SUMMER SCHOOL

An excursion into Machine Learning in Computational Engineering

July 4th – 8th, 2022

CIMNE

EXCELENCIA SEVERO OCHOA

Physics(PDE)-Informed Neural Networks

Mohammad R. Hashemi

CIMNE EXCELENCIA SEVERO OCHOA

Idea

- Using neural networks as “***universal function approximators***”
- Introduce physics via “***partial differential equations***”
- Automatic differentiation is the key!
- Compensating for the lack of data
  - Improving the convergence

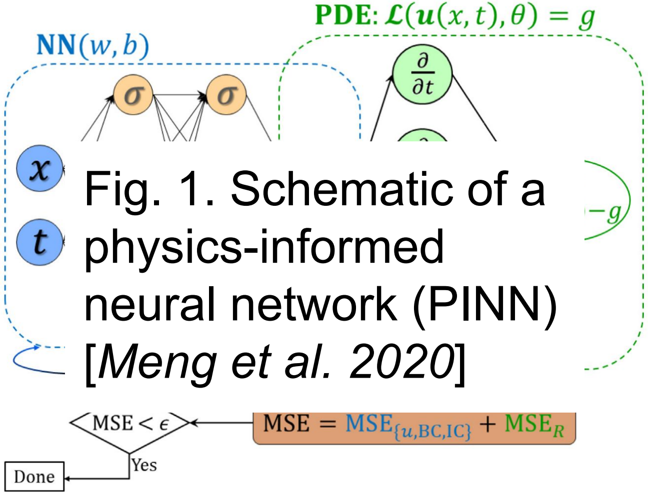


Fig. 1. Schematic of a physics-informed neural network (PINN) [*Meng et al. 2020*]

Meng, X., Li, Z., Zhang, D., & Karniadakis, G. E. (2020). PPINN: Parareal physics-informed neural network for time-dependent PDEs. *Computer Methods in Applied Mechanics and Engineering*, *370*, 113250.

Outline

1. Brief overview of the ***theory***
    a. Brief history
    b. Implementation
    c. Examples

Some references:

- https://www.sciencedirect.com/science/article/pii/S0021999118307125
- https://github.com/lululxvi/tutorials/blob/master/20211210_pinn/pinn.pdf
- https://www.nature.com/articles/s42254-021-00314-5
- https://www.science.org/doi/abs/10.1126/science.aaw4741

2. [Simple TensorFlow ***examples***]
    a. Automatic differentiation
    https://colab.research.google.com/drive/1xYvR6-KkWLrh0e5Zx3--jSbvvWxQJS20?usp=sharing
    b. Solving an ODE
    https://colab.research.google.com/drive/1FxUsHdenz9tGb1fkH8Ls7_O0hhTWR-dM?usp=sharing
    c. (Inverse) time-dependent diffusion
    https://colab.research.google.com/drive/1dnYw1k_mek8bbelIwTH3VCOfynajCfa7?usp=sharing

Brief History

- The first attempts were to infer “model parameters” using **a few “noisy” data**
  - Probabilistic machine learning (Gaussian process): maximum likelihood
  - Raissi, M., Perdikaris, P., & Karniadakis, G. E. (***2017***). Machine learning of linear differential equations using Gaussian processes. Journal of Computational Physics, 348, 683-693.

This work has so-far received 350 citations!

Main reference:
Owhadi, H. (**2015**). Bayesian numerical homogenization. Multiscale Modeling & Simulation, 13(3), 812-828. (147 citations)

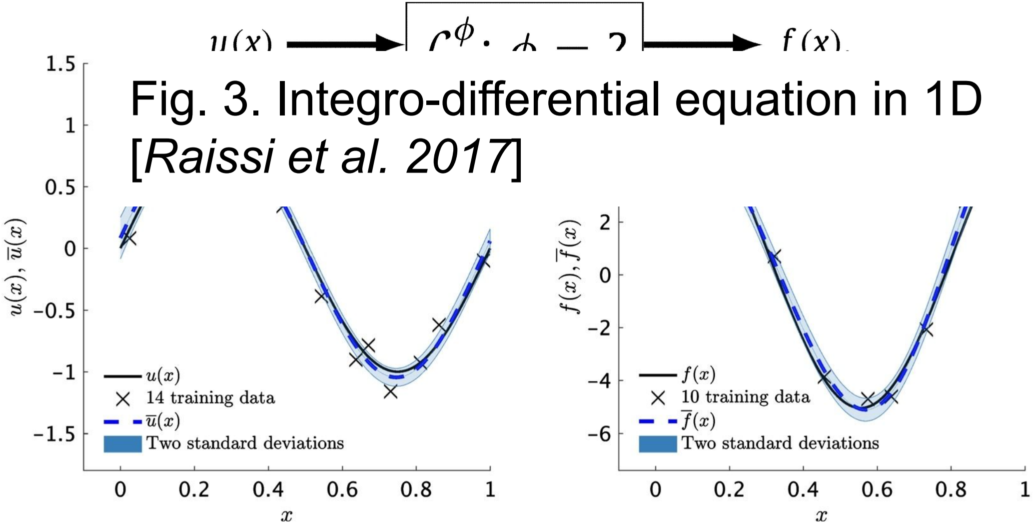


Fig. 3. Integro-differential equation in 1D [*Raissi et al. 2017*]

Brief History

- Shortcomings:
  - Nonlinear problems need prior linearization: discretization in time
  - Probabilistic nature of the Gaussian process

- Looking for a robust approach:
  - Deep neural networks are **universal function approximators**

    [Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. Neural networks, 2(5), 359-366.]

    “Standard multilayer feedforward networks are capable of ***approximating any measurable function*** *to any desired degree of accuracy. There are* ***no theoretical constraints*** *for the success of feedforward networks. Lack of success is due to inadequate learning, insufficient number of hidden units or the lack of a deterministic relationship between input and target.”*

http://www.cs.cmu.edu/afs/cs/user/bhiksha/WWW/courses/deeplearning/Fall.2016/notes/Sonia_Hornik.pdf

Brief History

- Automatic differentiation lets direct incorporation of PDEs
  - Straightforward treatment of nonlinearities
  - Continuous time models

- Constraints on NNs are derived from the governing physical laws:

  Physics-informed NNs

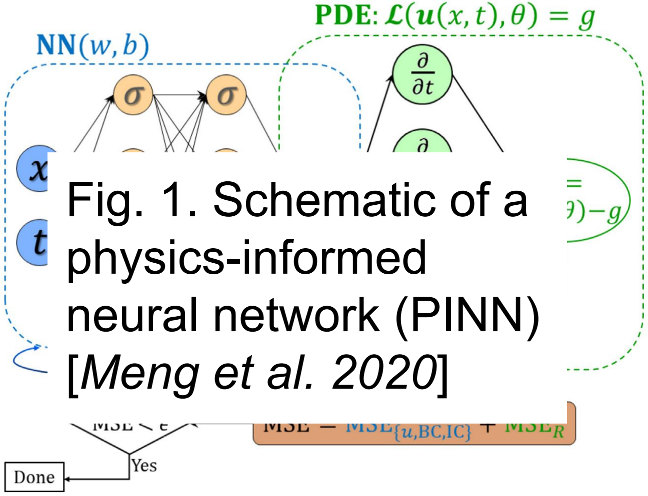


Fig. 1. Schematic of a physics-informed neural network (PINN) [*Meng et al. 2020*]

Meng, *et al.* (2020)

Foundational work:
Raissi, M., Perdikaris, P., & Karniadakis, G. E. (**2019**). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, *378*, 686-707. (3048 citations)

Brief History

- Previously overlooked publications:
  - **Dissanayake, M. W. M. G., & Phan-Thien, N. (1994)**. Neural-network-based approximations for solving partial differential equations. communications in Numerical Methods in Engineering, 10(3), 195-201 (213 citations).

$$\mathscr{L}u = f, \quad x \in \Omega$$
$$\mathscr{B}u = g, \quad x \in \partial\Omega$$

Neural network:

$$u = u_a(x, \beta_i)$$

Minimize to find the parameters:

$$h = \int_\Omega \| \mathscr{L}u_a - f \|^2 \, \mathrm{d}V + \int_{\partial\Omega} \| \mathscr{B}u_a - g \|^2 \, \mathrm{d}S$$

  - **Lagaris, I. E., Likas, A., & Fotiadis, D. I. (1998)**. Artificial neural networks for solving ordinary and partial differential equations. IEEE transactions on neural networks, 9(5), 987-1000 (1337 citations).

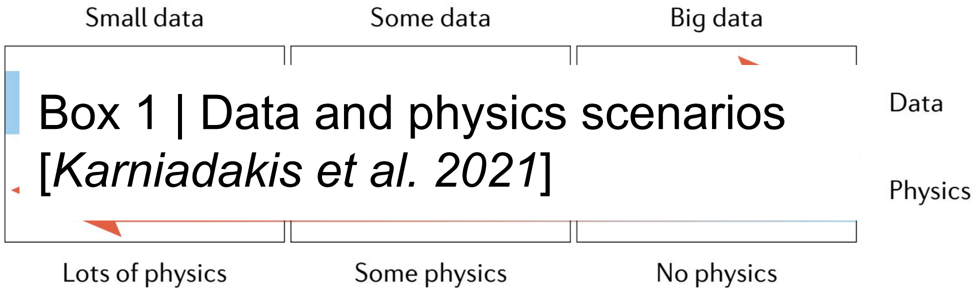
$$G(\vec{x}, \Psi(\vec{x}), \nabla\Psi(\vec{x}), \nabla^2\Psi(\vec{x})) = 0, \qquad \vec{x} \in D$$

$$\min_{\vec{p}} \sum_{\vec{x}_i \in \hat{D}} (G(\vec{x}_i, \Psi_t(\vec{x}_i, \vec{p}), \nabla\Psi_t(\vec{x}_i, \vec{p}), \nabla^2\Psi_t(\vec{x}_i, \vec{p})))^2$$

Implementation

- Assume that we have
  - Neural network: $u(x,t) = \mathcal{NN}(x,t;\boldsymbol{\theta})$ physics: $\mathcal{L}[u(x,t);\boldsymbol{\phi}] = 0$

- What is the main aim?
  - Parameters are fixed, find unknown “u”.
  - Some observations are available, identify the system (find parameters): inverse problem



Box 1 | Data and physics scenarios [*Karniadakis et al. 2021*]

Karniadakis, G. E., Kevrekidis, I. G., Lu, L., Perdikaris, P., Wang, S., & Yang, L. (2021). Physics-informed machine learning. *Nature Reviews Physics*, *3*(6), 422-440.

Implementation

- Assume that we have
  - Neural network: $u(x,t) = \mathcal{NN}(x,t;\boldsymbol{\theta})$ physics: $\mathcal{L}[u(x,t);\boldsymbol{\phi}] = 0$

    $\boldsymbol{\theta} = \{\mathbf{W}^j, \mathbf{b}^j\}_{1\le j\le l}$
  - Expanded: $u(x,t;\boldsymbol{\theta}) = \mathbf{W}^l\sigma(\mathbf{W}^{l-1}\sigma(\mathbf{W}^{l-1}\ldots\sigma(\mathbf{W}^1[x\ t]^T + \mathbf{b}^1)\ldots) + \mathbf{b}^{l-1}) + b^l$
  - Standard for NNs: $\dfrac{\partial \mathrm{MSE}(\mathbf{X};\boldsymbol{\theta})}{\partial \theta_i}$ needing: $\dfrac{\partial u(x,t;\boldsymbol{\theta})}{\partial x}, \dfrac{\partial u(x,t;\boldsymbol{\theta})}{\partial t}, \; etc.$
- Differentiation is done similar to the standard back-propagation algorithm
  - *Example …*
  - For a detailed overview of the implementation, see [Lu, L., Meng, X., Mao, Z., & Karniadakis, G. E. (2021). DeepXDE: A deep learning library for solving differential equations. SIAM Review, 63(1), 208-228.]

Implementation

- Assume that we have
  - Neural network: $u(x,t) = \mathcal{NN}(x,t;\boldsymbol{\theta})$ physics: $\mathcal{L}[u(x,t);\boldsymbol{\phi}] = 0$

- Defining the residual of PDE, $\mathcal{R}_i$, at collocation points $\{x_i, t_i\}$
  - Total mean square error subject to the minimization

$$\mathrm{MSE} = w_{data}\mathrm{MSE}_{data} + w_{PDE}\mathrm{MSE}_{PDE}$$

$$\mathrm{MSE}_{PDE} = \frac{1}{N}\sum_{i=1}^{N}(\mathcal{R}_i)^2$$

$$\mathrm{MSE}_{data} = \frac{1}{N_d}\sum_{i=1}^{N_d}(u(x_i,t_i;\boldsymbol{\theta}) - \bar{u}_i)^2$$

Data is composed of the **initial** and **boundary** conditions as well as the observations

Examples

- Burger's equation

$$u_t + uu_x - (0.01/\pi)u_{xx} = 0, \quad x \in [-1, 1], \quad t \in [0, 1],$$

$$u(0, x) = -\sin(\pi x),$$

$$u(t, -1) = u(t, 1) = 0.$$

  - Continuous time domain: spatio-temporal training data
  - 10,000 collocation points
  - 100 data points

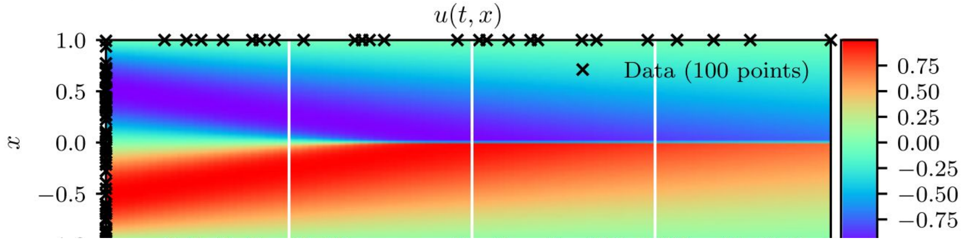
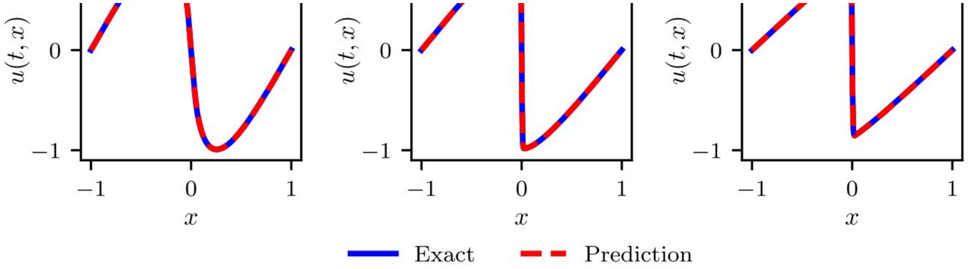


Fig. A.6. Burgers' equation [*Raissi et al. 2019*]



Raissi *et al.* (**2019**)

Examples

- Navier-Stokes equation
  - Continuous time domain: spatio-temporal training data
  - Goal is to predict pressure field as well as the value of viscosity and density.

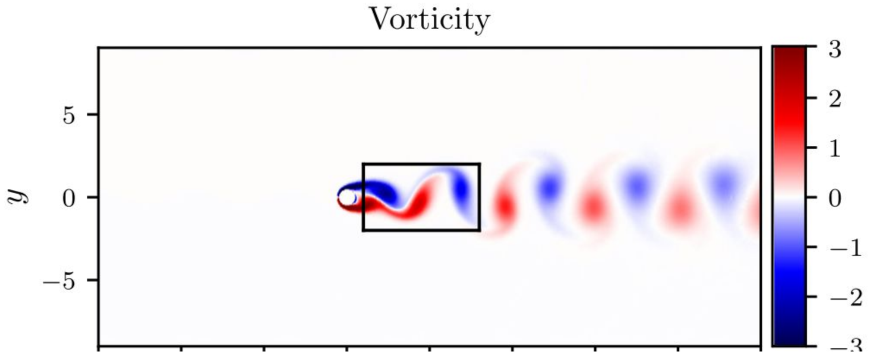
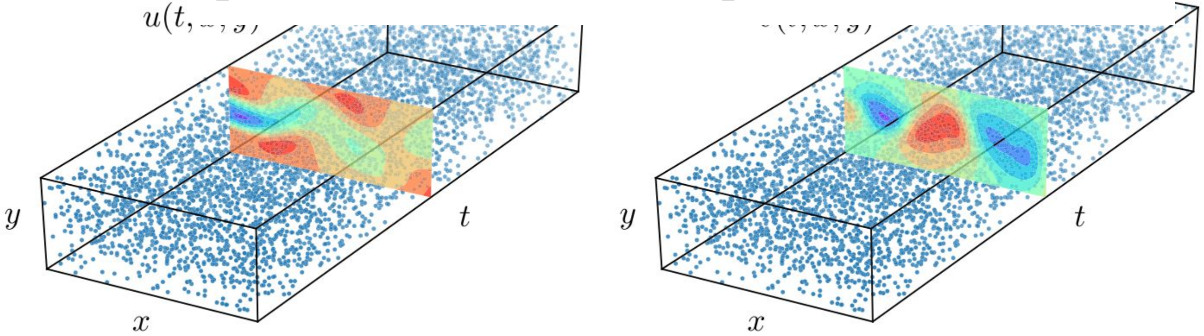


Fig. 3. Navier–Stokes equation [*Raissi et al. 2019*]



Raissi *et al.* (**2019**)

Examples

- Navier-Stokes equation

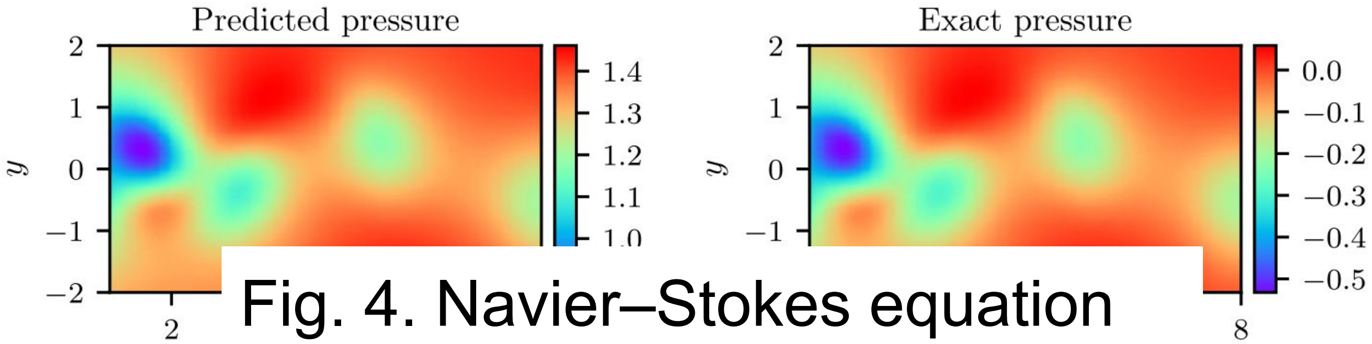


Fig. 4. Navier–Stokes equation [*Raissi et al. 2019*]

Correct PDE	[illegible] $v_t + (uv_x + vv_y) = -p_y + 0.01(v_{xx} + v_{yy})$
Identified PDE (clean data)	$u_t + 0.999(uu_x + vu_y) = -p_x + 0.01047(u_{xx} + u_{yy})$ $v_t + 0.999(uv_x + vv_y) = -p_y + 0.01047(v_{xx} + v_{yy})$
Identified PDE (1% noise)	$u_t + 0.998(uu_x + vu_y) = -p_x + 0.01057(u_{xx} + u_{yy})$ $v_t + 0.998(uv_x + vv_y) = -p_y + 0.01057(v_{xx} + v_{yy})$

Raissi *et al.* (**2019**)