Funtional and Numerical Analysis, Control of PDEs and Deep Learning

Francisco Periago

Universidad Politécnica de Cartagena
http://www.upct.es/mc3/en/dr-francisco-periago-esparza/

First meeting of the network COPI2A

Sevilla, january, 16-18, 2024

Goals and outline of the presentation

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

Goals and outline of the presentation

- **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: Functional Analysis and Machine Learning.** Are there solid Funcional and Numerical frameworks behind Machine Learning? What's known and what isn't known?

Goals and outline of the presentation

- **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: Functional Analysis and Machine Learning.** Are there solid Funcional and Numerical frameworks behind Machine Learning? What's known and what isn't known?
- **Part III: Control of PDEs and Machine Learning.** A toy control problem solved by using Deep-Learning to begin with...

Goals and outline of the presentation

- **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: Functional Analysis and Machine Learning.** Are there solid Funcional and Numerical frameworks behind Machine Learning? What's known and what isn't known?
- **Part III: Control of PDEs and Machine Learning.** A toy control problem solved by using Deep-Learning to begin with...
- Propose a list of open problems related to this topic.

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

Two cases:

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

Standard procedure for supervised learning(regression)

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 $\mathcal{H}_m$.

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

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

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**

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

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.

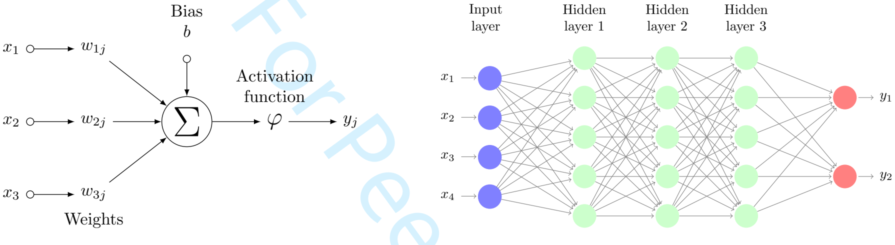
The overall objective is to minimize the **generalization error**

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

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)**.

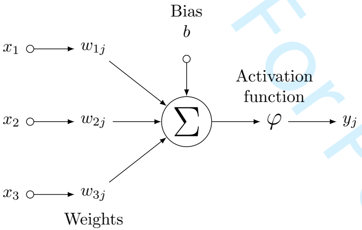


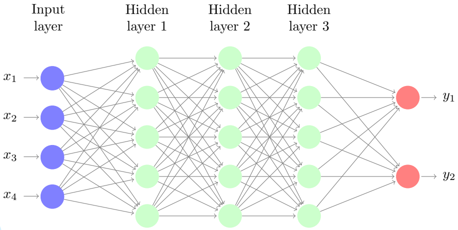
To each input $\boldsymbol{x} \in \mathbb{R}^d$ it associates the output $\boldsymbol{y} = f_m(\boldsymbol{x}) := \boldsymbol{x}^m$ defined by

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

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)**.





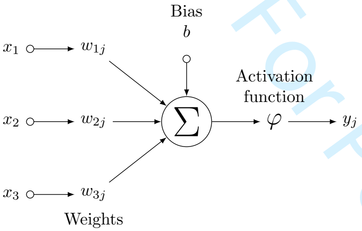
To each input $\boldsymbol{x} \in \mathbb{R}^d$ it associates the output $\boldsymbol{y} = f_m(\boldsymbol{x}) := \boldsymbol{x}^m$ defined by

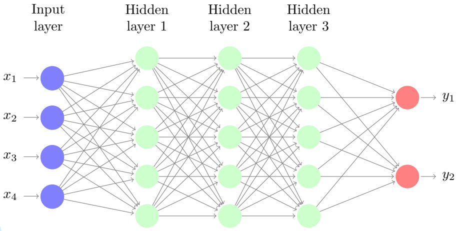
$$\begin{cases} \boldsymbol{x}^{k+1} = \sigma\left(\omega^k \boldsymbol{x}^k + b^k\right) \quad \text{for } k = 0, 1, \cdots, m-1 \\ \boldsymbol{x}^0 = \boldsymbol{x}, \end{cases} \tag{3}$$

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

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

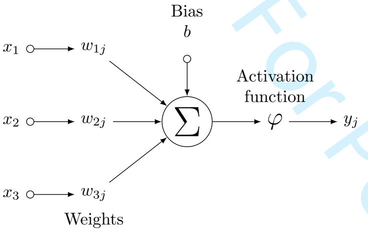
$$\begin{cases} \boldsymbol{x}^{k+1} = \sigma\left(\omega^k \boldsymbol{x}^k + b^k\right) & \text{for } k = 0, 1, \cdots, m-1 \\ \boldsymbol{x}^0 = \boldsymbol{x}, \end{cases} \tag{3}$$

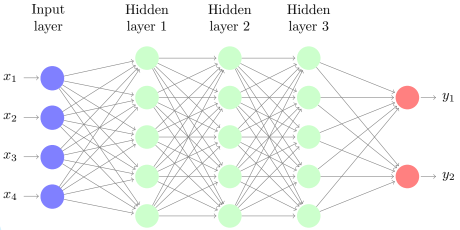
or in compositional form $\boldsymbol{x}^m = \left(\sigma \circ \Lambda^{m-1} \circ \cdots \circ \sigma \circ \Lambda^0\right)(\boldsymbol{x})$, $\Lambda^k \boldsymbol{x} = \omega^k \boldsymbol{x} + b^k$,

- 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

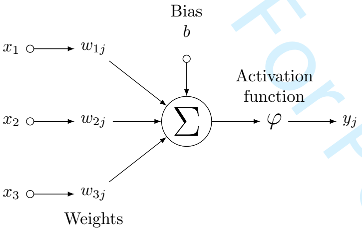
$$\begin{cases} \boldsymbol{x}^{k+1} = \sigma\left(\omega^k \boldsymbol{x}^k + b^k\right) & \text{for } k = 0, 1, \cdots, m-1 \\ \boldsymbol{x}^0 = \boldsymbol{x}, & \end{cases} \tag{3}$$

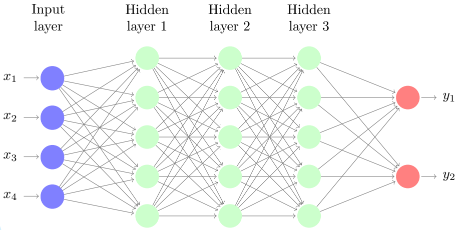
or in compositional form $\boldsymbol{x}^m = \left(\sigma \circ \Lambda^{m-1} \circ \cdots \circ \sigma \circ \Lambda^0\right)(\boldsymbol{x})$, $\Lambda^k \boldsymbol{x} = \omega^k \boldsymbol{x} + b^k$,

- 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

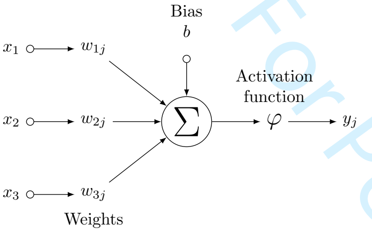
$$\begin{cases} \boldsymbol{x}^{k+1} = \sigma\left(\omega^k \boldsymbol{x}^k + b^k\right) & \text{for } k = 0, 1, \cdots, m-1 \\ \boldsymbol{x}^0 = \boldsymbol{x}, \end{cases} \tag{3}$$

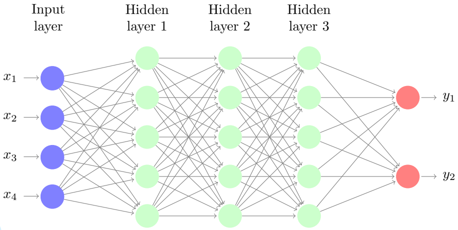
or in compositional form $\boldsymbol{x}^m = \left(\sigma \circ \Lambda^{m-1} \circ \cdots \circ \sigma \circ \Lambda^0\right)(\boldsymbol{x})$, $\Lambda^k \boldsymbol{x} = \omega^k \boldsymbol{x} + b^k$,

- 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** (denoted by φ in the figure)

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

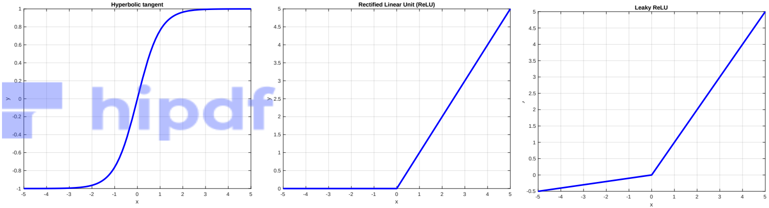
$$\sigma(\boldsymbol{x})_j := \sigma(\boldsymbol{x}_j), \quad 1 \leq j \leq d.$$

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

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

Common choices include *sigmoids* such as $\sigma(x) = \tanh(x)$, *rectifiers* such as ReLU: $\sigma(x) = \max\{x, 0\}$ or smooth ReLU: $\sigma(x) = \max\{x^3, 0\}$ and Leaky ReLU: $\sigma(x) = \max\{x, 0.1x\}$.



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

Part II

Functional Analysis and ML

Functional and numerical analysis

- Function to be approximated (learned): f^*

Functional and numerical analysis

- Function to be approximated (learned): f^*
- Hypothesis space: $\mathcal{H}_m$

Functional and numerical analysis

- Function to be approximated (learned): f^*
- Hypothesis space: $\mathcal{H}_m$
- Training error: $\hat{\mathcal{R}}_n(f) = \frac{1}{n}\sum_{i=1}^{n}\left(f(\boldsymbol{\theta}, \boldsymbol{x}_i) - f^*(\boldsymbol{x}_i)\right)^2, \quad f \in \mathcal{H}_m$

Functional and numerical analysis

- Function to be approximated (learned): f^*
- Hypothesis space: $\mathcal{H}_m$
- Training error: $\hat{\mathcal{R}}_n(f) = \frac{1}{n}\sum_{i=1}^{n}\left(f(\boldsymbol{\theta}, \boldsymbol{x}_i) - f^*(\boldsymbol{x}_i)\right)^2, \quad f \in \mathcal{H}_m$
- Output of the ML model: $\hat{f}(\boldsymbol{\theta}^\star) = \arg\min_{f \in \mathcal{H}_m} \hat{\mathcal{R}}_n(f)$

Functional and numerical analysis

- Function to be approximated (learned): f^*
- Hypothesis space: $\mathcal{H}_m$
- Training error: $\hat{\mathcal{R}}_n(f) = \frac{1}{n}\sum_{i=1}^{n}\left(f(\boldsymbol{\theta}, \boldsymbol{x}_i) - f^*(\boldsymbol{x}_i)\right)^2, \quad f \in \mathcal{H}_m$
- Output of the ML model: $\hat{f}(\boldsymbol{\theta}^\star) = \arg\min_{f \in \mathcal{H}_m} \hat{\mathcal{R}}_n(f)$
- Generalization error: $\mathcal{R}(f) = \mathbb{E}_{\boldsymbol{x} \sim \mathbb{P}}\left(f(\boldsymbol{x}_i) - f^*(\boldsymbol{x}_i)\right)^2, \quad f \in \mathcal{H}_m$

Functional and numerical analysis

- Function to be approximated (learned): f^*
- Hypothesis space: $\mathcal{H}_m$
- Training error: $\hat{\mathcal{R}}_n(f) = \frac{1}{n}\sum_{i=1}^{n}\left(f(\boldsymbol{\theta}, \boldsymbol{x}_i) - f^*(\boldsymbol{x}_i)\right)^2, \quad f \in \mathcal{H}_m$
- Output of the ML model: $\hat{f}(\boldsymbol{\theta}^\star) = \arg\min_{f\in\mathcal{H}_m} \hat{\mathcal{R}}_n(f)$
- Generalization error: $\mathcal{R}(f) = \mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}}\left(f(\boldsymbol{x}_i) - f^*(\boldsymbol{x}_i)\right)^2, \quad f \in \mathcal{H}_m$
- Best approximation in $\mathcal{H}_m$: $f_m = \arg\min_{f\in\mathcal{H}_m} \mathcal{R}(f)$

Functional and numerical analysis

- Function to be approximated (learned): f^*
- Hypothesis space: $\mathcal{H}_m$
- Training error: $\hat{\mathcal{R}}_n(f) = \frac{1}{n}\sum_{i=1}^{n}\left(f(\boldsymbol{\theta}, \boldsymbol{x}_i) - f^*(\boldsymbol{x}_i)\right)^2, \quad f \in \mathcal{H}_m$
- Output of the ML model: $\hat{f}(\boldsymbol{\theta}^\star) = \arg\min_{f\in\mathcal{H}_m} \hat{\mathcal{R}}_n(f)$
- Generalization error: $\mathcal{R}(f) = \mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}}\left(f(\boldsymbol{x}_i) - f^*(\boldsymbol{x}_i)\right)^2, \quad f \in \mathcal{H}_m$
- Best approximation in $\mathcal{H}_m$: $f_m = \arg\min_{f\in\mathcal{H}_m} \mathcal{R}(f)$
- Error $\equiv f^* - \hat{f}$

Functional and numerical analysis

- Function to be approximated (learned): f^*
- Hypothesis space: $\mathcal{H}_m$
- Training error: $\hat{\mathcal{R}}_n(f) = \frac{1}{n}\sum_{i=1}^{n}\left(f(\boldsymbol{\theta}, \boldsymbol{x}_i) - f^*(\boldsymbol{x}_i)\right)^2, \quad f \in \mathcal{H}_m$
- Output of the ML model: $\hat{f}(\boldsymbol{\theta}^\star) = \arg\min_{f\in\mathcal{H}_m} \hat{\mathcal{R}}_n(f)$
- Generalization error: $\mathcal{R}(f) = \mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}}\left(f(\boldsymbol{x}_i) - f^*(\boldsymbol{x}_i)\right)^2, \quad f \in \mathcal{H}_m$
- Best approximation in $\mathcal{H}_m$: $f_m = \arg\min_{f\in\mathcal{H}_m} \mathcal{R}(f)$
- Error $\equiv f^* - \hat{f} = \underbrace{f^* - f_m}_{\text{approximation error}} + \underbrace{f_m - \hat{f}}_{\text{estimation error}}$

Functional and numerical analysis

- Function to be approximated (learned): f^*
- Hypothesis space: $\mathcal{H}_m$
- Training error: $\hat{\mathcal{R}}_n(f) = \frac{1}{n}\sum_{i=1}^{n}\left(f(\boldsymbol{\theta}, \boldsymbol{x}_i) - f^*(\boldsymbol{x}_i)\right)^2, \quad f \in \mathcal{H}_m$
- Output of the ML model: $\hat{f}(\boldsymbol{\theta}^\star) = \arg\min_{f\in\mathcal{H}_m} \hat{\mathcal{R}}_n(f)$
- Generalization error: $\mathcal{R}(f) = \mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}}\left(f(\boldsymbol{x}_i) - f^*(\boldsymbol{x}_i)\right)^2, \quad f \in \mathcal{H}_m$
- Best approximation in $\mathcal{H}_m$: $f_m = \arg\min_{f\in\mathcal{H}_m} \mathcal{R}(f)$
- Error $\equiv f^* - \hat{f} = \underbrace{f^* - f_m}_{\text{approximation error}} + \underbrace{f_m - \hat{f}}_{\text{estimation error}}$

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 (CoD)**.

Functional and numerical analysis

- Function to be approximated (learned): f^*
- Hypothesis space: $\mathcal{H}_m$
- Training error: $\hat{\mathcal{R}}_n(f) = \frac{1}{n}\sum_{i=1}^{n}\left(f(\boldsymbol{\theta}, \boldsymbol{x}_i) - f^*(\boldsymbol{x}_i)\right)^2, \quad f \in \mathcal{H}_m$
- Output of the ML model: $\hat{f}(\boldsymbol{\theta}^\star) = \arg\min_{f \in \mathcal{H}_m} \hat{\mathcal{R}}_n(f)$
- Generalization error: $\mathcal{R}(f) = \mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}}\left(f(\boldsymbol{x}_i) - f^*(\boldsymbol{x}_i)\right)^2, \quad f \in \mathcal{H}_m$
- Best approximation in $\mathcal{H}_m$: $f_m = \arg\min_{f \in \mathcal{H}_m} \mathcal{R}(f)$
- Error $\equiv f^* - \hat{f} = \underbrace{f^* - f_m}_{\text{approximation error}} + \underbrace{f_m - \hat{f}}_{\text{estimation error}}$

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 (CoD)**. In ML we look for approximation errors that overcome (or at least mitigate) CoD.

Functional and numerical analysis

- Function to be approximated (learned): f^*
- Hypothesis space: $\mathcal{H}_m$
- Training error: $\hat{\mathcal{R}}_n(f) = \frac{1}{n}\sum_{i=1}^{n}(f(\boldsymbol{\theta}, \boldsymbol{x}_i) - f^*(\boldsymbol{x}_i))^2, \quad f \in \mathcal{H}_m$
- Output of the ML model: $\hat{f}(\boldsymbol{\theta}^\star) = \arg\min_{f\in\mathcal{H}_m} \hat{\mathcal{R}}_n(f)$
- Generalization error: $\mathcal{R}(f) = \mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}}(f(\boldsymbol{x}_i) - f^*(\boldsymbol{x}_i))^2, \quad f \in \mathcal{H}_m$
- Best approximation in $\mathcal{H}_m$: $f_m = \arg\min_{f\in\mathcal{H}_m} \mathcal{R}(f)$
- Error $\equiv f^* - \hat{f} = \underbrace{f^* - f_m}_{\text{approximation error}} + \underbrace{f_m - \hat{f}}_{\text{estimation error}}$

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 (CoD)**. In ML we look for approximation errors that overcome (or at least mitigate) CoD. A result that stands out CoD is the following one proven by Barron

$$\inf_{f_m\in\mathcal{H}_m} \|f^* - f_m\|_{L^2}^2 \lesssim \frac{\|f^*\|_*^2}{m}, \quad \|\cdot\|_* \text{ a suitable norm.}$$

Functional and numerical analysis

Estimation error (due to the fact that we have a finite dataset): typically Monte Carlo type estimates

$$I(g) = \int_X g(\boldsymbol{x})\, d\boldsymbol{x} = \underbrace{\frac{1}{n}\sum_{i=1}^{n} g(\boldsymbol{x}_i)}_{I_n(g)} + O(1/\sqrt{n})$$

Functional and numerical analysis

Estimation error (due to the fact that we have a finite dataset): typically Monte Carlo type estimates

$$I(g) = \int_X g(\boldsymbol{x})\, d\boldsymbol{x} = \underbrace{\frac{1}{n}\sum_{i=1}^{n} g(\boldsymbol{x}_i)}_{I_n(g)} + O(1/\sqrt{n})$$

We would like to accomplish the following:
Given an hypothesis space $\mathcal{H}_m$, identify a natural function space and a norm $\|\cdot\|_*$ that satisfies:

Functional and numerical analysis

Estimation error (due to the fact that we have a finite dataset): typically Monte Carlo type estimates

$$I(g) = \int_X g(\boldsymbol{x})\, d\boldsymbol{x} = \underbrace{\frac{1}{n}\sum_{i=1}^{n} g(\boldsymbol{x}_i)}_{I_n(g)} + O(1/\sqrt{n})$$

We would like to accomplish the following:

Given an hypothesis space $\mathcal{H}_m$, identify a natural function space and a norm $\|\cdot\|_*$ that satisfies:

$$\text{error de generalización} \lesssim \frac{\|f^*\|_*^2}{m} + \frac{\|f^*\|_*}{\sqrt{n}}.$$

Two-layer neural networks and Barron space

Two-layer neural networks and Barron space

A two-layer neural network may be represented as

$$f_m(\boldsymbol{x}) = \frac{1}{m}\sum_{j=1}^{m} a_j \sigma\left(\boldsymbol{\omega}_j^T \boldsymbol{x} + b_j\right) \tag{4}$$

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

Two-layer neural networks and Barron space

A two-layer neural network may be represented as

$$f_m(\boldsymbol{x}) = \frac{1}{m} \sum_{j=1}^{m} a_j \sigma \left(\boldsymbol{\omega}_j^T \boldsymbol{x} + b_j \right) \tag{4}$$

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

Two-layer neural networks and Barron space

A two-layer neural network may be represented as

$$f_m(\boldsymbol{x}) = \frac{1}{m}\sum_{j=1}^{m} a_j \sigma\left(\boldsymbol{\omega}_j^T \boldsymbol{x} + b_j\right) \tag{4}$$

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

Where does this expression come from?

Starting from the Fourier transform-type representation

$$f(\boldsymbol{x}) = \int_{\mathbb{R}^d} a(\boldsymbol{\omega}) e^{i(\boldsymbol{\omega} x)} \rho\left(d\boldsymbol{\omega}\right),$$

with ρ a probability measure on $\mathbb{R}^d$, and by independently sample $\{\boldsymbol{\omega}_j\}_{j=1}^m$ we obtain the dimension-independent approximation

$$f(\boldsymbol{x}) \approx f_m(\boldsymbol{x}) = \frac{1}{m}\sum_{j=1}^{m} a(\boldsymbol{\omega}_j)\sigma\left(\boldsymbol{\omega}_j^T \boldsymbol{x}\right) = \frac{1}{m}\sum_{j=1}^{m} a_j \sigma\left(\boldsymbol{\omega}_j^T \boldsymbol{x}\right), \quad \sigma(z) = e^{iz},$$

which is of the same type as in (4).

Two-layer neural networks and Barron space

A two-layer neural network may be represented as

$$f_m(\boldsymbol{x}) = \frac{1}{m}\sum_{j=1}^{m} a_j \sigma\left(\boldsymbol{\omega}_j^T \boldsymbol{x} + b_j\right) \tag{4}$$

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

Where does this expression come from?

Starting from the Fourier transform-type representation

$$f(\boldsymbol{x}) = \int_{\mathbb{R}^d} a(\boldsymbol{\omega}) e^{i(\boldsymbol{\omega} x)} \rho\,(d\boldsymbol{\omega})\,,$$

with ρ a probability measure on $\mathbb{R}^d$, and by independently sample $\{\boldsymbol{\omega}_j\}_{j=1}^m$ we obtain the dimension-independent approximation

$$f(\boldsymbol{x}) \approx f_m(\boldsymbol{x}) = \frac{1}{m}\sum_{j=1}^{m} a(\boldsymbol{\omega}_j)\sigma\left(\boldsymbol{\omega}_j^T \boldsymbol{x}\right) = \frac{1}{m}\sum_{j=1}^{m} a_j \sigma\left(\boldsymbol{\omega}_j^T \boldsymbol{x}\right), \quad \sigma(z) = e^{iz},$$

which is of the same type as in (4). Passing to the limit when the with of the hidden layer goes to infinity in (4) we get the representation formula

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

Two-layer neural networks and Barron space

For the case of ReLU- activation function, the space for two-layer NN is that so-called *Barron space* $\mathcal{B}$, which is composed of functions $f : D \subset \mathbb{R}^d \to \mathbb{R}$ for which the following norm is finite

$$\|f\|_{\mathcal{B}} := \inf \left\{ \int_{\mathbb{R}^{d+2}} |a|[|\boldsymbol{\omega}| + |b|]\, \rho\,(da, d\boldsymbol{\omega}, db) \;:\; \rho \text{ s.t. } f = f_\rho \right\}.$$

Two-layer neural networks and Barron space

For the case of ReLU- activation function, the space for two-layer NN is that so-called *Barron space* $\mathcal{B}$, which is composed of functions $f : D \subset \mathbb{R}^d \to \mathbb{R}$ for which the following norm is finite

$$\|f\|_{\mathcal{B}} := \inf \left\{ \int_{\mathbb{R}^{d+2}} |a|[|\boldsymbol{\omega}| + |b|] \, \rho\,(da, d\boldsymbol{\omega}, db) \, : \, \rho \text{ s.t. } f = f_\rho \right\}.$$

Basic properties of Barron space

Two-layer neural networks and Barron space

For the case of ReLU- activation function, the space for two-layer NN is that so-called *Barron space* $\mathcal{B}$, which is composed of functions $f : D \subset \mathbb{R}^d \to \mathbb{R}$ for which the following norm is finite

$$\|f\|_{\mathcal{B}} := \inf \left\{ \int_{\mathbb{R}^{d+2}} |a|[|\boldsymbol{\omega}| + |b|]\, \rho\,(da, d\boldsymbol{\omega}, db) \;:\; \rho \text{ s.t. } f = f_\rho \right\}.$$

Basic properties of Barron space

- If $f \in H^s(\mathbb{R}^d)$ for $s > d/2 + 2$, then $f \in \mathcal{B}$.

Two-layer neural networks and Barron space

For the case of ReLU- activation function, the space for two-layer NN is that so-called *Barron space* $\mathcal{B}$, which is composed of functions $f : D \subset \mathbb{R}^d \to \mathbb{R}$ for which the following norm is finite

$$\|f\|_{\mathcal{B}} := \inf \left\{ \int_{\mathbb{R}^{d+2}} |a|[|\boldsymbol{\omega}| + |b|] \, \rho\left(da, d\boldsymbol{\omega}, db\right) \, : \, \rho \text{ s.t. } f = f_\rho \right\}.$$

Basic properties of Barron space

- If $f \in H^s(\mathbb{R}^d)$ for $s > d/2 + 2$, then $f \in \mathcal{B}$.
- Barron space embeds into the space of Lipschitz-continuous functions.

Two-layer neural networks and Barron space

For the case of ReLU- activation function, the space for two-layer NN is that so-called *Barron space* $\mathcal{B}$, which is composed of functions $f : D \subset \mathbb{R}^d \to \mathbb{R}$ for which the following norm is finite

$$\|f\|_{\mathcal{B}} := \inf \left\{ \int_{\mathbb{R}^{d+2}} |a|[|\boldsymbol{\omega}| + |b|]\, \rho\,(da, d\boldsymbol{\omega}, db) \;:\; \rho \text{ s.t. } f = f_\rho \right\}.$$

Basic properties of Barron space

- If $f \in H^s(\mathbb{R}^d)$ for $s > d/2 + 2$, then $f \in \mathcal{B}$.
- Barron space embeds into the space of Lipschitz-continuous functions.
- If $f \in \mathcal{B}$, then $f = \sum_{i=1}^{\infty} f_i$, where $f_i(\boldsymbol{x}) = g_i(P_i \boldsymbol{x} + b_i)$ and
  - g_i is C^1 except at the origin, b_i is a shift vector, and
  - P_i is an orthogonal projection on a k_i-dimensional subspace, $0 \leq k_i \leq d - 1$.

Two-layer neural networks and Barron space

For the case of ReLU- activation function, the space for two-layer NN is that so-called *Barron space* $\mathcal{B}$, which is composed of functions $f : D \subset \mathbb{R}^d \to \mathbb{R}$ for which the following norm is finite

$$\|f\|_{\mathcal{B}} := \inf \left\{ \int_{\mathbb{R}^{d+2}} |a|[|\boldsymbol{\omega}| + |b|] \, \rho \left(da, d\boldsymbol{\omega}, db \right) \, : \, \rho \text{ s.t. } f = f_\rho \right\}.$$

Basic properties of Barron space

- If $f \in H^s(\mathbb{R}^d)$ for $s > d/2 + 2$, then $f \in \mathcal{B}$.
- Barron space embeds into the space of Lipschitz-continuous functions.
- If $f \in \mathcal{B}$, then $f = \sum_{i=1}^{\infty} f_i$, where $f_i(\boldsymbol{x}) = g_i(P_i \boldsymbol{x} + b_i)$ and
  - g_i is C^1 except at the origin, b_i is a shift vector, and
  - P_i is an orthogonal projection on a k_i-dimensional subspace, $0 \leq k_i \leq d - 1$.
- **Approximation error.** For any $f \in \mathcal{B}$ and $m \in \mathbb{N}$, there exists a two-layer neural network f_m, with m neurons $(a_j, \boldsymbol{\omega}_j, b_j)$ such that

$$\|f - f_m\|_{L^2} \lesssim \frac{\|f^*\|_*^2}{m},$$

Two-layer neural networks and Barron space

For the case of ReLU- activation function, the space for two-layer NN is that so-called *Barron space* $\mathcal{B}$, which is composed of functions $f : D \subset \mathbb{R}^d \to \mathbb{R}$ for which the following norm is finite

$$\|f\|_{\mathcal{B}} := \inf \left\{ \int_{\mathbb{R}^{d+2}} |a|[|\boldsymbol{\omega}| + |b|] \, \rho\,(da, d\boldsymbol{\omega}, db) \; : \; \rho \text{ s.t. } \; f = f_\rho \right\}.$$

Basic properties of Barron space

- If $f \in H^s(\mathbb{R}^d)$ for $s > d/2 + 2$, then $f \in \mathcal{B}$.
- Barron space embeds into the space of Lipschitz-continuous functions.
- If $f \in \mathcal{B}$, then $f = \sum_{i=1}^{\infty} f_i$, where $f_i(\boldsymbol{x}) = g_i(P_i\boldsymbol{x} + b_i)$ and
    - g_i is C^1 except at the origin, b_i is a shift vector, and
    - P_i is an orthogonal projection on a k_i-dimensional subspace, $0 \leq k_i \leq d - 1$.
- **Approximation error.** For any $f \in \mathcal{B}$ and $m \in \mathbb{N}$, there exists a two-layer neural network f_m, with m neurons $(a_j, \boldsymbol{\omega}_j, b_j)$ such that

$$\|f - f_m\|_{L^2} \lesssim \frac{\|f^*\|_*^2}{m},$$

- **Estimation error** in Barron spaces is controlled by a Monte Carlo type ratio.

Function spaces for neural networks architectures

- **Residual networks** $\Longrightarrow$ *flow-induced spaces*
- **Multilayer networks** $\Longrightarrow$ *tree-like spaces*

Function spaces for neural networks architectures

- **Residual networks** $\Longrightarrow$ *flow-induced spaces*
- **Multilayer networks** $\Longrightarrow$ *tree-like spaces*
- **Convolutional networks** $\Longrightarrow$ *???*

Function spaces for neural networks architectures

- **Residual networks** $\Longrightarrow$ *flow-induced spaces*
- **Multilayer networks** $\Longrightarrow$ *tree-like spaces*
- **Convolutional networks** $\Longrightarrow$ *???*
- **DenseNets** $\Longrightarrow$ *???*

Function spaces for neural networks architectures

- **Residual networks** $\Longrightarrow$ *flow-induced spaces*
- **Multilayer networks** $\Longrightarrow$ *tree-like spaces*
- **Convolutional networks** $\Longrightarrow$ *???*
- **DenseNets** $\Longrightarrow$ *???*

Weinan E. et al.: Towards a mathematical understanding of Neural Network-based Machine Learning: what we know and we don't known Preprint (2020). Available at https://web.math.princeton.edu/~weinan/

Weinan E, Chao Ma and Lei Wu, "Machine Learning from a Continuous Viewpoint" , 2019. Available at https://web.math.princeton.edu/~weinan/

Function spaces for neural networks architectures

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.

Weinan E. and S. Wojtowytsch: Some observations on high-dimensional PDEs with Barron data. (2021) Available at
https://web.math.princeton.edu/~weinan/

Function spaces for neural networks architectures

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.

Weinan E. and S. Wojtowytsch: Some observations on high-dimensional PDEs with Barron data. (2021) Available at https://web.math.princeton.edu/~weinan/

Open problem: **regularity theory for PDEs in high dimension**

Part III

Control of PDEs and ML

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

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

Raisi, M., Perdikaris, P., Karniadakis, G.E.: *Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations.* **J. Comput. Phys.** 378, 686-707 (2019)

Numerical approximation using ML

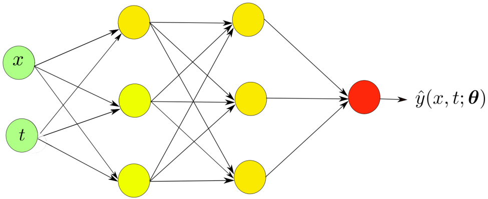
A Physics-informed neural networks (PINNs) algorithm

Numerical approximation using ML

A Physics-informed neural networks (PINNs) algorithm

Step 1: Neural network

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



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

- $\mathcal{N}^\ell : \mathbb{R}^{d_{in}} \to \mathbb{R}^{d_{out}}$ 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. $\sigma(s) = \tanh(s)$

A Physics-informed neural networks (PINNs) algorithm

Step 2: Training dataset

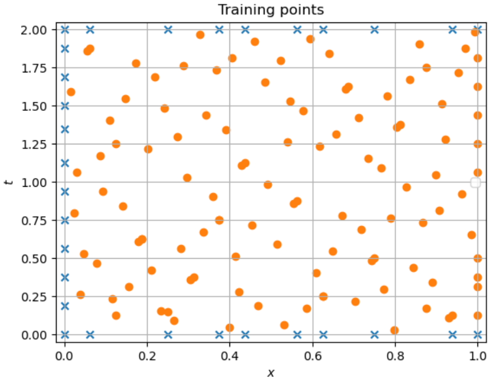


Figure: Illustration of a training dataset (based on Sobol points) in the domain $Q_2 = (0,1) \times (0,2)$. Interior points are marked with circles and boundary points in blue color. (x_j, t_j) are the features.

Numerical approximation using ML

A Physics-informed neural networks (PINNs) algorithm

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.

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

Numerical approximation using ML

A Physics-informed neural networks (PINNs) algorithm

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

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

Numerical approximation using ML

A Physics-informed neural networks (PINNs) algorithm

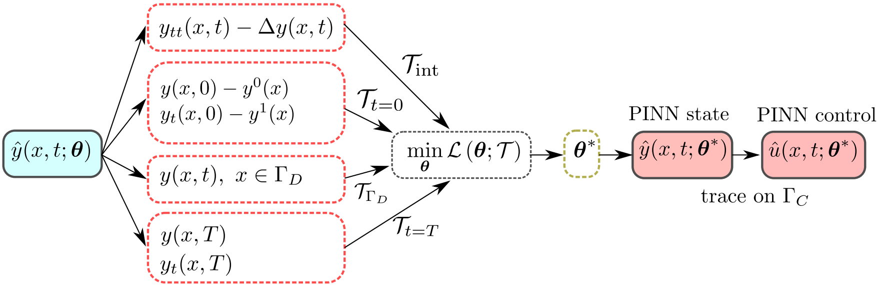
Step 4: **Training process**

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

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:



Numerical approximation using ML

Estimates on generalization error

Numerical approximation using ML

Estimates on generalization error

Training error

$$\mathcal{E}_{\text{train}} := \mathcal{E}_{\text{train, int}} + \mathcal{E}_{\text{train, boundary}} + \mathcal{E}_{\text{train, initialpos}} + \mathcal{E}_{\text{train, initialvel}} + \mathcal{E}_{\text{train, finalpos}} + \mathcal{E}_{\text{train, finalvel}},$$

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

Numerical approximation using ML

Estimates on generalization error

Training error

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

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

Generalization error **for control and state**

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

Numerical approximation using ML

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.**(2023) 196:391–414

Numerical approximation using ML

Idea of the proof. Let $\overline{y} = y - \hat{y}$ and $\overline{u} = u - \hat{u}$. By linearity,

$$\begin{cases} \overline{y}_{tt} - \Delta \overline{y} = \hat{y}_{tt} - \Delta \hat{y}, & \text{in } Q_T \\ \overline{y}(x,0) = y^0(x) - \hat{y}(x,0), & \text{in } \Omega \\ \overline{y}_t(x,0) = y^1(x) - \hat{y}_t(x,0) & \text{in } \Omega \\ \overline{y}(x,T) = \hat{y}(x,T), & \text{in } \Omega \\ \overline{y}_t(x,T) = \hat{y}_t(x,T) & \text{in } \Omega \\ \overline{y}(x,t) = \hat{y}(x,t), & \text{on } \Gamma_D \times (0,T) \\ \overline{y}(x,t) = u(x,t) - \hat{y}(x,t) & \text{on } \Gamma_C \times (0,T). \end{cases} \tag{5}$$

Numerical approximation using ML

Idea of the proof. Let $\overline{y} = y - \hat{y}$ and $\overline{u} = u - \hat{u}$. By linearity,

$$\begin{cases} \overline{y}_{tt} - \Delta \overline{y} = \hat{y}_{tt} - \Delta \hat{y}, & \text{in } Q_T \\ \overline{y}(x,0) = y^0(x) - \hat{y}(x,0), & \text{in } \Omega \\ \overline{y}_t(x,0) = y^1(x) - \hat{y}_t(x,0) & \text{in } \Omega \\ \overline{y}(x,T) = \hat{y}(x,T), & \text{in } \Omega \\ \overline{y}_t(x,T) = \hat{y}_t(x,T) & \text{in } \Omega \\ \overline{y}(x,t) = \hat{y}(x,t), & \text{on } \Gamma_D \times (0,T) \\ \overline{y}(x,t) = u(x,t) - \hat{y}(x,t) & \text{on } \Gamma_C \times (0,T). \end{cases} \tag{5}$$

Again by linearity, $\overline{y}(x,t;\boldsymbol{\theta})$ is decomposed as $\overline{y} = \overline{y}^1 + \overline{y}^2$, where

$$\begin{cases} \overline{y}^1_{tt} - \Delta \overline{y}^1 = 0, & \text{in } Q_T \\ \overline{y}^1(x,0) = y^0(x) - \hat{y}(x,0), & \text{in } \Omega \\ \overline{y}^1_t(x,0) = y^1(x) - \hat{y}_t(x,0) & \text{in } \Omega \\ \overline{y}^1(x,t) = 0, & \text{on } \Gamma_D \times (0,T) \\ \overline{y}^1(x,t) = u(x,t) - \hat{y}(x,t) & \text{on } \Gamma_C \times (0,T) \end{cases} \tag{6}$$

$$\begin{cases} \overline{y}^2_{tt} - \Delta \overline{y}^2 = \hat{y}_{tt} - \Delta \hat{y}, & \text{in } Q_T \\ \overline{y}^2(x,0) = 0, & \text{in } \Omega \\ \overline{y}^2_t(x,0) = 0 & \text{in } \Omega \\ \overline{y}^2(x,T) = \hat{y}(x,T) - \overline{y}^1(x,T), & \text{in } \Omega \\ \overline{y}^2_t(x,T) = \hat{y}_t(x,T) - \overline{y}^1_t(x,T), & \text{in } \Omega \\ \overline{y}^2(x,t) = \hat{y}(x,t), & \text{on } \Gamma_D \times (0,T) \\ \overline{y}^2(x,t) = 0 & \text{on } \Gamma_C \times (0,T). \end{cases} \tag{7}$$

Numerical approximation using ML

Idea of the proof (cont). By applying an observability inequality to system (6), and an energy estimate to (7),

$$
\begin{aligned}
&\|u-\hat{u}\|_{L^2(\Gamma_C;(0,T))}\\
&\leq C_o\left(\|y^0-\hat{y}(0)\|_{L^2(\Omega)}+\|y^1-\hat{y}_t(0)\|_{H^{-1}(\Omega)}+\|\overline{y}^1(T)\|_{L^2(\Omega)}+\|\overline{y}^1_t(T)\|_{H^{-1}(\Omega)}\right)\\
&\leq C_o\left(\|y^0-\hat{y}(0)\|_{L^2(\Omega)}+\|y^1-\hat{y}_t(0)\|_{L^2(\Omega)}+\|\hat{y}(T)\|_{L^2(\Omega)}+\|\hat{y}_t(T)\|_{L^2(\Omega)}\right.\\
&\left.+\|\overline{y}^2(T)\|_{L^2(\Omega)}+\|\overline{y}^2_t(T)\|_{H^{-1}(\Omega)}\right)\\
&\leq C_o\left(\|y^0-\hat{y}(0)\|_{L^2(\Omega)}+\|y^1-\hat{y}_t(0)\|_{L^2(\Omega)}+\|\hat{y}(T)\|_{L^2(\Omega)}+\|\hat{y}_t(T)\|_{L^2(\Omega)}\right.\\
&\left.+C_e\left(\|\hat{y}\|_{L^2(\Gamma_D\times(0,T))}+\|\hat{y}_{tt}-\Delta\hat{y}\|_{L^2(0,T;L^2(\Omega))}\right)\right). \qquad (8)
\end{aligned}
$$

The fact that training error converges to zero is a consequence of Pinkus' universal approximation theorem, which basically states that any function $f\in C^k$ may be approximate in the $\|\cdot\|_{C^k}$ by a suitable two-layer neural network. □

Numerical approximation using ML

Some comments and related open questions:

- The PINN algo generalises to any control system both linear and nonlinear

Numerical approximation using ML

Some comments and related open questions:

- The PINN algo generalises to any control system both linear and nonlinear
- How does the constant $C(\Omega, T)$ depends on the dimension d?

Numerical approximation using ML

Some comments and related open questions:

- The PINN algo generalises to any control system both linear and nonlinear
- How does the constant $C(\Omega, T)$ depends on the dimension d?
- The proof uses linearity. How can be get similar estimates of generalization error for semilinear PDEs?

Numerical approximation using ML

Some comments and related open questions:

- The PINN algo generalises to any control system both linear and nonlinear
- How does the constant $C(\Omega, T)$ depends on the dimension d?
- The proof uses linearity. How can be get similar estimates of generalization error for semilinear PDEs?
- Construct a unique prediction model for *all* initial data.
- .....

Numerical approximation using ML

Numerical experiments

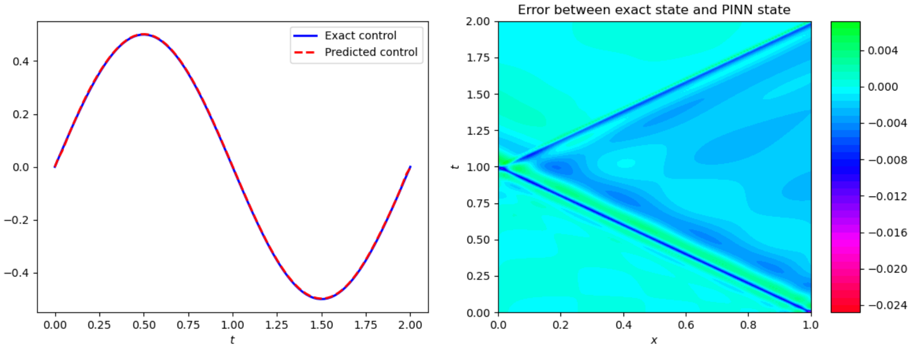


Figure: Experiment 1 (linear wave equation). $y^0(x) = \sin(\pi x)$, $y^1(x) = 0$. Neural network composed of 4 hidden layers and 50 neurons in each layer. Relative generalization error of the order of 2%.

Implementation with https://github.com/lululxvi/deepxde
Python scripts available at https://github.com/fperiago/deepcontrol

Work supported by

- Fundación Séneca (Agencia de Ciencia y Tecnología de la Región de Murcia (Spain)) under contract 20911/PI/18 and grant number 21503/EE/21 (mobility program Jiménez de la Espada), and through the programme for the development of scientific and technical research by competitive groups (21996/PI/22).

Región de Murcia

f SéNeCa(+)

Agencia de Ciencia y Tecnología
Región de Murcia

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

GOBIERNO DE ESPAÑA

MINISTERIO DE ECONOMÍA Y COMPETITIVIDAD

FECYT

Thank you for your attention !