f_m(\boldsymbol{x}) = \frac{1}{m}\sum_{j=1}^{m} a_j \sigma\left(\boldsymbol{\omega}_j^T \boldsymbol{x} + b_j\right) \tag{5}$$

where $(a_j, \boldsymbol{\omega}_j, b_j)$ are the parameters and σ is the activation function.
Where does this expression come from? Passing to the limit when the width of the hidden layer goes to infinity in (5) we get the representation formula

$$f_\rho(\boldsymbol{x}) = \int_{\mathbb{R}^{d+2}} a\sigma\left(\boldsymbol{\omega}^T \boldsymbol{x} + b\right)\, \rho\left(da, d\boldsymbol{\omega}, db\right) = \mathbb{E}_\rho\left[a\sigma(\boldsymbol{\omega}^T \boldsymbol{x})\right]$$

Open problem 2

A two-layer neural network may be represented as

$$f_m(\boldsymbol{x}) = \frac{1}{m}\sum_{j=1}^{m} a_j \sigma\left(\boldsymbol{\omega}_j^T \boldsymbol{x} + b_j\right) \tag{5}$$

where $(a_j, \boldsymbol{\omega}_j, b_j)$ are the parameters and σ is the activation function. **Where does this expression come from?** Passing to the limit when the width of the hidden layer goes to infinity in (5) we get the representation formula

$$f_\rho(\boldsymbol{x}) = \int_{\mathbb{R}^{d+2}} a\sigma\left(\boldsymbol{\omega}^T \boldsymbol{x} + b\right)\rho\left(da, d\boldsymbol{\omega}, db\right) = \mathbb{E}_\rho\left[a\sigma(\boldsymbol{\omega}^T \boldsymbol{x})\right]$$

Definition (Barron space)

Let $\sigma(x) = \max\{0, x\}$ be the ReLU activation function. Barron space $\mathcal{B}$ is the one composed of continuous functions that admit a representation in the form

$$f_\rho(\boldsymbol{x}) = \int_{\mathbb{R}^{d+2}} a\sigma\left(\boldsymbol{\omega}^T \boldsymbol{x} + b\right)\rho\left(da, d\boldsymbol{\omega}, db\right) = \mathbb{E}_\rho\left[a\sigma(\boldsymbol{\omega}^T \boldsymbol{x})\right],$$

ρ being a probability distribution on $\left(\mathbb{R} \times \mathbb{R}^d \times \mathbb{R}, \Sigma\right)$, and, in addition, its norm

$$\|f\|_{\mathcal{B}} := \inf_{\rho} \max_{(a,\boldsymbol{w},b)\in \text{supp }(\rho)} |a|\left(\|\boldsymbol{\omega}\|_1 + |b|\right)$$

is finite.

Open problem 2

Theorem (Direct approximation)

For any $f \in \mathcal{B}$ and $m \in \mathbb{N}$, there exists a two-layer neural network $f_m(\boldsymbol{x};\boldsymbol{\theta}) = \frac{1}{m}\sum_{j=1}^m a_j \sigma\left(\boldsymbol{\omega}_j^T \boldsymbol{x} + b_j\right)$, with m neurons $\boldsymbol{\theta} = (a_j, \boldsymbol{\omega}_j, b_j)$ such that

$$\|f - f_m\|_{L^2}^2 \leq 3\frac{\|f\|_{\mathcal{B}}^2}{m}.$$

Moreover,

$$\|\boldsymbol{\theta}\| := \frac{1}{m}\sum_{j=1}^m |a_j| \left(\|\boldsymbol{\omega}_j\|_1 + |b_j|\right) \leq 2\|f\|_{\mathcal{B}}.$$

Open problem 2

Theorem (Direct approximation)

For any $f \in \mathcal{B}$ and $m \in \mathbb{N}$, there exists a two-layer neural network $f_m(\boldsymbol{x};\boldsymbol{\theta}) = \frac{1}{m}\sum_{j=1}^{m} a_j \sigma\left(\boldsymbol{\omega}_j^T \boldsymbol{x} + b_j\right)$, with m neurons $\boldsymbol{\theta} = (a_j, \boldsymbol{\omega}_j, b_j)$ such that

$$\|f - f_m\|_{L^2}^2 \leq 3\frac{\|f\|_{\mathcal{B}}^2}{m}.$$

Moreover,

$$\|\boldsymbol{\theta}\| := \frac{1}{m}\sum_{j=1}^{m} |a_j| \left(\|\boldsymbol{\omega}_j\|_1 + |b_j|\right) \leq 2\|f\|_{\mathcal{B}}.$$

Application to high-dimensional PDEs

Open problem 2

Theorem (Direct approximation)

For any $f \in \mathcal{B}$ and $m \in \mathbb{N}$, there exists a two-layer neural network $f_m(\boldsymbol{x};\boldsymbol{\theta}) = \frac{1}{m}\sum_{j=1}^m a_j \sigma\left(\boldsymbol{\omega}_j^T \boldsymbol{x} + b_j\right)$, with m neurons $\boldsymbol{\theta} = (a_j, \boldsymbol{\omega}_j, b_j)$ such that

$$\|f - f_m\|_{L^2}^2 \leq 3\frac{\|f\|_{\mathcal{B}}^2}{m}.$$

Moreover,

$$\|\boldsymbol{\theta}\| := \frac{1}{m}\sum_{j=1}^m |a_j|\left(\|\boldsymbol{\omega}_j\|_1 + |b_j|\right) \leq 2\|f\|_{\mathcal{B}}.$$

Application to high-dimensional PDEs

Proposition

Let $\sigma(z) = \max\{z, 0\}$ and $g(x) = \sigma(x_1)$ be a Barron function on $\mathbb{R}^d$, $d \geq 2$. Denote by B^d the unit ball in $\mathbb{R}^d$ and by u the solution to

$$\begin{cases} -\Delta u = 0 & \text{in} \quad B^d \\ u = g & \text{on} \quad \partial B^d. \end{cases}$$

If $d \geq 3$, then u is not a Barron function on B^d.

Open problem 2

Open problem 2: regularity theory for PDEs in high dimensions

Open problem 2

Open problem 2: **regularity theory for PDEs in high dimensions**

Weinan E, Chao Ma, Lei Wu : *The Barron Space and the Flow-Induced Function Spaces for Neural Network Models* **Constructive Approximation,** Vol. 55, 369–406, 2022.

Weinan E. and S. Wojtowytsch: *Some observations on high-dimensional PDEs with Barron data.* **Proceedings of Machine Learning Research,** Vol. 107, 253–269, 2021.

Open problem 2

Open problem 2: regularity theory for PDEs in high dimensions

Weinan E, Chao Ma, Lei Wu : *The Barron Space and the Flow-Induced Function Spaces for Neural Network Models* **Constructive Approximation,** Vol. 55, 369–406, 2022.

Weinan E. and S. Wojtowytsch: *Some observations on high-dimensional PDEs with Barron data.* **Proceedings of Machine Learning Research,** Vol. 107, 253–269, 2021.

Required skills for applicants: Good command of Functional Analysis and PDEs

Open problem 2

Open problem 2: regularity theory for PDEs in high dimensions

- Weinan E, Chao Ma, Lei Wu : *The Barron Space and the Flow-Induced Function Spaces for Neural Network Models* **Constructive Approximation,** Vol. 55, 369–406, 2022.
- Weinan E. and S. Wojtowytsch: *Some observations on high-dimensional PDEs with Barron data.* **Proceedings of Machine Learning Research,** Vol. 107, 253–269, 2021.

Required skills for applicants: **Good command of Functional Analysis and PDEs**

Contact person: **Francisco Periago.** Email: f.periago@upct.es

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 !