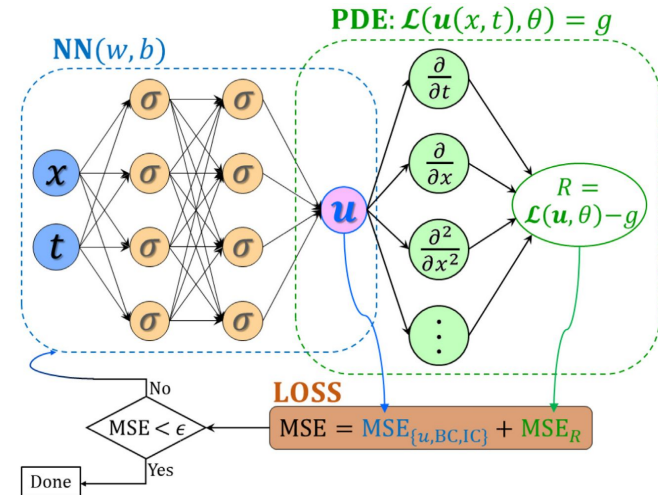


Physics(PDE)-Informed Neural Networks

Mohammad R. Hashemi

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



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*

- Brief history
- Implementation
- Examples

Some references:

- <https://www.sciencedirect.com/science/article/pii/S021999118307125>
- 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.aa.w4741>

2. Simple TensorFlow *programs*

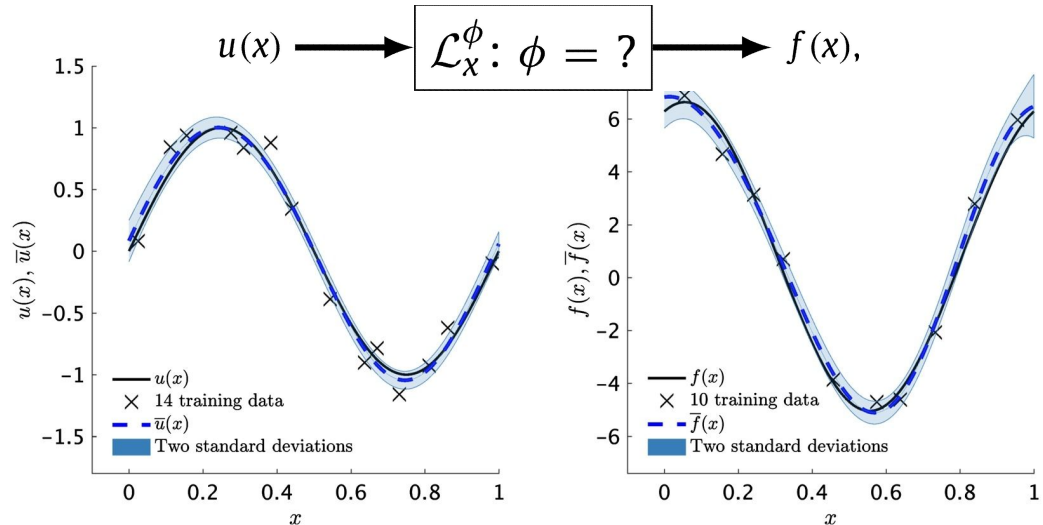
- Automatic differentiation
<https://colab.research.google.com/drive/1xYvR6-KkWLrh0e5Zx3--jSbvWxQJS20?usp=sharing>
- Solving an ODE
<https://colab.research.google.com/drive/1dn1n8tiON6npHRJOcbb9stbtpo7GNDiO?usp=sharing>
- (Inverse) time-dependent diffusion
https://colab.research.google.com/drive/1dnYw1k_mek8bbellwTH3VCOFynajCfa7?usp=sharing

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**)



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.]

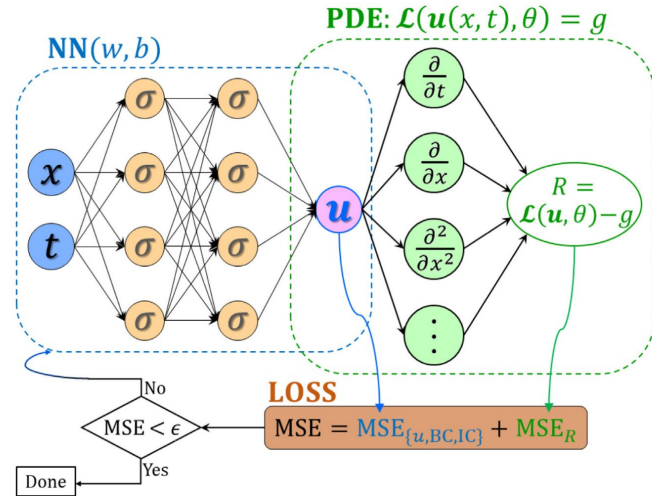
*“Standard multilayer feedforward networks are capable of **approximating any measurable function** to any desired degree of accuracy. There are **no theoretical constraints** for the success of feedforward networks. Lack of success is due to inadequate learning, insufficient number of hidden units or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



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).

$$\begin{array}{lll}
 \mathcal{L}u = f, & x \in \Omega & \text{Neural network:} \\
 \mathcal{B}u = g, & x \in \partial\Omega & u = u_a(x, \beta_i)
 \end{array}
 \quad \text{Minimize to find the parameters:}$$

$$h = \int_{\Omega} \| \mathcal{L}u_a - f \|^2 dV + \int_{\partial\Omega} \| \mathcal{B}u_a - g \|^2 dS$$

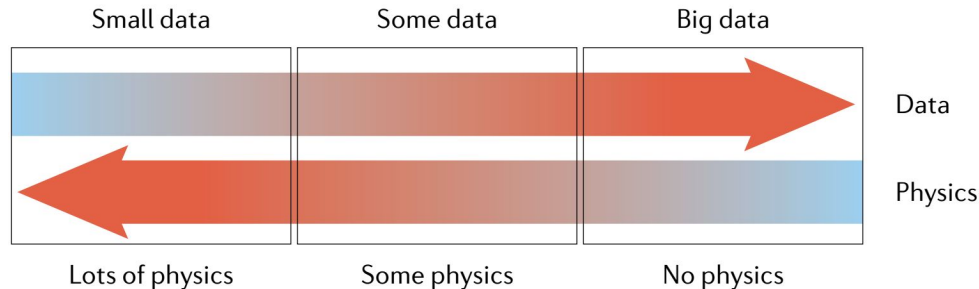
- **Lagaris, I. E., Likas, A., & Fotiadis, D. I. (1998).** Artificial neural networks for solving ordinary and partial differential equations. IEEE transactions on neural networks, 9(5), 987-1000 (1337 citations).

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

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

Implementation

- Assume that we have
 - Neural network: $u(x, t) = \mathcal{NN}(x, t; \theta)$ physics: $\mathcal{L}[u(x, t); \phi] = 0$
- What is the main aim?
 - Parameters are fixed, find unknown “u”.
 - Some observations are available, identify the system (find parameters): inverse problem



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: $\mathcal{L}[u(x, t); \boldsymbol{\phi}] = 0$
 - $\boldsymbol{\theta} = \{\mathbf{W}^j, \mathbf{b}^j\}_{1 \leq j \leq l}$
 - Expanded: $u(x, t; \boldsymbol{\theta}) = \mathbf{W}^l \sigma(\mathbf{W}^{l-1} \sigma(\mathbf{W}^{l-1} \dots \sigma(\mathbf{W}^1 [x \ t]^T + \mathbf{b}^1) \dots) + \mathbf{b}^{l-1}) + b^l$
 - Standard for NNs: $\frac{\partial \text{MSE}(\mathbf{X}; \boldsymbol{\theta})}{\partial \theta_i}$ needing: $\frac{\partial u(x, t; \boldsymbol{\theta})}{\partial x}, \frac{\partial u(x, t; \boldsymbol{\theta})}{\partial t}, \text{etc.}$
- 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

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

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

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

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

Examples

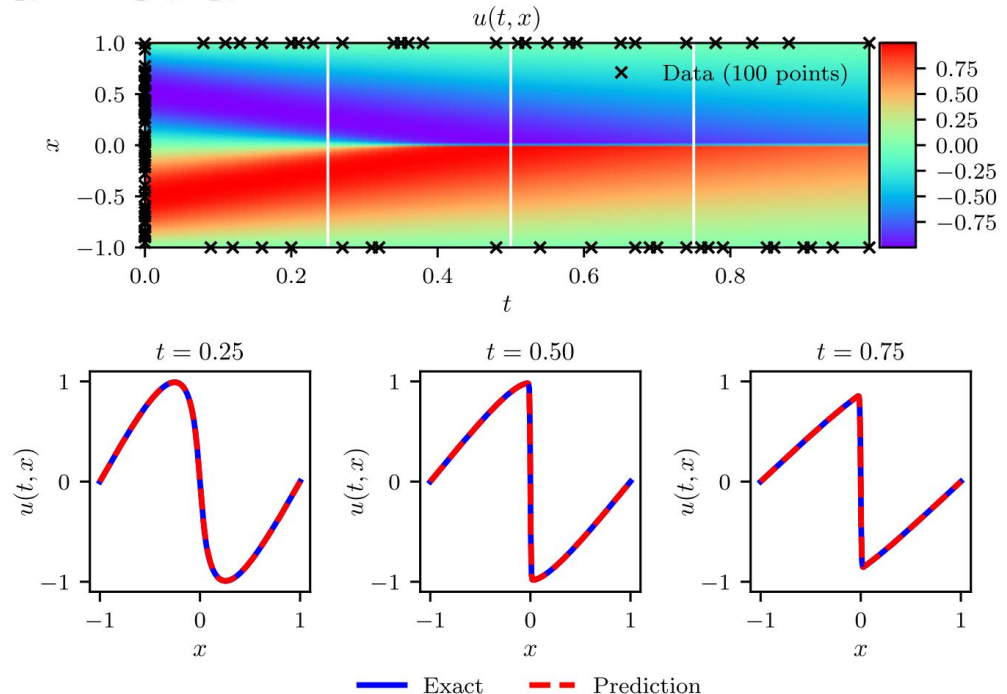
- Burger's equation

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

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

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

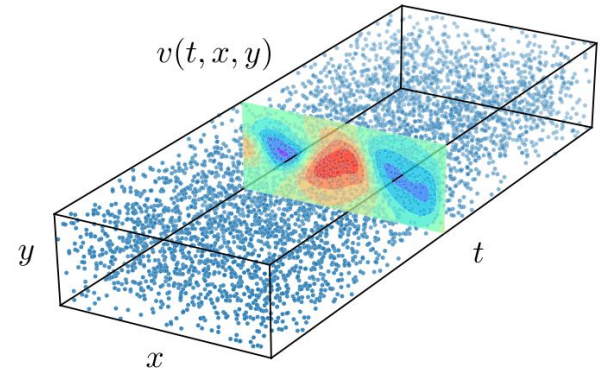
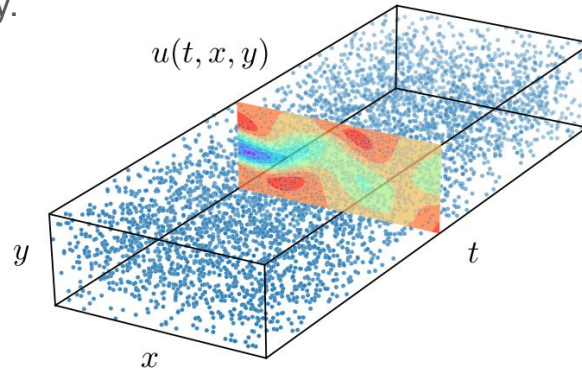
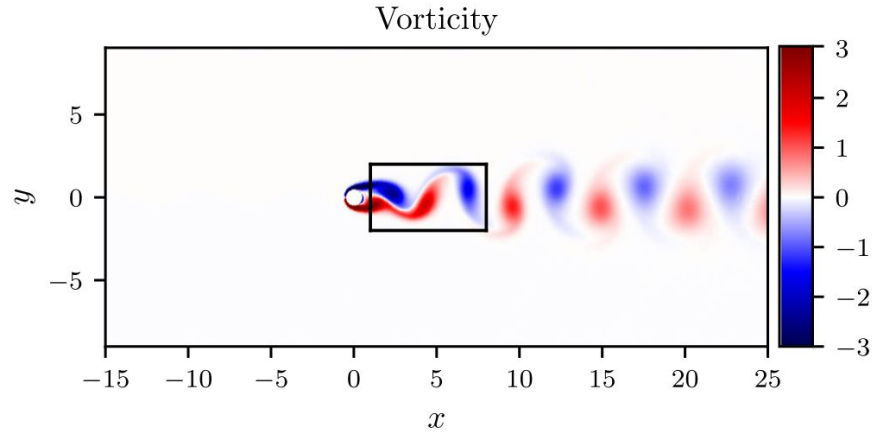
- Continuous time domain:
spatio-temporal training data
- 10,000 collocation points
- 100 data points



Raissi *et al.* (2019)

Examples

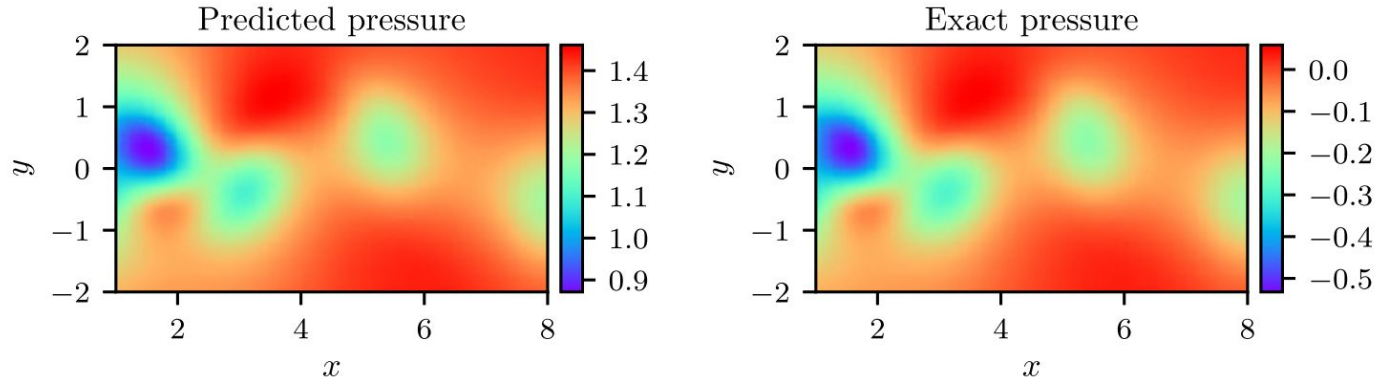
- Navier-Stokes equation
 - Continuous time domain: spatio-temporal training data
 - Goal is to predict pressure field as well as the value of viscosity and density.



Raissi *et al.* (2019)

Examples

- Navier-Stokes equation



Correct PDE	$u_t + (uu_x + vu_y) = -p_x + 0.01(u_{xx} + u_{yy})$ $v_t + (uv_x + vv_y) = -p_y + 0.01(v_{xx} + v_{yy})$
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)	$u_t + 0.998(uu_x + vu_y) = -p_x + 0.01057(u_{xx} + u_{yy})$ $v_t + 0.998(uv_x + vv_y) = -p_y + 0.01057(v_{xx} + v_{yy})$